Technical Analysis

Individual Project: Cellular automaton

Michał Szklarski

Computer Science

4 April 2016

Document metric					
Project:	Cellular Automaton			**Company:**	WUT
Name:	Technical Analysis – Individual Project				
Topics:	Technical specification of Cellular Automaton				
Author:	Michał Szklarski				
File:	Technical Analysis.docx				
Version no:	07	**Status:**	FINAL	**Opening date:**	2016-04-04
Summary:	To extend and concretize requirements described in Business Analysis for the Cellular Automaton project.				
Authorized by:				**Last modification date:**	2016-04-07

History of changes			
Version	**Date**	**Who**	**Description**
01	2016-04-04	Michał Szklarski	Initial version, definition
02	2016-04-04	Michał Szklarski	Added document metric and history of changes parts
03	2016-04-05	Michał Szklarski	Added summary – overview part.
04	2016-04-05	Michał Szklarski	Extended summary + general specification + conclusion
05	2016-04-06	Michał Szklarski	Technologies + methodology + dev. process flow
06	2016-04-07	Michał Szklarski	Similar solutions analysis
07	2016-04-07	Michał Szklarski	Algorithm and other elements description, diagrams, GUI

Michał Szklarski

Computer Science

4 April 2016

Technical Analysis

Individual Project: Cellular automaton

Technical specification

Table of Contents

Summary – overview

Aim of this document is to present requirements from the business analysis of a Cellular Automaton application as a technical specification concretization. All implementation details and decisions, such as chosen language, technologies, frameworks, libraries and algorithms will be described in this document. Document is divided into several parts, starting from general specification description, planned technologies, methodology description, development process flow, similar solutions analysis, algorithm description, other program elements description, diagrams (activity, class), GUI description and finally: conclusion and last summary about this document.

This document opens path to the implementation process by giving exact directions and decisions, which will be followed by testing phase.

General specification

A Cellular Automation software, as described in previous document from the business point of view is a software emulating cellular life.

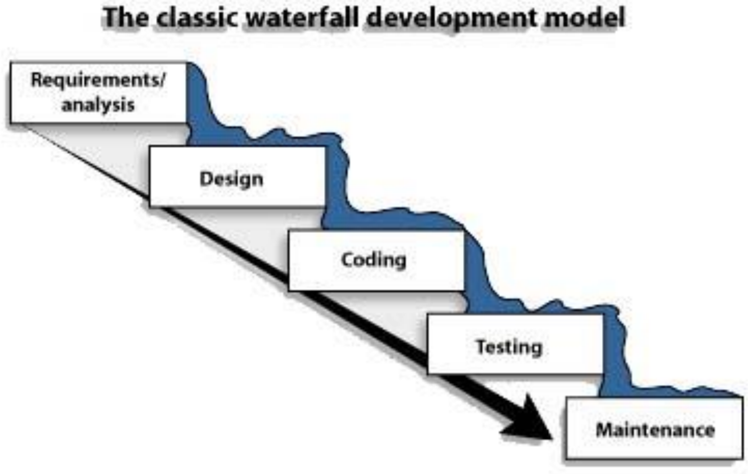
Generally speaking, program will be composed of three main parts: main grid with cells, menus, inputs and buttons (both on main window) for operations and custom rules editor as a separate window. Algorithm handling grid and rules check & enforce policies will be present in the background of whole solution.

Technologies

As a best suited option for development of such project, **C#** programming language was chosen. Best choice for presentation layer was estimated as a **Windows Presentation Foundation** (WPF) solution. WPF provides a rich and stable API for simple and complex solution, as the one described in this document. It is also well documented, which is an important factor for development process. Main IDE for the project was appointed to be **Visual Studio 2015**, as it is most powerful C# development environment, with integrated **Visual Studio Unit Testing Framework**, that also will be uses here, for testing phase.

