Fundamentals of Computer Programming:

Project Report

Conway's Life

Author:	Rafał Potempa
Supervisor:	Piotr Fabian, Ph.D.
Date:	16 January 2018

Rafał Potempa, Gliwice, 2018-01-16

Table of content

1. Task description

Task: 'Conway's Game of Life'

One of the first and most known cellular automatons devised by the British mathematician John Conway in 1970.

Evolution of system of cells given depending on initial conditions, that requires no further input. Rules governing the 'life and death' of each cell, depending only on neighborhood of this cell, are applied every generation and are following:

If cell has:

• 0-1 neighbors	it dies (of isolation) or remains dead,
• 2 neighbors	it remains as it is,
• 3 neighbors	if it is dead, it becomes alive,
• 4 or more n.	it dies (of congestion).

As stated the rules are applied every generation, the system is recalculated, the rules are applied again and so on.

2. Solution analysis

The program needs several variables types to optimize the data flow.

The solution uses Allegro library, version 5.2.3.1, to display the board state in graphical way, white for living cells, and black for dead.

The array which stores the system board with cell status stores state of cell, described in code as '0' or '1', for ('dead' and 'alive' respectively) is dynamically allocated **`char`** 2D array. **`char`** variable type is used because it occupies only one byte of memory per entry.

The program on its start checks if input file exists. If result is positive it opens it and copies its content into array, filling empty cells with zeros, otherwise a proper message is displayed. Next the program prints the board to the console with living cells as '**`1`**' and dead as '**`0`**'.

Next the program initializes the Allegro library with the screen showing board state. Pressing [enter] displays next generation, which is now calculated. Program continues as pressing [enter] displays new generations. Pressing anything else + [enter] results in program termination.

3. Internal specification

3.1. Designed regular functions

`int checkFile(char* path)`

checks if file at given **`path`** exists, if doesn't writes message.

`int checkBoardSize(char* path, char mode)`

returns number of columns or rows, depending on **`mode`** (**`'c'`** or **`'r'`** respectively). When number of columns in each row is not equal, function returns length of longest row, displaying a warning.

`void getFromFile(char* path, char** list, int rows, int cols)`

checks the content of file at path directory and if entities are '**0**' or '**1**' copying them to list array, for invalid entities '**0**' is applied. Meanwhile function checks the length of rows and fills missing row end entities with '**0**'s.

`void printArray(char** list, int rows, int cols)`

prints array into a console.

`int checkNeighborhood(char** list, int rows, int cols, int i, int j)`

returns number of neighbors of cell **`list[i][j]`**.

`int** neighborhoodArray(char** list, int rows, int cols)`

returns pointer to new array where each cell is number of neighbors of this cell in **`list`**.

Remark:

`void lifeAndDeath(char** list, int rows, int cols)`

checks *Conway's Life's* rules for each cell, and then replaces old array **`list`** with recalculated one. Old array is destructed in this process.

3.2. Designed Allegro functions

`void drawRectangle(float x1, float y1,float x2,float y2, ALLEGRO_COLOR color)`

draws on display rectangle of color **`color`** with opposite corner coordinates (**`x1`**,**`y1`**) and (**`x2`**,**`y2`**).

`void drawBoard(char** list, int rows, int cols, double size)`

draws status of entities in **`list`**, as white squares of size **`size`** pixels, for living cells and brings the display form buffer, to screen.

4. External specification

This program is designed to work with command line prompt and to display the results also in graphical way.

Program uses by default <board.txt> file as input, which should be contained in .exe directory before running.

