London Metropolitan University

Faculty of Life Sciences and Computing

CU6051 Introduction to Artificial Intelligence

Emergence: Cellular Automata

Documentation

Names: , Denzel Dadson

ID Numbers: , 11062511

Date: Friday, 24 January 2014

# Contents

[1 Communication 1](#_Toc378330566)

[1.1 Deciding our group work 1](#_Toc378330567)

[1.2 How we communicated when it came to the work 1](#_Toc378330568)

[1.3 Presentation 1](#_Toc378330569)

[2 User Documentation 2](#_Toc378330570)

[2.1 Conway’s Game of Life 2](#_Toc378330571)

[2.1.1 Plain Mode 2](#_Toc378330572)

[2.1.2 Tab: Task 3](#_Toc378330573)

[2.2 Wolfram’s Rule 30 3](#_Toc378330574)

[3 Technical Documentation 4](#_Toc378330575)

[3.1 Code Organization 4](#_Toc378330576)

[3.2 Graphics 5](#_Toc378330577)

[3.2.1 Drawing the grid on canvas 5](#_Toc378330578)

[3.3 Conway’s Game of Life 5](#_Toc378330579)

[3.3.1 Data structures and main algorithm 5](#_Toc378330580)

[3.3.2 Send Gliders Mode 6](#_Toc378330581)

[3.3.3 Create Glider Mode 6](#_Toc378330582)

[3.4 Wolfram’s Rule 30 9](#_Toc378330583)

[4 Conclusion 11](#_Toc378330584)

[5 References 12](#_Toc378330585)

[6 Appendix 13](#_Toc378330586)

[6.1 List of files submitted 13](#_Toc378330587)

# Table of Figures

[Figure 1 - Code Organisation 4](#_Toc378330604)

[Figure 2 - Cells to Pixels Transformation 5](#_Toc378330605)

[Figure 3 – Pixels to Cells Transformation 5](#_Toc378330606)

[Figure 4 – Next Generation 6](#_Toc378330607)

[Figure 5 - Send Gliders Mode 6](#_Toc378330608)

[Figure 6 - Create Glider Mode 7](#_Toc378330609)

[Figure 7 - Cartesian coordinate system – Quadrants [3] 8](#_Toc378330610)

[Figure 8 - Glider rotation 8](#_Toc378330611)

[Figure 9 - Rule 30 Table 9](#_Toc378330612)

[Figure 10 - Rule 30 Pseudo Algorithm 9](#_Toc378330613)

# Communication

## Deciding our group work

We decided to make a 2-D cellular automata game called ‘Game of Life’; this is because looking at the module specifications we came to a conclusion that the ‘Game of Life’ was suitable for a group work. After researching the ‘Game of Life’ on the internet we picked up some valuable points about the game and how it works. The man behind this game is called John Horton Conway and he invented the Game of Life in the late 1960s at the University of Cambridge. The objective of the game is to make a cellular automata game which is as unpredictable as possible using the simplest possible cellular automata rule. The Game of Life is not like other games that has a winner or loser. Once the "pieces" are placed in the starting position, the rules determine everything that happens later. In most cases, it is impossible to determine what will happen in future by looking at the starting position. The only way to find out is to follow the rules of the game.

## How we communicated when it came to the work

We decided that we should post pieces of the group work that we had completed on GitHub. This was a good idea, because work that has been lost can be easily gained back. For example, work saved on a memory stick that becomes lost, stolen or corrupt is difficult to retrieve however using GitHub the work would be saved on external server therefore making it easy to retrieve.

GitHub also allows more than one person to have access to work at any time. This is good for group work as it makes it easier for all the group members to have access to all the codes and data used for the programming of the game. This increases the co-operation of the group and makes the team work more efficiently.

We met twice a week to discuss how the work would be completed, having researched about certain specific topics i.e. what cellular automata is and how it can be used particularly on the “Conway’s Game of Life”. Further research was made on the gliders and how they function. We also had to find out how the game works so further research was done.

## Presentation

We have agreed together to create presentation which content would discuss how cellular automata can simulate patterns in nature. What inspired us on biological processes is the possibility to recreate patterns and designs created by nature. There is also a practical side to this project. Patterns generated by cellular automata may well be utilized for textile designs [4].

# User Documentation

This section describes the Graphic User Interface and its functionalities of developed web application. The web application presents itself by drawing a grid with dimensions matching the browser window size. The user interface is divided into two dialog windows. Following chapters describes operations of controls for each window.

## Conway’s Game of Life

Both dialog windows, Conway’s Game of Life and Wolfram’s Rule 30, use Tabbed Document Interface, called “Tabs”, which allows multiple documents to be contained in a single window. Conway’s Game of Life window contains Home, Patterns, Graphics and Task tabs. Each tab is composed of group of menu buttons. Following sections in chapter 1.1 describes the main modes of the web application. Application controls are explained through the description of modes.