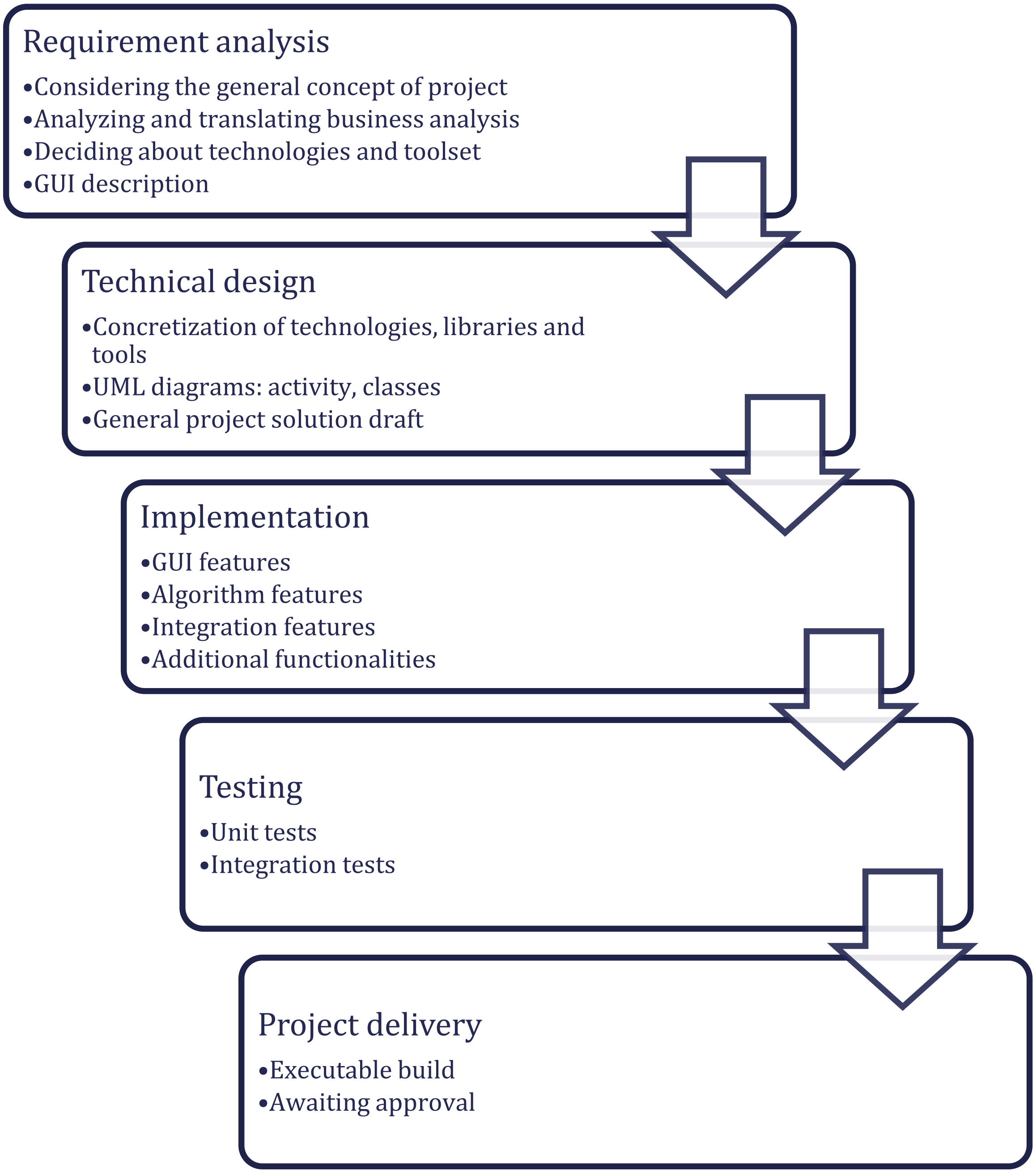
Methodology (development model)

Business specification is finished now, chosen methodology for this project is **Waterfall**. Due to project complexity, clarity of the requirements, individuality, and course requirements described in initial presentation it is exactly the perfect solution for this assignment. As presented here:



One modification to that diagram is such, that we won't be handling maintenance phase. Everything up to Testing remains according to the official Waterfall methodology rules. Current phase is determined as Design on the diagram. Each stage has clearly defined goal therefore it is possible to efficiently control project flow, and detect any impediment.

