Game of Life
Report

Shamil Shukurov
Vugar Mammadov

Introduction

The **Game of Life**, also known simply as **Life**, is a cellular automaton devised by the British mathematician John Horton Conway in 1970. It is a zero-player game, meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves.
The goal of the project is to create a program for Game of Life in C. Multiple versions have been developed for achieving the goal.

Game of Life v1.0 : Manual

In this version of the game, the goal is to build the board for playing not randomly but manually. And the reason was that, we wanted to test our game logic manually first, with the different popular configurations like Glider, Beacon, Toad (one can find lots of them in wikipedia).
We used simple structure for board, in which grid sizes and the 2D matrix itself stored.

```
typedef struct board{
   int nrow;    /*!< nrow - number of rows in the board */
   int ncol;   /*!< ncol- number of columns in the board */
   int** gameGrid; /*!< gameGrid - 2D matrix as a representative of game
board */
}Board;
```

In the board each cell can be either dead or alive:

```

typedef enum{DEAD,ALIVE} cellState;
```

Cell's next state is determined based on the number of alive neighbours of it, and since we are encoding dead as 0 and alive as 1, we can just add all the neighbours, therefore we will have the count of the alive neighbours.

```

int aliveNeighboursCount(int row, int col, Board* b, bool isCircular);
```

When we want to find the next state for the whole board, we are reading from the board at time t, and writes the next states to another board for time t+1. For achieving this, we implemented the function:

```
Board* board_t1(Board* board_t, bool isCircular);
```

We're almost done. We just need the game loop :

```
void game(Board* b_t, int maxIter){
   print_matrix(b_t->gameGrid,b_t->nrow,b_t->ncol);
   if(deadBoard(b_t) || maxIter==0) return;
   Board* b_t1 = board_t1(b_t);
   sleep(1);
   return game(b_t1, maxIter-1);
}
```
The above function takes the board at time t as a parameter, prints it, and calls itself with the board for the time+1, if all cells are not dead and maximum iteration number has reached.
Note that we have a lot of other functions and you can read about them by looking at the doc file we have generated by using doxygen. The logic of the game is almost complete in these step, but we need to add new features, modify our code for better performance, rewrite some of the bad written code. So we have next versions.

Game of Life v1.1 : Random

In the version v1.0 we had to enter starting configuration manually, now our goal is create a version where starting configuration is random.
Main change :

- For filling the board randomly new function added and by using the function random version created:
  ```
  void fill_matrix_random(int** matrix, int nrow, int ncol);
  ```

Other changes:

- ANSI codes added for better looking boards
- 5 new macros defined for constant values : default number of rows, default number of columns, default timeout between displaying the boards etc.,
- In the previous version we never deleted the old boards and these boards occupied a lot of memory. So new function created for deleting the board and used in every iteration:
  ```
  /**
  * @brief Deletes the board and frees the memory
  * @param b: Board which we wanted to delete
  * @return void
  **/
  void deleteBoard(Board* b);
  ```
- In a lot of function we passed Board pointer as a parameter, but we never thought the case of NULL pointer, we don't want segmentation faults in our program, so in necessary functions we added the condition for this, e.g:
  ```

  if(b==NULL) return;
  ```

Game of Life v2.0 : Libs

Up to this point we worked on one library. It would be nice if we split our development. So we took the game loop function which draws the board to the console and created another library for it.
Other Changes:

- ANSI code modified, so that dead cells have no background, alive cells have now magenta color.
- Added feature: When executing the code, user now can specify width, height, and timeout with the command line arguments
- Modification: Count neighbour function modified to a cleaner and easier version.

Game of Life v3.0 : Circular Board

Up to this point our board is clipped, it has a fixed row and column size specified by the user, and if the cell reaches the boundary, nothing happens. For example when the glider reached the boundary, since our board was not growing, it was transformed to the block at the corner of the board.
We can make our board circular so that the cell always has 8 neighbours and if the cell reaches the boundary it can come from the other end of the board. In this version we've achieved this. To do that we modified `aliveNeighboursCount` function, we add a new parameter to it : `bool isCircular`. If it is true we count neighbours differently. And note that we also add this parameter to the functions which uses `aliveNeighboursCount.` Actually the first idea of ours it that we can add new field to the Board structure, but it seemed more time consuming and also we don't need the version of the board in many functions, so it would be a memory waste.