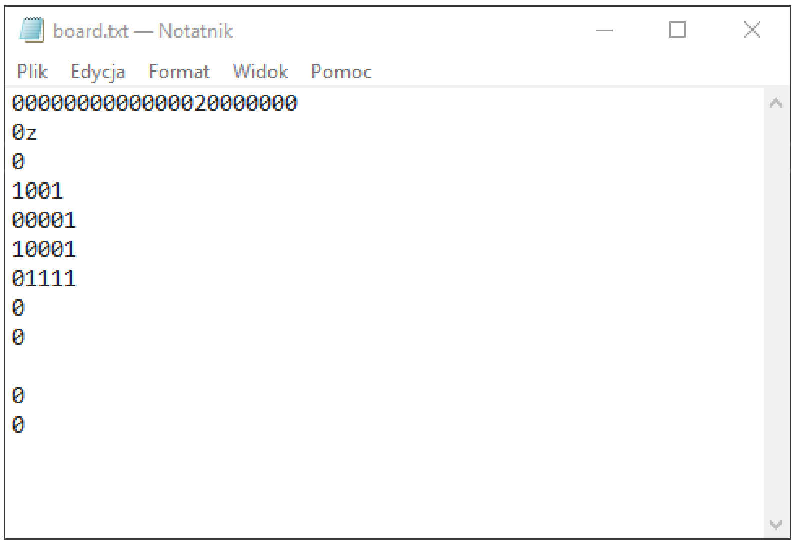
<board.txt> file should be written as rows '0's and '1's not separated by any operator. '0' for dead cell, '1' for living cell.

5. Testing

5.1. Ordinary conditions

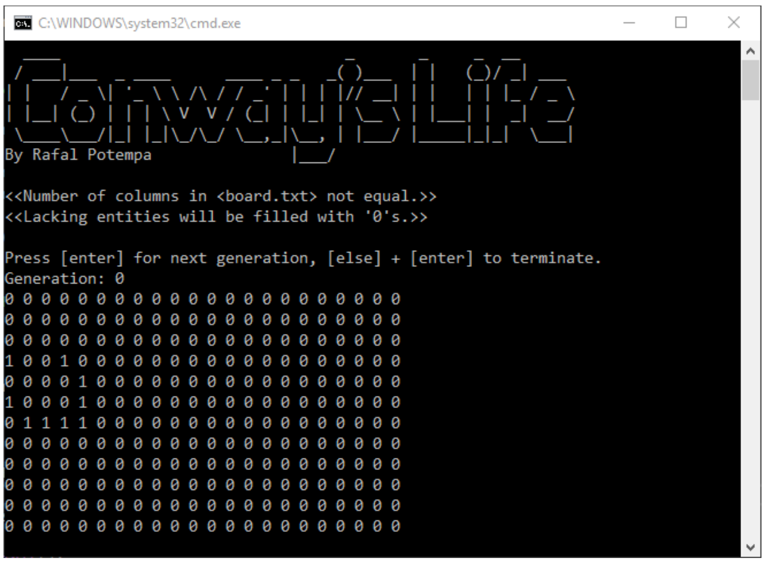
Remark: display animation attached to project.

Input conditions:
(text file which is poorly formatted and contains improper symbols)

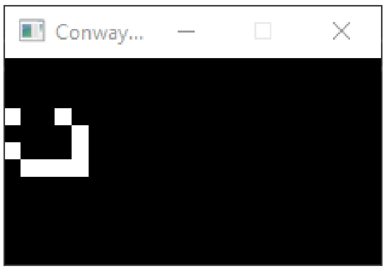


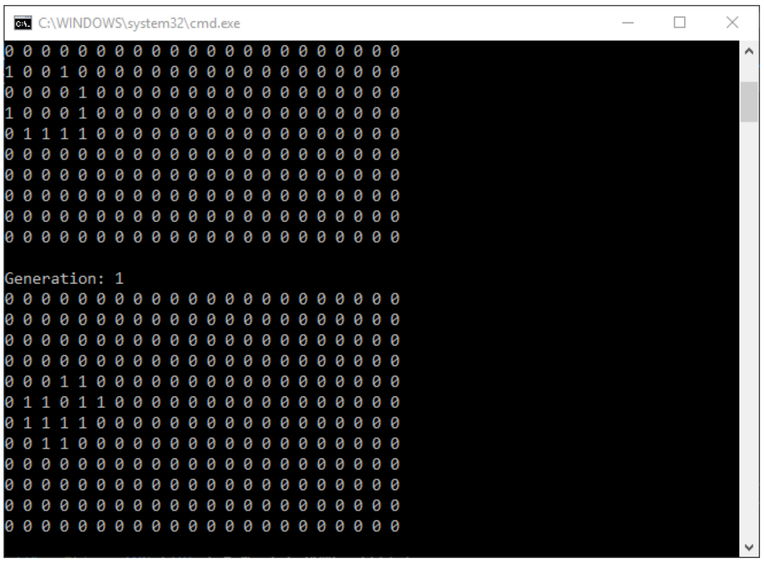
```
0000000000000020000000
0z
0
1001
00001
10001
01111
0
0

0
0
```