Development process flow



Requirement analysis

- Considering the general concept of project
- Analyzing and translating business analysis
- Deciding about technologies and toolset
- GUI description

Technical design

- Concretization of technologies, libraries and tools
- UML diagrams: activity, classes
- General project solution draft

Implementation

- GUI features
- Algorithm features
- Integration features
- Additional functionalities

Testing

- Unit tests
- Integration tests

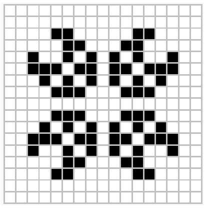
Project delivery

- Executable build
- Awaiting approval

Similar solutions analysis

As supposed, our project isn't the pioneer one. There exist similar solutions, like, for example: Conway's Life, Wireworld, Langton's Ant, Brian's Brain. Let's discuss and analyze most popular two of them - Conway's Game of Life and Langton's Ant cellular automaton.

Conway's Game of Life

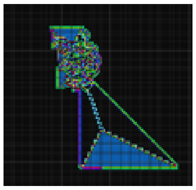


Conway's Game of Life is a cellular automaton invented by the British mathematician John Horton Conway in 1970. It implements infinite, two-dimensional and orthogonal grid of cells, each on it in two possible states: dead or alive. Neighborhood of interaction for one cell is set as an eight direct adjacent cells. For each iteration of lifecycle (a tick) following rules are applied:

- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with fewer than two live neighbors dies, as if caused by under-population.
- Any live cell with more than three live neighbors dies, as if by over-population.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

These rules are applied repeatedly in order to create further generations. Initial one is created by applying all of the rules on every cell field.

Langton's Ant



Langton's Ant is a Cellular Automaton with a very simple set of rules but complex resulting behavior. It was invented by Chris Langton in 1986. It also implements two-dimensional grid of cells, each on it in two possible states: black or white. One cell is designated to be an "ant", which can travel in any of the four basic directions (N, S, W, E) according to the rules:

- At a white square, turn 90° right, flip the color of the square, move forward one unit
- At a black square, turn 90° left, flip the color of the square, move forward one unit

These simple rules are proven to create complex behavior, described either as a simple, chaotic, or emerging order.

With similar solutions analyze completed, we can now focus on delivering our own one, knowing about advantages and disadvantages of other implementations.

Algorithm description

Considering existing solutions and general requirements specification for this project (from the business point of view, first part). In order to operate the main program algorithm, we need, at first, to specify overview on input and output:

Input parameters:

- Initial set of rules (RuleSet)
- Initial state of grid (Grid)
- Count of cycles to run (Integer)

Output:

- Grid after performed operations
- (optional) file with grid state or custom rules

Cell in algorithm is defined in three states – dead, alive, and empty, colored properly: red, dark green, white.

Describing main module and window of an application: initial size of the grid is estimated as a screen resolution plus one (possibly two) additional row/column at the edges of projected grid. Algorithm should iterate according to set of rules through whole space of grid, starting from upper left side. After one iteration on whole grid, it is updated, according to the rules (starting from the first one in RuleSet) concerning neighborhood from last iteration, new states of each cell are copied to temporary state (threaded operation for each). Users are able to play, pause, stop simulation using buttons on main menu, and define number of steps (cycles) to perform on the simulation. After desired number of steps, simulation is stopped.

Other program elements and structure description

Main window module

Whole grid will be possibly implemented as a WPF Canvas, due to its graphical features. Minimal size of one cell is set to be one pixel. Menu on the right side will contain buttons of flow control of simulation, input for cycles count and zoom in/out controls. Menu will contain options regarding rules management, file saving and opening, application closing. Application should prompt if exit was planned.

User might save grid state after stop of simulation in file, and open grid state at the beginning (binary file output/input).

