|  |
| --- |
|  |
| 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:** | 2016-04-14 |

|  |  |  |  |
| --- | --- | --- | --- |
| 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 | 2016-04-14 | Michał Szklarski | Algorithm and other elements description, diagrams, GUI, fixes |

# Technical Analysis

## Individual Project: Cellular automaton

## Technical specification

Table of Contents

[Technical Analysis 1](#_Toc448405530)

[Individual Project: Cellular automaton 1](#_Toc448405531)

[Technical specification 1](#_Toc448405532)

[Summary – overview 3](#_Toc448405533)

[General specification 3](#_Toc448405534)

[Technologies 3](#_Toc448405535)

[Methodology (development model) 4](#_Toc448405536)

[Development process flow 5](#_Toc448405537)

[Similar solutions analysis 6](#_Toc448405538)

[Risk analysis 7](#_Toc448405539)

[Algorithm description 8](#_Toc448405540)

[Other program elements and structure description 14](#_Toc448405541)

[Activity diagram 16](#_Toc448405542)

[Use Case diagram 17](#_Toc448405543)

[Class diagram 18](#_Toc448405544)

[Data Structure description 18](#_Toc448405545)

[GUI prototype 20](#_Toc448405546)

[Conclusion 20](#_Toc448405547)

### 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, risk analysis, algorithm description, other program elements description, diagrams (use case, activity, class), data structure description, 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.

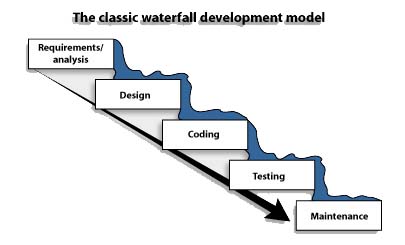
Generally speaking, program will be composed of three main parts: main grid with cells, menus, inputs and buttons (both on main window) for operations and custom rules editor as a separate window. Algorithm handling grid and rules check & enforce policies will be present in the background of whole solution. This document, specification may be a subject of changes due to optimization or correction features.

### 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:



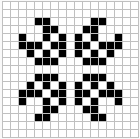
One modification to that diagram is such, that we won’t be handling maintenance phase. Everything up to Testing remains according to the official Waterfall methodology rules. Current phase is determined as Design on the diagram. Each stage has clearly defined goal therefore it is possible to efficiently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

### Development process flow

### 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.

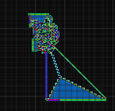
##### Conway’s Game of Life

 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.

### Risk analysis

Risk analysis is an important part of this documentation. It gives an insight on how project might be delayed, abrupt or failed. It is an essential step for proper planning of development and testing processes. One can prepare various scenarios, just in case of unknown perturbations. For this project, we can model risks such as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Who? | Risk | Probability | Impact | How to prevent? |
| Author | Missing deadline | High | High | Better time & project management |
| Author | Testing failure (not detecting various crucial errors) | High | Medium | Being focused when programming, creating proper methods, checking code beforehand |
| Author | Testing failure (application crashing) | Medium | High | Better debugging, checking code beforehand. |
| Author | Project delivery failure | Medium | High | Keep deadline dates and specification on mind all time |
| Author | Author sickness | Low | High | Keeping good health status |
| Author | Misunderstanding of concept | Medium | Medium | Attending additional meetings with supervisor |
| Author | Failure to deploy on faculty computer | Low | High | Test beforehand on faculty PC |
| Author | Incompatibility with project technical documentation | Medium | Medium | Read project technical documentation carefully |

Following risks are described with fields of owner (who?), probability factor, impact factor, and prevention method. From there, one can estimate risk factor of every risk on this list by assigning numerical, discrete values to probabilities and impacts (e.g. low = 1, high = 3). Then, it’s possible to obtain that factor value by multiplication those numerical values. Obtaining that risk factor shows us what is most important risk to be aware of. For case of this project, “Missing Deadline” is the most important risk, with factor estimated to 9 points.

