Individual Project: Cellular Automaton

Business Analysis

Michał Słupczyński

Document metric					
Project	Cellular Automaton			**Company**	WUT
Name:	Individual Project, Cellular Automaton, Business Analysis				
Topic:	Cellular Automaton				
Author:	Michał Słupczyński				
File:	indproj_business_analysis_slupczynskim.pdf				
Version no.	1.8	**Status:**	final	**Opening date:**	**2015-02-28**
Summary:	The main purpose of this document is to provide an initial business analysis of the Cellular Automaton application				
Authorized by				**Last modification date**	**2015-03-14**

History of changes

Version	Date	Who	Description
1.0	2015-02-28	Michał Słupczyński	Document creation, initial structure
1.1	2015-03-01	Michał Słupczyński	Glossary
1.2	2015-03-02	Michał Słupczyński	Glossary cont.
1.3	2015-03-04	Michał Słupczyński	Usability Requirements
1.4	2015-03-05	Michał Słupczyński	User stories, Summary
1.5	2015-03-06	Michał Słupczyński	User stories cont., Functional Requirements
1.6	2015-03-07	Michał Słupczyński	Minor changes to everything, UI Mockups
1.7	2015-03-10	Michał Słupczyński	UI Mockups cont., minor changes to layout
1.8	2015-03-14	Michał Słupczyński	Minor fixes

Table of Contents

Summary

The history and development of cellular automata is attributed to polish-born Stanisław Ulam *(1909 – 1984)* and John von Neumann *(1903 – 1957)* of Hungarian origin, dates back to the 1940s and was based on research on crystal growth modelling and self-replicating robots. Cellular automata are relevant to the study of biology, chemistry, physics and many branches of science.

The main goal of this application is to allow a legible and easy to use method of modelling and visualizing cellular automata.

Glossary

Cellular Automaton *(short CA)*

A cellular automaton consists of a rectangular **grid** of **cell**s, which can be in a specific **state**, a set of **rule**s that defines the creation of a new **generation** in the system and an **initial configuration**.

Grid

In general, a grid is a tiling of a geometric shape *(called tile or* ***cell****)* and can exist in any finite number of dimensions. For the sake of simplicity, in the context of this cellular automaton project, the grid is a two-dimensional *(fixed width x and height y <u>between 20 and 200</u>)* table of quadratic cells. Each cell in the grid can be addressed by means of an integer index [i, j], which represents the horizontal and vertical coordinate of the cell in the grid. In theory, the grid size of a **cellular automaton** is infinite, yet the practical implementation demands a finite size of the grid. If a cell outside of the specified grid is to be accessed, a **passive** *(0)* **state** of this cell is to be assumed.

Cell

An item in the **grid** of the cellular automaton is called a cell. A cell by definition can be in one of a finite set of **state**s in addition to having a neighborhood.

Cell state

A cell state describes the value of the cell at a given address in the **grid**. This value is from a finite fixed set of possibilities and is used in the application of a **rule** to the **generation** of cells in the automaton. Apart from this, the value of a cell defines the solid background color of this cell.

In this project, three cell states are taken into account:

Active (“alive”) cell state

The active cell state can be described as 1. This cell state should be colored with the **primary cell color**.

Passive (“dead”) cell state

The passive cell state can be described as 0. A white cell background should denote this cell state.

Fixed (“map”) cell state

The fixed cell state can be described as -1 and should be ignored in the process of counting the amount of neighbors of a cell which are alive. The state of a fixed cell cannot change by means of a **rule**. A fixed cell can be part of the **initial configuration** of the automaton. A black cell background should denote this cell state.

Primary cell color

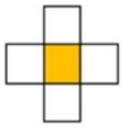
This color can be either a hard-coded value or a user preference. A “strong” color is suggested for legibility purposes. This could mean for example red, green, blue, orange or purple.

Neighborhood

A neighborhood is the set of **cell**s *(in the same grid)* situated directly *(or in the case of a 24 point neighborhood also indirectly)* adjacent *(or neighboring)* to the cell in question *(or “current cell” – the one which is marked yellow in the images on the right)*.

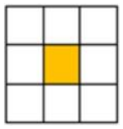
In this project, the following three types of neighborhoods are taken into account:

4 point neighborhood



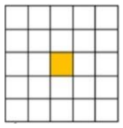
In the four-point neighborhood, only the four cells orthogonally adjacent are considered, that is, the ones above, below, to the right and to the left.

8 point neighborhood



In the eight-point neighborhood, the eight horizontally, vertically and diagonally, adjacent cells are considered.

24 point neighborhood



In the twenty-four-point neighborhood, the two closest cells in every direction *(vertical, horizontal and diagonal)* are considered.