Rule set window module

On this module of application, user is able to select one of available defined rule sets from drop-down list. Also, there is a possibility to create own rules, using graphical editor or text input, showing grid with exemplary cell with its neighborhood (24 surrounding cells, from specification). User can change the state on a cell by clicking on it (from 3 possibilities, disclosed earlier). User has to check one of the following conditions in order to run the rule:

- Check number and positions of neighboring cells
- Check for a specific positions of neighboring cells
- Check for a specific state of neighboring cells

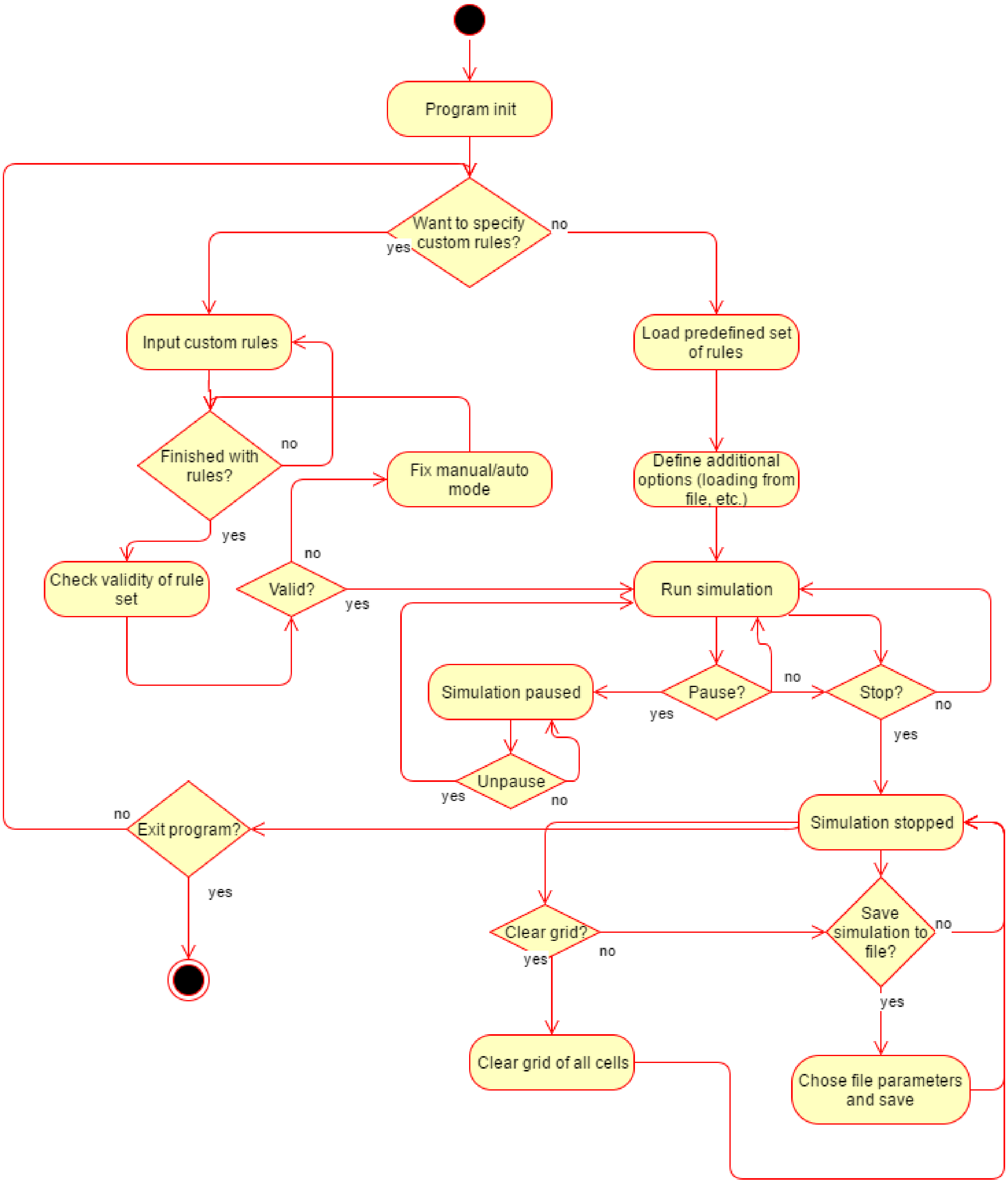
User is also able to disclose them using text input, which will be parsed into a program rule. Then, validity checks for rules will be applied.