### 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 column (threaded operation for each column). 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.

#### Algorithm flowcharts

As there is a need to present direct algorithmic solution for mentioned parts of cellular automaton, following are presented strictly technical and ready to implement algorithm flowchart for following parts of application:

1. General iterating through main Grid:



Main flow of cellular automaton project – here presented are iterations through main Grid with a help of snapshot one. Operations are threaded – one Grid column (including Neighborhoods) per thread. Applying rules and oscillating status detection are presented in next diagrams.

1. Rule applying:

Here one specific iteration on Cell with Neighborhood is presented. For such process all Rules from RuleSet are applied regarding nearest Cells. Input is initState from rule, output: finalState of specific Cell (dead, alive, empty).

1. Rules validity check (automated):

Rules are checked according to their initState (as in note in diagram). If set is empty, we also cannot accept it. Then, if rules are contradicting with each other or are duplicates, we have a reason to reject them. Rules are contradicting with each other when they give other finalStates for the same initState.

1. Oscillation Grid status check:



Oscillation state, as described in state diagram, is a state that requires us to stop a computation. It can be detected using history Grid snapshots, comparing them to current one. It is not perfect solution for optimization, but deals with such problem in a non-complex way. If we detect such state above, we return false, which should stop current simulation.

#### Automaton States

Our computation, and through that cellular automaton might enter following states:

* Setup
* Applying
* Working (computing)
* Paused
* Oscillating
* Stopped

For detailed description, it is feasible to present automaton state diagram:

Program enters initial state, which is Setup. User supplies input options there such as number of iterations (generations) n, RuleSet. Then, if user clicks apply button, custom (or predefined) set of rules is applied and ready for simulation run. From there, user can go back and correct their RuleSet. Then, if start is triggered, simulation enters computation state. From there it can go either to Oscillating, Paused or Stopped state, as indicated nearby transitions. From Stopped state user cannot go back to simulation run. Then it can be reset to Setup state.

### 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

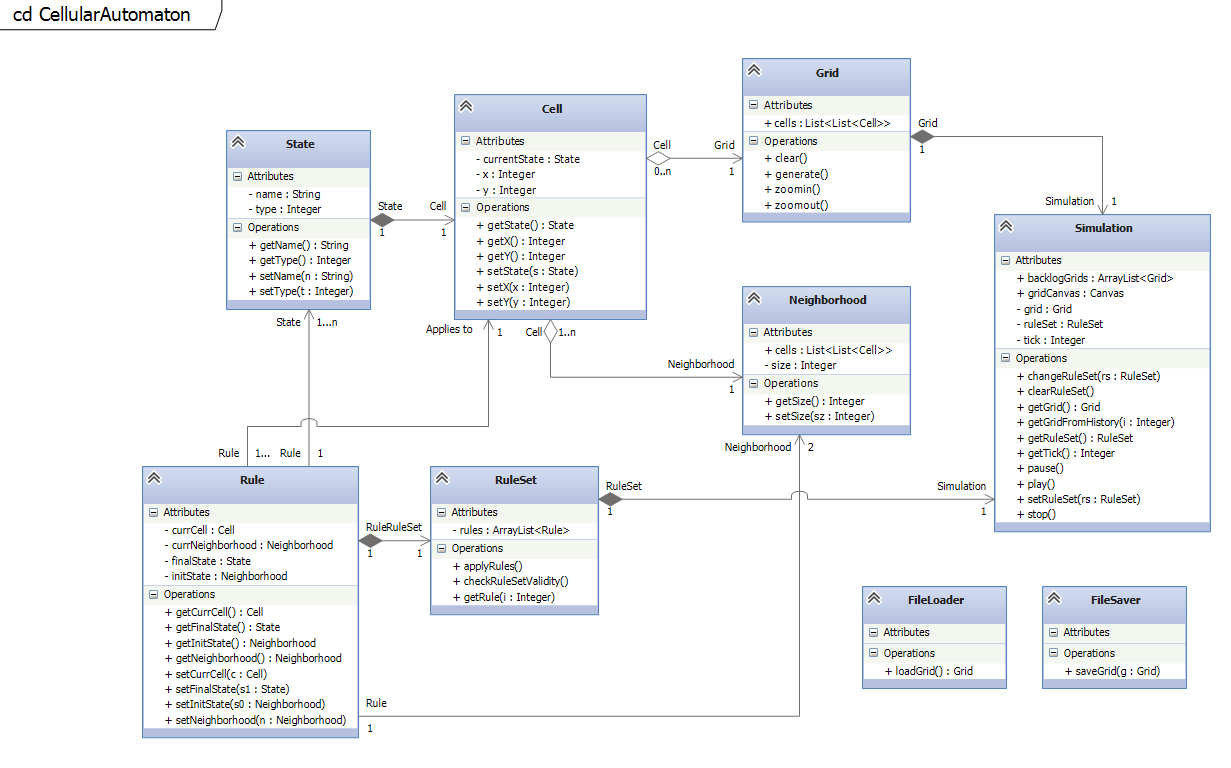
Project will be structured into separate modules describing windows, as in: Main, Rule Editor (with possibility of some Help window). Code will consist of classes with public or private modifiers and XAML files describing User Interface and bindings, as in C# with Windows Presentation Foundation Technology.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

