London Metropolitan University

Faculty of Life Sciences and Computing

CU6051 Introduction to Artificial Intelligence

Emergence: Cellular Automata

Documentation

Name:

ID Number:

Date: Saturday, 04 January 2014

# Contents

[1 User Documentation 1](#_Toc376636183)

[1.1 Conway’s Game of Life 1](#_Toc376636184)

[1.1.1 Plain Mode 1](#_Toc376636185)

[1.1.2 Tab: Task 2](#_Toc376636186)

[1.2 Wolfram’s Rule 30 2](#_Toc376636187)

[2 Technical Documentation 3](#_Toc376636188)

[2.1 Code Organization 3](#_Toc376636189)

[2.2 Graphics 4](#_Toc376636190)

[2.2.1 Drawing the grid on canvas 4](#_Toc376636191)

[2.3 Conway’s Game of Life 4](#_Toc376636192)

[2.3.1 Data structures and main algorithm 4](#_Toc376636193)

[2.4 Wolfram’s Rule 30 5](#_Toc376636194)

[3 Conclusion 6](#_Toc376636195)

[4 References 7](#_Toc376636196)

[51. Appendix 8](#_Toc376636197)

[5.11.1. Code Listings 8](#_Toc376636198)

# Table of Figures

[Figure 1 - Code Organisation 3](#_Toc376636163)

[Figure 2 - Cells to Pixels Transformation 4](#_Toc376636164)

[Figure 3 – Pixels to Cells Transformation 4](#_Toc376636165)

[Figure 4 - Pseudo Algorithm: function Life.nextGeneration 5](#_Toc376636166)

[Figure 5 – create\_rfmon.sh 5](#_Toc376636167)

# 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 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. Life cells appear blue, dead cell has white inside and blue border. Orange border of dead or alive cell indicates that that 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.

#### Create Pattern 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 4 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

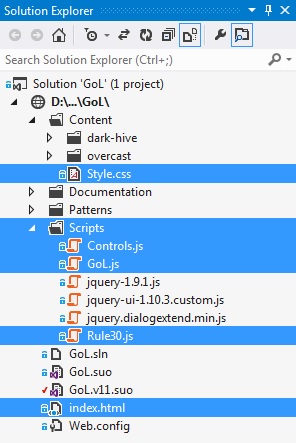
Technical documentation describes algorithms and main thoughts behind the produced solution. Documentation of the solution is introduced by chapter 2.1, Code Organization, which outlines how the code in these files fits together.

## 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. The pixels were recalculated to cell 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 position of starting point in pixels from cell coordinates. These equations were needed for recognizing the position of user’s click onto the grid.

Figure – Pixels to Cells Transformation



Left offset of canvas is distance of canvas element from left border of the browser window, and top offset is the distance of canvas from top border.

## 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 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.

Solution to program the Conway’s Game of Life 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.

When user selects a pattern, the “alive” value is kept in previous generation. Next generation is the same as previous, calculating next generation by Conway’s game of life, Adding life forms to next generation depending on game mode, drawing only cells from next generation which are different from cells in previous generation, Forgetting previous generation (previous generation becomes current)

Figure - Pseudo Algorithm: function Life.nextGeneration



The generations were kept in two two-dimensional arrays to keep two states. (object Life)

**Speed**: time delay is recalculated to "speed" with scale from 1 to 100. This speed is similar to frame rate.

## Wolfram’s Rule 30

vsvsa

**Error! Reference source not found.** deletes the interface.

Figure 5 – create\_rfmon.sh



fsdds

# Conclusion

This project was the first attempt to develop a system of this nature. I aimed to achieve developing a tool that will recognize Probe Request attacks in a real time. Furthermore, I have analyzed wireless traffic captured on a home wireless network and used supervised feed forward neural network to determine the results. I have also considered the ethical side of development of this tool.

It was identified from the beginning that producing a complete result would be impossible within the given timeframe. Software solution meets the functionality in accordance to the design strategies determined becoming a starting point for researching in this area. The working system has been created using many programming techniques and at least three programming languages. However, as it has been demonstrated in previous chapter, training neural network has been partly successful.

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/>

# Appendix

## Code Listings

APPENDIX A