Console and display view for given input:

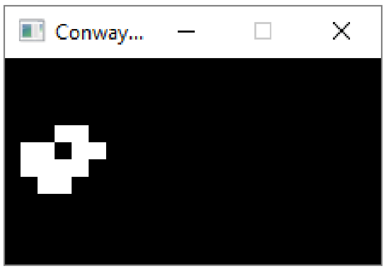

C:\WINDOWS\system32\cmd.exe

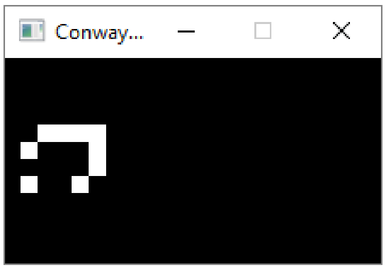
```
  ____                                _       _     _  __
 / ___|___  _ __ __      ____ _ _   _( )___  | |   (_)/ _| ___
| |   / _ \| '_ \\ \ /\ / / _` | | | |// __| | |   | | |_ / _ \
| |__| (_) | | | |\ V  V / (_| | |_| | \__ \ | |___| |  _|  __/
 \____\___/|_| |_| \_/\_/ \__,_|\__, | |___/ |_____|_|_|  \___|
By Rafal Potempa                |___/

<<Number of columns in <board.txt> not equal.>>
<<Lacking entities will be filled with '0's.>>