Initial Configuration

The initial configuration of the cellular automaton describes the state of all **cell**s in the **grid** before the first **rule** gets applied. This is usually denoted by the **time step** variable t being equal to 0.

Step t

This helper variable denotes the amount of times the specified rule or rule set has been applied to the **grid** of the cellular automaton since its **initial configuration**. On every application of a **rule**, this variable should be incremented by one. When the rule set is changed, this variable should be reset to 0.

Rule

At every time step, a rule calculates the **state** of the current **cell** *(i.e. every cell in the **grid**)* based on the set of states of the neighboring cells at the previous **time step**. This means, e.g., that on the first application of a rule, the **initial configuration** is taken into account for the calculation of the first **generation**. A rule can, depending on the user preference *(or input)*, consist of a fixed finite set of sub-rules, where a **sub-rule** defines the number of active neighbors surrounding the current cell ***(cardinality)*** in total, or in a specific row, column or diagonal. The application of a rule to the grid of cells creates a new **generation**. A rule is applied to the entire grid at once.

Generation

A generation of **cell**s denotes the set of **state**s of the cells of the cellular automaton at a given **time step**. The concept of a generation in the context of this project can be easily understood by means of a metaphor of the *frame count* of an *animation*, where the frame of the animation shows the content of the image at the given moment of time. This means that when a new **rule** or rule-set gets applied to the current generation, a new generation gets created and the **time step** gets incremented by one.

Contradicting rule

Two **rule**s are said to be contradicting when for the output value is different for the same set of **sub-rule**s. In other words, when the **CA** cannot easily decide which *(out of two or more)* rules to apply.

Requirement specification

Usability requirements

The planned application should operate in a quick manner, that is, preferably all time-consuming tasks should be carried out in an asynchronous *(i.e. for every few points to be calculated, a thread should be spawned to calculate their neighborhood)* manner and progress indicators can be used where applicable. Unnecessary features should be avoided, as it is better to have a well functioning smaller feature-set than to have a lot of features and a lot of bugs. In addition to this, the program should be easy to use and provide the user with legible information.

Functional requirements

Priority	Requirement	Comment
1	Display current state of CA	Color cells in grid based on their state
1	Change the initial configuration	Change grid size *(resets automaton)* Set Cell states by click on grid *(toggle 0 <-> 1 <-> -1)*
1	See the state of CA after n steps	After n steps *(input n)*
1	See the state of CA after the next step	Click a button to step
1	Apply rule to grid	See *Rule*
1	Change neighborhood type	See *Neighborhood*
1	Define rule by neighbor count	Cardinality can be *[>, <, =]* specified integer value
1	Define rule by specific neighbor location	Can be row, column and/or diagonal *Or in the case of 4pt: above, below, left, right*
1	Define rule input/output	Input = value before rule, output = value after rule
1	Save and load state of CA	Use file on hard drive, load as Initial Configuration
1	Save and load rule set	Use file on hard drive, ask for conflicts
2	Resolve conflicting rules on adding	- Choose one rule, discard the other one or - Merge rules *(AND, OR, XOR)*
2	Zoom and drag grid	The shown grid can be a subset of the total grid
3	Animate the state of CA	Define Timer for automatic step every k ms
3	Allow loading of preset Rules and Initial Configurations	Examples can include: Conway's Game of Life and other well-known cellular automata
3	Change primary color	For list of colors proposed for legibility *see Primary cell color*

