John Conway's Game of Life Project

Introduction

Abstract

The Game of Life is a cellular automaton defined in 1970 by the English mathematician John Conway in order to try to solve a problem raised by a mathematician, father of computer science, John Von Neumann.

In a literal sense of the word, it isn't a game, as in it does not require the intervention of a human player, except to set the initial conditions of the cells. It consists of a universe in which live cells evolve according to precise evolution rules. In the original version described by John Conway, the universe is defined on a two-dimensional grid, of variable size, where each cell is a cell that can take two different states: dead or alive. The switch from one state to another is determined by the following rules:

• A dead cell at time t becomes alive at time t + 1 if and only if it has exactly 3 living cells in its vicinity.

• A living cell at time t remains alive at time t + 1 if and only if it has exactly 2 or 3 living cells in its vicinity, otherwise it dies.

• All other cells die.

• The neighborhood used is the 8-neighborhood: for a given cell, its neighbors are the 8 cells that surround it.

Objectives

The implementation will be developed in 4 steps, and a final step combining the features from all previous steps.

• Step 1

In this step the program just asks from user to enter the number of rows and columns to initiate a grid, and then asks for the number of alive cells, and then makes random cells alive. This step just creates random board, there is no evolution.

• Step 2

In this step the program totally works in circular mode, and as in step 1, user should enter ne size of grid and the number of cells, the game works in the terminal by using ANSI code, and the game can evolve by clicking enter key.

• Step 3

In this step the clipped mode was created, and the program totally works in both modes, and as in previous steps, user should enter ne size of grid and the number of cells, the game works in the terminal by using ANSI code, and the game can evolve by clicking enter key and the user can change the mode of the game by clicking to c key.

• Step 4

In this step the game was written in SDL mode, all of the previous game features still work in this step. A new window will be running the game, while in the terminal the information about the game will be written. To evolve, the user should click to n key, to change the mode user should click to c key, to quit user should just close the window.

• Final

The final step is the main step of this project, where the final version of game with 2 modes (circular, clipped) are written in both terminal mode and SDL mode. Additionally, now the user can choose: whether choose a specific file to generate the game and make it more interesting, or just enter the size of the grid and the number of alive cells which will create a random board.

Implementation

Step 1

The aim of this step is to visualize the board in which the game will be running. This will be done by implementing a simple grid structure, and Boolean logic to determine if a given cell in an integer array grid is dead or alive. The grid structure also keeps the age of the grid (turn), number of columns and number of rows.

Initialized in the game.h header file, the “game” module of this step is responsible for the creation, initialization and memory control of the grid structure using the allocate_grid, free_grid, init_grid functions. It can also alter the state of individual cells using the set_alive and set_dead functions by setting their integer values to 0 or 1. State of an individual cell can be checked by the is_alive function.

Visualization part of the first step happens in the terminal using the ANSI codes to help illustrate the grid. Initialized by drawer.h, the “drawer” module of this step is responsible for projecting the grid on the terminal by utilizing ANSI code. In the main utilization part of this module, the set_grid function takes a grid structure as a parameter and projects onto the terminal, and the start_game function takes the address of a grid and loops in a while to advance the turns while the user input doesn’t declare otherwise.

The drawer module also utilizes several auxiliary function regarding cursor and symbol manipulation through ANSI code (clear_screen, hide_cursor, show_cursor, move_cursor_to, set_color, reset, delete_ecran).

Step 2

The objective of this step is to ensure that the grid function through a clipped, meaning the neighboring cells of edge cells are not overlapped through the opposite edges. This step also includes the evolution process of given cells.

Changes are made from the previous iteration of the “game”, adding functions for checking neighbors and evolution, and an additional utility function to copy a grid to another grid given as a parameter.

The added check_neighbours function returns the number of alive cells among the neighbouring(8 , circular overlapping included) cells.

The added evouluation function is the main logic part specified for the game. It is utilized for applying the initial game rules to all cells in a grid, using the previously mentioned function to check all neighbours of a cell in a circular manner.

Step 3

The objective in this step is to ensure that the grid checking functions in both circular and clipped mode.

The additions in this version, check_neighbours function from the previous step was seperated into check_neighbours_circular and check_neighbours_clipped functions, adding the circular system. This way it can be specified in which manner the program needs to check for neighbor cells.

This is done by changing the evoluation function to take a function pointer specifying which of the check_neighbours functions to utilize. This paramater is given in the start_game function located in the “drawer” module (pointer currently set to clipped).

Step 4

The aim of this step was to change the program to work in SDL mode, giving access to work with much more cells than before. The drawer module is removed, putting the cell rendering all through SDL mode. This step consists of a single module “game_sdl” and includes both memory handling and displaying the cells.

In addition to the “game_sdl” module from the “game” module from previous versions, the init_grid_from_file function is added which is utilized for setting a grid map specified through a file.

Added display_grid function is responsible for displaying information about the game through the console, initializing the parameters for the potential SDL render, i.e drawing the foundation white space, the row and column lines for the cells, iterating through the grid and rendering a separate color for the alive cells.

Part of the core process is moved to the main file, namely the allocation and the initialization of the grid, specification of a grid fil if there exists one, initializing SDL renderers for the screen and OS window.

Additionally the main program is also responsible for setting the checking mode to circular or clipped, creating the previously initialized renderers for the screen and OS window, looping over the SDL event, preforming checks and running the evoluate function.

Final

In this step , the previous additions made to the program are combined , and tests are made.

Previously explained console and SDL versions are both included in this step, and checks are run through main and main_sdl programs.

Additionally in these main files, one of the included grid text files can be included to test interesting cases for the game.

Additionally a ctest file is included for circular and clipped testing of the algorithms.