Press [enter] for next generation, [else] + [enter] to terminate.
Generation: 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

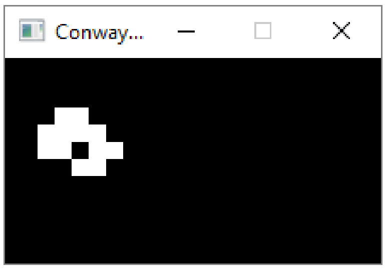


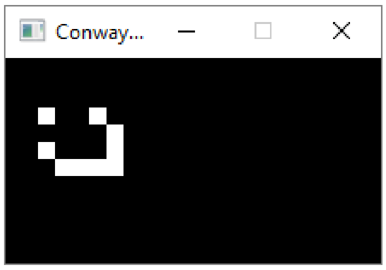

C:\WINDOWS\system32\cmd.exe

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Generation: 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

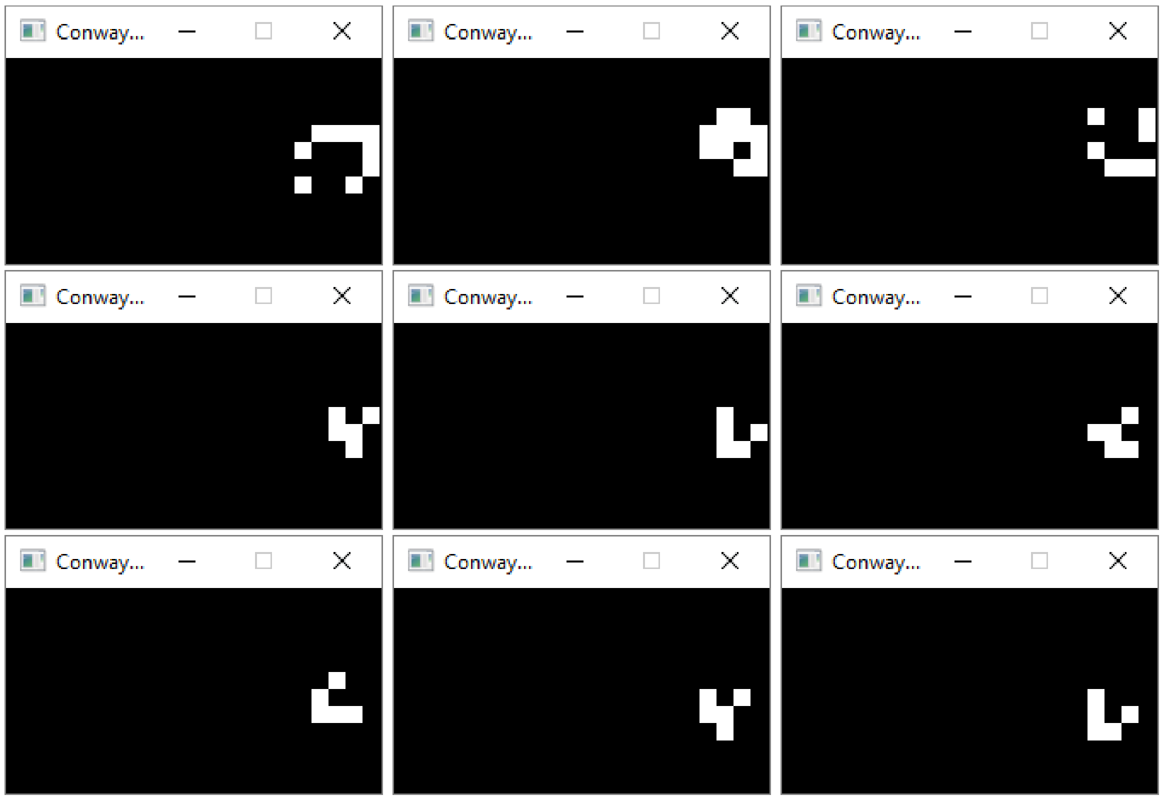




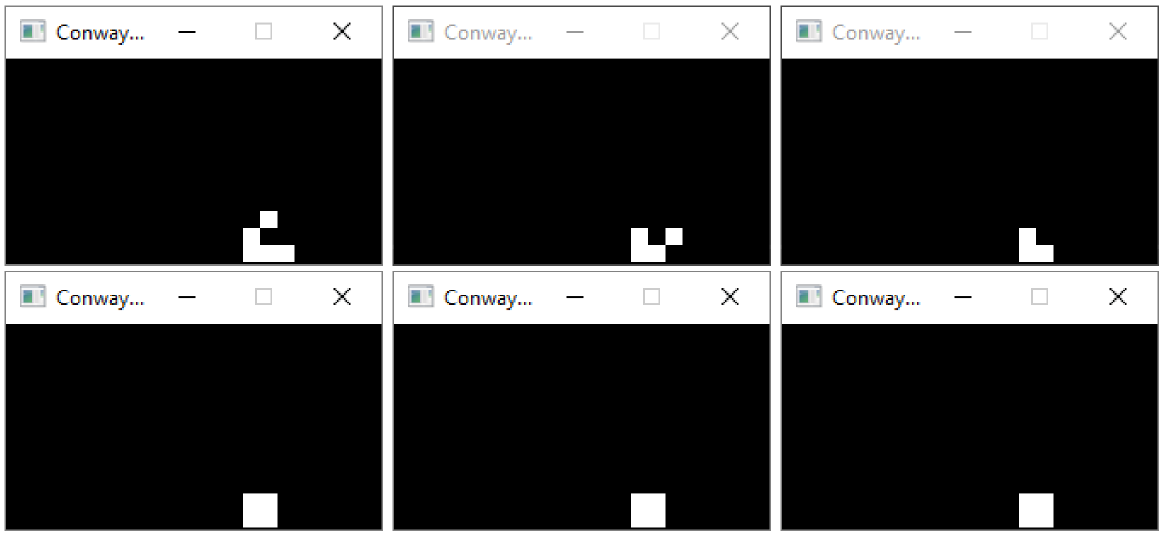




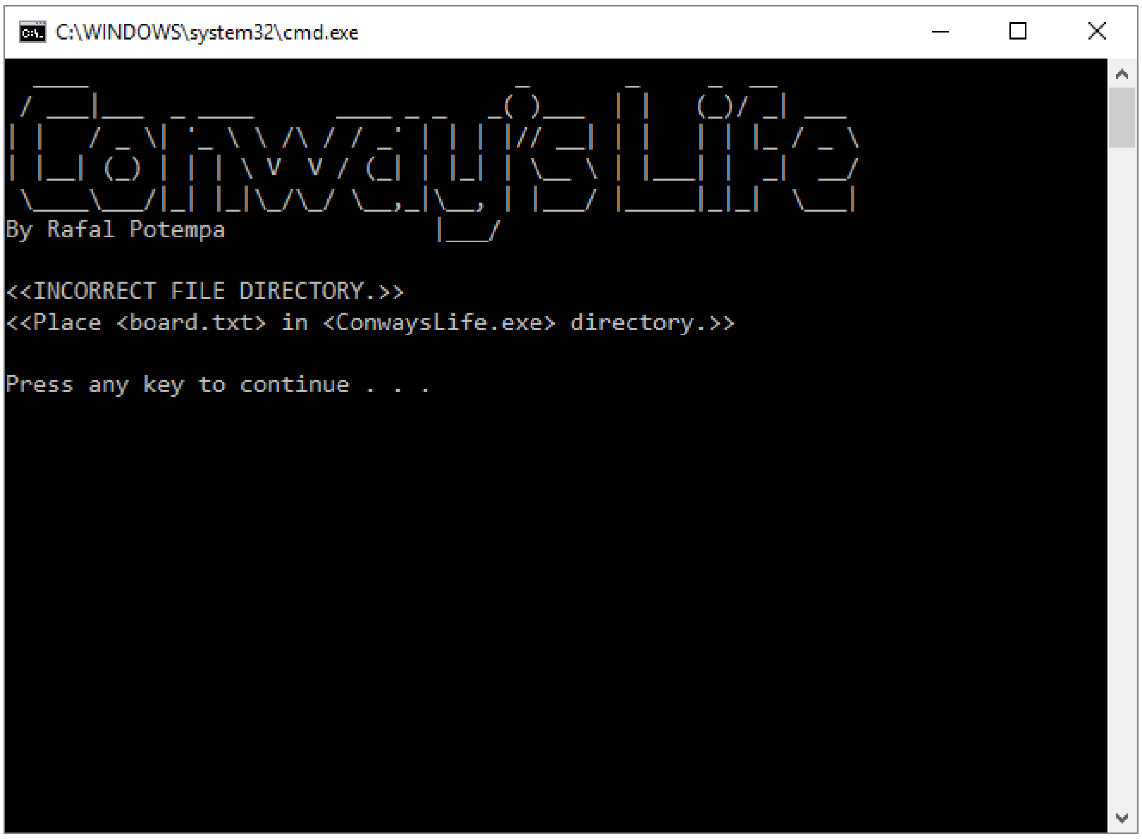
(...) Generation 34



(...) Generation 56



5.2. No input file in directory



```
  ____                                 _       _     _  __
 / ___|___  _ __  __      ____ _ _   _( )___  | |   (_)/ _| ___
| |   / _ \| '_ \ \ \ /\ / / _` | | | |// __| | |   | | |_ / _ \
| |__| (_) | | | | \ V  V / (_| | |_| | \__ \ | |___| |  _|  __/
 \____\___/|_| |_|  \_/\_/ \__,_|\__, | |___/ |_____|_|_|  \___|
By Rafal Potempa                 |___/

<<INCORRECT FILE DIRECTORY.>>
<<Place <board.txt> in <ConwaysLife.exe> directory.>>

Press any key to continue . . .
```

6. Conclusions

- board is generated from file,
- all calculations are properly performed,
- program flow is correct,
- simulation is stable and sustainable,
- graphics displays properly.