### Plain Mode

Tab Home has four groups of controls: Selection Mode, Navigation Controls, Speed and Stats. “Selection Mode” group contains three radio buttons, which enable switching between three different game modes. The default mode is “Plain Mode”. In this mode user can use “Navigation Controls” to start and stop the game, or step forward through the game by pressing button “Next Generation”. Grid can be cleared using button “Clear”. Speed of redrawing the grid can be manually changed using the slider or by entering the value instead of the orange number and pressing enter. The last group contains non-editable statistics about the game, for example number of past generations.

This mode allows loading of patterns using drop-down list from tab “Patterns”. Application provides 34 patterns. All patterns have a word “glider” in the name.

Tab “Graphics” allows user to resize the grid using spinner. Spinner generally enhances a text input for entering numeric values with up/down buttons and arrow key handling. In addition to this functionality user can manually rewrite the numbers. By pressing ENTER user confirms his attention to resize the grid. The current version of this application initializes the grid on every resize event. That causes the loss of the current pattern on the grid.

The last feature is “Display Mode”, where user can select between four different display modes. The default mode is “Rectangle”, which displays rectangles with white inside having black border for dead cell, and black inside for a live cell. Mode “Circle” draws blue circles inside rectangle when cell is alive. “Trail” mode redraws the grid with blue colour. Living cells appear blue, dead cell has white inside and blue border. Orange border of dead or alive cell indicates that that particular cell has been previously alive. This mode displays the movement of live patterns. Mode “Trail without grid” is similar to “Trail” mode, except that it does not draw grid for Universe, but draws orange grid for previously visited (alive) cells. In this mode user can see more clearly the shape of the area which has been created by pattern.

#### Send Gliders Mode

This is the first half of the solution to the given task. User can select starting points, or patterns and system continuously sends gliders. If Universe does not contain live cells on first and every 14th generation, system does not send gliders. When Universe is alive, gliders emerge starting from upper left corner and continue to emerge up to upper right corner. Every 14th generation a new glider is drawn moved by 7 cells horizontally. The same way and in the same time gliders emerge from bottom. System this way creates chaos.

#### Create Glider Mode

“Create Glider” mode is a second part to the solution. When game is switched to this mode, 15 random shapes on random positions are added to the next generation. This shapes are displayed on the grid together with message which invites user to set starting points. In the next generation a glider or lightweight spaceship (small fish) emerges from every starting point. Glider moves diagonally at a quarter of the speed of light. The lightweight spaceship moves horizontally or vertically at half the speed of light. Glider or spaceship have directions to the closest live structure (organism). If there are more than two closest organisms to the starting point, algorithm selects direction to one of these organisms randomly.

### Tab: Task

This tab describes the task. Task is to create an interactive implementation of the Game of Life grid, where a player can select starting points for a variety of blinkers. The system then sends gliders to destroy them, resulting in chaos. Alternatively, system randomly sets some blinkers and player sets starting point for gliders. Documentation should clearly explain algorithms used to generate the life-forms (chapters 2.1 and 2.2).

## Wolfram’s Rule 30

Wolfram’s Rule 30 window has two tabs. First tab “Home” has basic “Navigation Controls”, “Speed” and “Stats” groups, which have the same functionality as in Conway’s Game of Life window. Rule 30 is one dimensional cellular automaton, therefore first row of the grid represents first generation; second row represents second generation, etc. Using this window user can see in slow motion how generations are generated for Rule 30.

Second tab “About Rule 30” partly explains the Wolfram Code and how number 30 becomes the rule number of this automaton.

# Technical Documentation

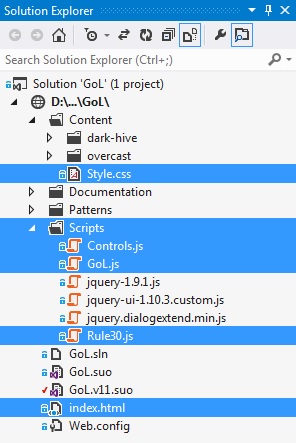
Technical documentation describes algorithms and main thoughts behind the produced solution. Documentation of the solution is introduced by chapter 2.1, Code Organization, which outlines how the code in these files fits together.

## Code Organization

Development of this project has been tracked using Git. Software source code has been uploaded to GitHub server where the functional website is hosted too [1].

Development branch “CleanWebsite” has file structure displayed on Figure 1. Files highlighted with blue colour are files containing all developed source code.

Figure - Code Organisation

File “index.html” is a main file and loads all JavaScript and Cascading Style Sheets. The controls created by main file are managed by JQuery code contained in “Controls.js” and “Rule30.js”. “Controls.js” manages Conway’s Game of Life dialog functionality and “Rule30.js” controls Wolfram’s Rule 30 dialog window. This JQuery code initializes the whole website and activates appropriate functions chosen by the user.