### Activity diagram

### Use Case diagram

This diagram corresponds business side of documentation, now described as correct and detailed features from Cellular Automaton user point of view.

### Class diagram

 It describes general idea behind whole project and classes allocation, as it is prone to further changes and revisions during implementation (not crucial).

### Data Structure description

For the project following data structures (basing on class diagram above) shall be used as optimal solutions. Classes won’t be described here, only their container structures as fields. They were selected from available collections in C# language:

* **For Grid:** List of Lists of type Cell (two dimensional iterative data structure holding all cells for current simulation). With maximal size of screen resolution.
  + **List<List<Cell>>**
  + **Max size x: screen res. x + 2**
  + **Max size y: screen res. y + 2**
  + **Here, grid canvas (Canvas class) will be applied as main visual Grid representation for Windows Presentation Foundation. It will contain all Cells with their states, prepared to be displayed on screen. It inherits size from main Grid container.**
* **For Grid snapshots:** List of Lists of type Cell (two dimensional iterative data structure holding all cells for current simulation). With maximal size of screen resolution.
  + **List<List<Cell>>**
  + **Max size x: screen res. x + 2**
  + **Max size y: screen res. y + 2**
* **For Grid snapshots backlog:** ArrayList ofList of Lists of type Cell (three dimensional iterative data structure holding all relevant historical representation of grid of cells for current simulation). With maximal size of screen resolution.
  + **ArrayList<List<List<Cell>>>**
  + **Max size x: 4 (2 for optimized solution)**
  + **Max size y: screen res. x + 2**
  + **Max size z: screen res. y + 2**
* **For initState of Rules:** Neighborhood class containing List of Lists of type Cell (two dimensional iterative data structure holding all cells for current neighborhood). Maximal and only size of neighborhood, estimated to 24 cells.
  + **List<List<Cell>>**
  + **Max (static) size x: 5**
  + **Max (static) size y: 5**
* **For RuleSet:** ArrayList of type Rule (one dimensional iterative data structure of objects whose size is dynamically increased as required). Maximal size not defined.
  + **ArrayList<Rule>**
  + **Max size x: not defined**
* **For Neighborhood:** List of Lists of type Cell (two dimensional iterative data structure holding all cells for current neighborhood). Maximal and only size of neighborhood, estimated to 24 cells.
  + **List<List<Cell>>**
  + **Max (static) size x: 5**
  + **Max (static) size y: 5**

Any temporary data structures might be created as if needed during implementation phase, for example as optimization solutions or snapshot containers for automaton.

### 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.