For validity, each rule, starting from the first one on a rule set is checked according to the rest. If it's not parseable or contradicting, then the error is raised with question how to solve it. Module is able to manually fix broken rules, or do it by approximation. If approximation also fails, user is prompted to fix it ultimately manually, according to the rest of set.

Program structure

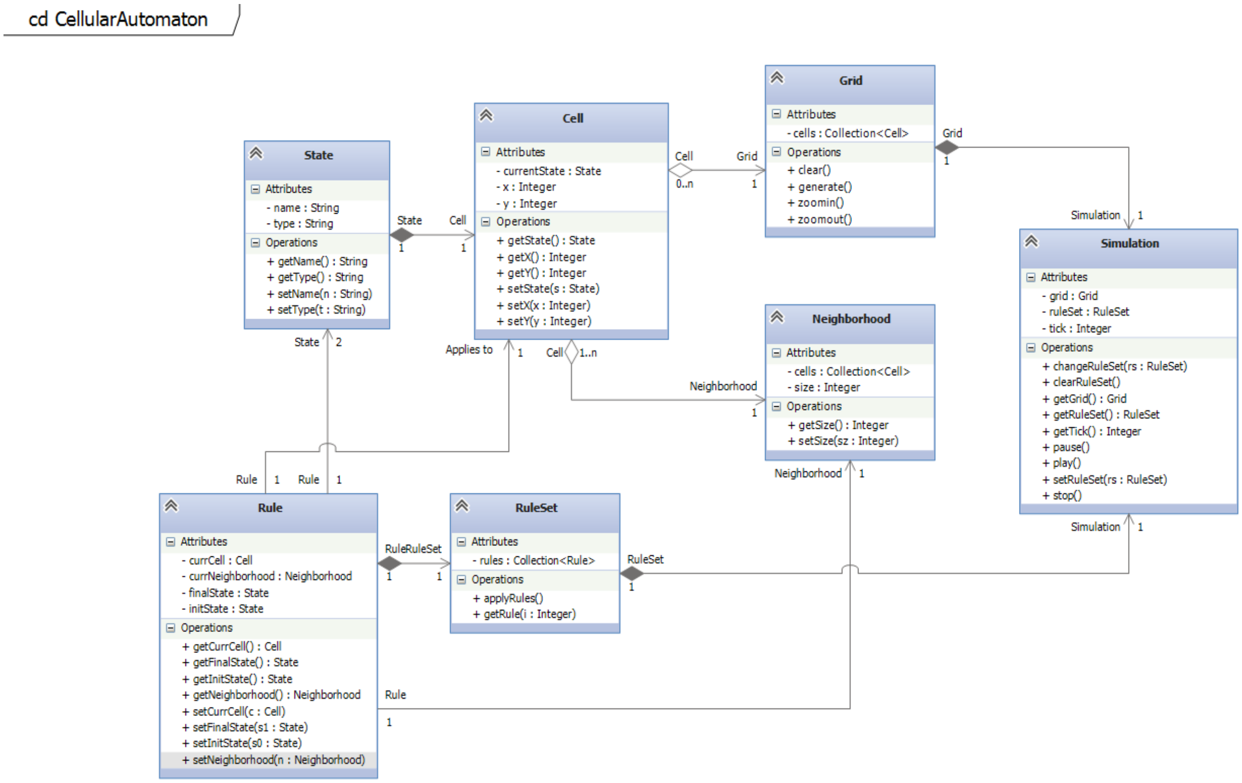
Project will be structured into separate modules describing windows, as in: Main, Rule Editor. Code will consist of classes with public or private modifiers and XAML files describing User Interface and bindings, as in C# with Windows Presentation Foundation Technology. Due to this architecture, implementation process should be seamless and intuitive.

Activity diagram



Class diagram

It describes general idea behind whole project and classes allocation, as it is prone to further changes and revisions:



GUI prototype

For GUI prototype description, please see Business Analysis document for this project, part *Graphical User Interface description*, where it is presented with all relevant details.

Conclusion

As described in this document, technical analysis is done. Following remaining part is to implement desired solution. Because of complexity of the project, topics presented in this document are subject of further changes. All points considered here should be thoroughly translatable into chosen programming language. From there, development processes can move on into testing phase, which is very important in every such project.