The underlying code which draws onto the canvas, catches the user’s click and executes the algorithms (of interest) is contained in file “GoL.js”, which is a shortcut for “Game of Life”.

The original source code was initially downloaded (forked) from GitHub [2]. Since then most of the code has been changed. At present the only thing remained is the main skeleton of objects, few variables and methods. I have downloaded 34 patterns and kept them within the solution.

## Graphics

### Drawing the grid on canvas

The grid consists of rectangles drawn next to each other. To draw grid on canvas, the cell size had to be chosen. I have decided to use ten pixels per cell’s width and height. During drawing the grid on canvas, the cell coordinates were recalculated to pixel coordinates using equations displayed on Figure 2 below.

Figure - Cells to Pixels Transformation



The Xcell, Ycell variables are coordinates of the cell, starting from [0, 0]. The x, y are variables holding position of starting point from which rectangle will be drawn.

Figure 3 below shows equations for calculating the cell coordinates. These equations were needed for recognizing the position of user’s click onto the grid. Mouse click event returns horizontal and vertical distance in pixels from position [0, 0]. These coordinates are represented by variables x and y on Figure 3.

Figure – Pixels to Cells Transformation



Left offset of canvas (first line on Figure 3) is distance of canvas element from left border of the browser window, and top offset is the distance of canvas from top border. These equations can be found in method “Graphics.GetCoordinates”.

## Conway’s Game of Life

### Data structures and main algorithm

The “GoL.js” file described in chapter 2.1 contains three objects: Cell, Graphics and Life. The Cell object will be described in chapter 3.3.3. The last two objects are singleton objects. Object Graphics has properties and methods which hold information and perform operations related to drawing onto the canvas. Object Life has attributes and methods related to Conway’s Game of Life and Rule 30.

The solution to program the Conway’s Game of Life web application is to keep two states of the game. Except that individual cells can be alive or dead (thus have two states) and the game is on and off, the next generation is always based on the cells before the Conway’s Game of Life rules were applied. That means the object Life has to store two states, previous generation and next generation. Pseudo code of Conway’s Game of Life algorithm is shown below on Figure 4. Name of the method is “Life.nextGeneration”.

When user selects a pattern, the “alive” value is kept in previous generation as Boolean. The next generation is made equal to previous generation (line 2), calculating next generation by Conway’s game of life rules (line 4). Adding life forms to next generation depends on the selected game mode (line 5). After that only cells from next generation which are different from cells in previous generation are drawn (lines 6 to 8). The last step in algorithm is to overwrite previous generation with next generation. The algorithm repeats until the stop button is pressed by user.

Figure – Next Generation



### Send Gliders Mode

This mode automatically sends gliders. Method “Life.userSelection” corresponds to “Send Gliders” mode. Following pseudo code on Figure 5 describes the method.

Figure - Send Gliders Mode



Algorithm checks on lines 3 to 6 if “Universe” contains any living organism. If whole “Universe” is dead, than algorithm ends its execution and nothing will be drawn on canvas. Having any life forms in Universe will cause formation of new gliders, described in chapter “1.1.1.1 Send Gliders Mode”.

### Create Glider Mode

Create Glider Mode is represented by method “Life.automaticSelection”. Pseudo code of the algorithm is displayed on Figure 6 below.

When defining moving pattern, the direction of pattern has to be considered. Glider orientation is in North-West direction, spaceship has west direction. Then rotation matrix is defined (line 4). When multiplicating pattern with rotation matrix, pattern rotates in clock-wise direction around first point within the direction. Further functions “rotate” and “addToNextGen” have been defined, lines 2 and 3. Function “rotate” multiplies pattern with rotation matrix and returns new pattern. Function “addToNextGen” adds pattern to next generation.

Main loop of the algorithm is to walk through the list of starting points and for every starting point a number of actions is executed (lines 8 to 27). First of all a coordinates of starting point are received. After that, algorithm runs a check if neighbours of the point are dead. If neighbours are dead, there is a space to place the pattern on the grid (lines 15 and 16).

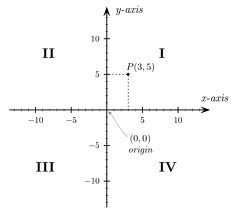
Algorithm is using method “Life.getOrganisms” to obtain the list of live organisms (line 14). This function defines organism as a collection of cells. Every **cell** is an object and has coordinates and survival status as properties. Cell status can be alive or dead. Alive is usually represented as Boolean true or decimal number one in the program. Dead status is False or zero.