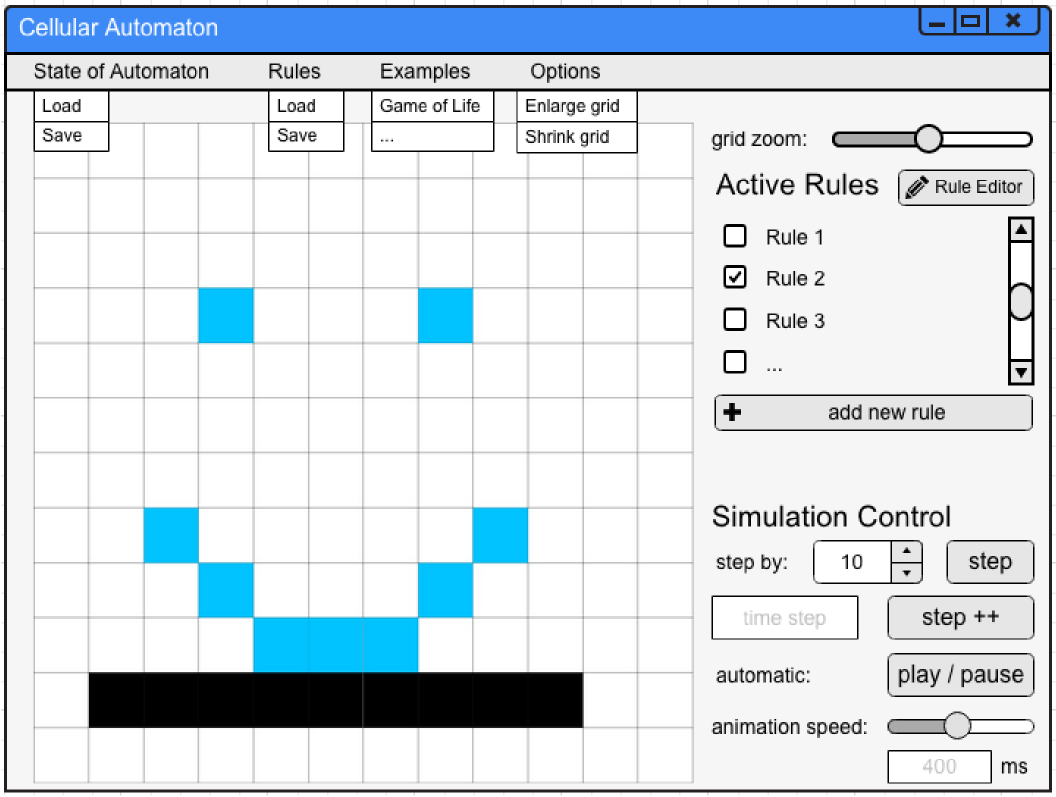
Explanation: 1 – must be, obligatory; 2 – should be, good to have; 3 – could be, optional

Usage description

As a user, I want to…

- See the current **state** of the automaton
    - After n **time step**s *(and to change n)* because I might be only interested in the state after a specified amount of generations have been created
    - Step by step so that I can easily analyze the simulation
    - Be able to play/pause/resume the calculation with an appropriate button or hotkey *(e.g. spacebar)* so that I have full control over the simulation
    - Change the speed of the simulation *(preferably in ms)* to be able to animate the state of the cellular automaton
    - See the current **step t**
- Save the current **state** of the automaton
    - Load a saved state as the **initial configuration** of the CA
- See and edit the **initial configuration** of the system
    - Change the size *(squared)* of the **grid**
    - Toggle the **state** of cells in the grid *(preferably on click)* between **active, passive and fixed**
    - Add and remove **fixed cell**s to the initial configuration
    - Zoom and drag the current position of the **grid**
- **CRUD** *(create, read, update, delete)* elements in the **rule** set during runtime
    - Change the **neighborhood** *(4pt, 8pt, 24pt)* type
    - Edit the rule with easy input:
        - Change cardinality *(number of alive cells)* of a rule/sub-rule to be bigger than *(>)*, smaller than *(<)* or equal to *(=)* a specified integer value
        - Specify whether the cardinality applies to the whole neighborhood, a specific row, a specific column or one of the diagonals in the neighborhood
        - Specify the input *(value before application of rule)* and output *(value after application of rule)* state of the middle cell *("current cell", see Neighborhood)*
    - Stop the current calculation on every rule change and use the current state of the CA as the **initial configuration** for the next calculation
    - Be asked to resolve conflicts *(e.g. by means of a popup)* when two contradicting rules are input
        - See the sub-rules of the conflicting rules
        - Be able to resolve the conflict by either merging (AND, OR, XOR) or picking one of the conflicting rules
    - Save and load the **rule** set
- Load an exemplary **rule** set and/or an according set of **initial configurations** to be able to quickly
- Change the Primary cell color

Proposed Graphical User Interface (GUI) – mockups

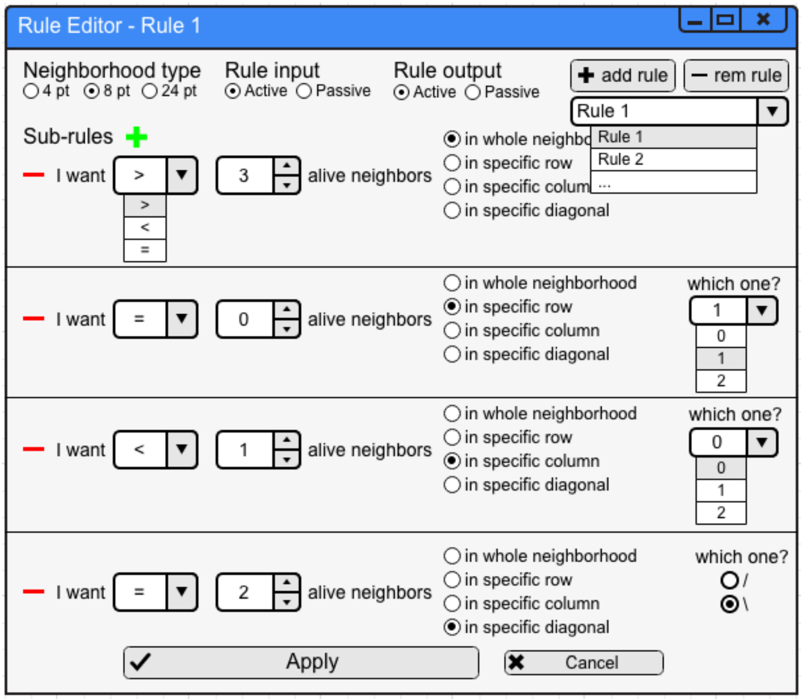


Active Rules: [ ] Rule 1 [✓ Rule 2] [ ] Rule 3 [ ] ...

step by: 10

animation speed: 400 ms

Figure 1 Main Application Window



Neighborhood type: 4 pt [✓ 8 pt] 24 pt

Rule input: [✓ Active] Passive

Rule output: [✓ Active] Passive

Rule 1

Sub-rules:

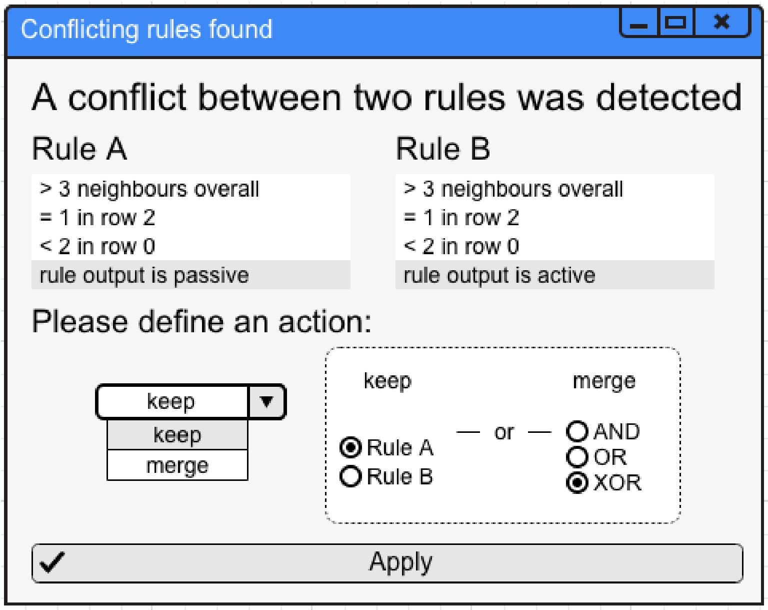
I want > 3 alive neighbors: [✓ in whole neighborhood] in specific row / in specific column / in specific diagonal

I want = 0 alive neighbors: in whole neighborhood / [✓ in specific row] / in specific column / in specific diagonal — which one? 1

I want < 1 alive neighbors: in whole neighborhood / in specific row / [✓ in specific column] / in specific diagonal — which one? 0

I want = 2 alive neighbors: in whole neighborhood / in specific row / in specific column / [✓ in specific diagonal] — which one? / [✓ \]

Figure 2 Rule Editor



Conflicting rules found

A conflict between two rules was detected

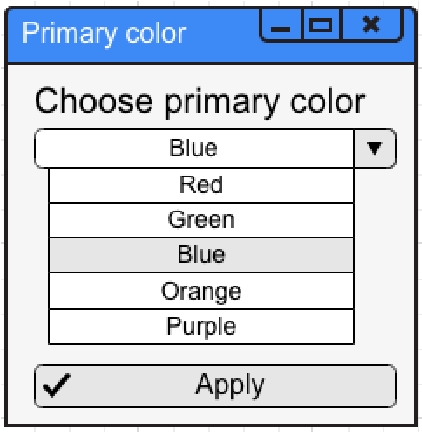
Rule A	Rule B
> 3 neighbours overall	> 3 neighbours overall
= 1 in row 2	= 1 in row 2
< 2 in row 0	< 2 in row 0
rule output is passive	rule output is active

Please define an action:

keep ▼ (keep, merge)

keep: [● Rule A] ○ Rule B — or — merge: ○ AND ○ OR [● XOR]

Figure 3 Conflict resolving: The content of the white box should dynamically change based on the input of the drop down picker of the action



Primary color

Choose primary color

Blue ▼ (Red, Green, [✓ Blue], Orange, Purple)

Figure 4 Primary Color picker

Ending part

Unit testing is highly advised.

The layout and design of the application is dependent on the implementation and the chosen programming language and libraries, the GUI mockups are only a general guideline.

Any implementation questions can be sent to slupczynskim@student.mini.pw.edu.pl