Figure - Create Glider Mode



The object which holds all organisms has function which returns the closest cell from all cells in one organism to the starting point. After that a distance between closest cell and starting point is computed and stored in an array (line 20). This array is used to find the minimal distance from all distances. Possibly there can be more organisms in the closest distance. This situation is being handled by choosing randomly exactly one organism (line 23). From this organism a closest cell to starting point is retrieved again. The cell coordinates are used for calculation of direction point, which is used to point the glider or small spaceship into the right direction (line 24). Line 25 summarizes the eight conditionals which serve as rules by which pattern will be rotated once, twice or three times by 90 degrees in clock-wise direction depending in which quadrant the rotation point is located. Figure 7 below depicts the numbering of quadrants. For example, if direction point is located in third quadrant, the glider or small spaceship pattern has been rotated three times.

Figure - Cartesian coordinate system – Quadrants [3]



All directions have been marked as cardinal directions. The starting direction for glider is north-west (quadrant 2). After triple rotation its new direction is south west, which corresponds to quadrant 3. Following figure 8 represents the glider rotation.

Figure - Glider rotation

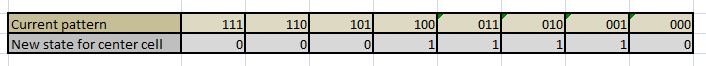
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  | | --- | |  | |  |  |  |  |
| |  | | --- | |  | |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  | | | | |  |  |  |
|  |  |  |  |

The last step of algorithm is to redraw the starting point to the colour of the live cell. The starting point will be a part of the correctly rotated glider. Glider has been placed on position of the starting point.

## Wolfram’s Rule 30

Rule 30 uses infinite one dimensional cellular automaton, which can be represented on computer as one dimensional array of cells. Built structure (two-dimensional array) has been reused to accelerate software development. The whole solution for this rule is incorporated in one method: “Life. nextGenerationRule30”. The “Rule 30” can be boiled down to eight simple rules, which are displayed on following figure 9. These rules are represented by binary numeral system (first row). Every rule (or pattern) has corresponding value in the cell under (second row).

Figure - Rule 30 Table



Binary representation of number 30 is 00011110 (little endian used). This number is composed of all possible configurations for a cell and its two immediate neighbours (left and right neighbour). These configurations have following order: 111, 110, 101, 100, 011, 010, 001, 000, which is a first row of figure 9. First configuration (000) from right **has assigned** right-most bit 0, second configuration (001) has assigned 1, etc. The binary number (00011110) is taken to be the rule number of the automaton.

Pseudo algorithm using this table of rules is depicted on figure 10 below.

Figure - Rule 30 Pseudo Algorithm



Lines 1 to 4 ensure, that drawing on canvas will be stopped, when generation count exceeds the size of canvas in cells vertically. It is important to note that vertical axis represents time. Every next row from top to bottom is a next generation produced by the Rule 30.

Lines 5 to 15 handle main loop of the algorithm within the method. Algorithm walks through all cells in one row. For every cell, value of left neighbour to the cell is multiplied by 4 and the value of cell itself is multiplied by two. Value of right neighbour is left unchanged. These three values have been summed and resulting number corresponds to one current pattern from figure 9. As you can see from the table, only some patterns result in having assigned “alive” status for middle cell in next generation. These patterns (or rules) have been used in conditional, lines 9 to 12. For patterns with result zero for middle cell in next generation, line 15 sets “dead” status.

Last steps of the algorithm are to advance the generation counter and draw the next generation on the canvas.

# Conclusion

The above work shows how the Conway’s Game of Life and Rule 30 has been implemented. We have explained step by step the various functions of the game and how it can be used. It is evident that the various functions of the game all come together to make the game a successful and intellectually crafted game.

Whilst carrying out the coursework there were a few problems with the coding. By communicating with each other on the regular basis and frequent meetings we were able to evaluate and solve the problems. We discussed the problems we faced creating the programme and brainstormed ideas on how to overcome them. Having done this we came to realise that two heads worked better than one and using each other’s strengths and weaknesses we both fixed the problems and generated a successful game.

Not only have we learnt from this experience as a group the importance of team work but we have also come realise how inspiring creating a game is. True planning and dedication is needed in order to produce a game of a good standard.

Another problem was working against time. We as a group have come to the conclusion that if we had more time we could have developed the game further. Introducing more complex patterns and rules which would help made the game more inspirational and challenging.

Personally I consider this project a success if the ideas described in the report can become a useful reference for a future work on the subject.

# References

[1] <http://pavolondzik.github.io/GoL/>

[2] <https://github.com/jpulgarin/canvaslife>

[3] <http://commons.wikimedia.org/wiki/File:Cartesian_coordinates_2D.svg>

[4] <http://nopr.niscair.res.in/bitstream/123456789/22859/1/IJFTR%2027%283%29%20242-247.pdf>

# Appendix

## List of files submitted

