CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Design and Implementation of Cellular Automaton Simulating Domain Growth
Student:	Martin Mužák
Supervisor:	RNDr. Jiří Kroc, Ph.D.
Study Programme:	Informatics
Study Branch:	Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

The main goal of the work is to implement a cellular automaton simulating growth of domains. Theoretical aspects of the algorithm are explained in the publication [1]. The application will consist from a back-end that is already written in C and from a front-end that is to be designed and implemented.
1. Familiarize yourself with the design concept and implementation of the cellular automaton simulating domain growth.
2. Rewrite the existing back-end from C to C++.
3. Design and implement a front-end in the C++ language with use of the Qt library.
4. Properly document the whole application and test it.
The final program will be available on portals Researchgate and Sourceforge in the form of freeware as an example of self-organization.

References

[1] J. Kroc: Diffusion Controlled Cellular Automaton Performing Mesh Partitioning, LECTURE NOTES IN COMPUTER SCIENCE, Vol. 3305, 131-140, 2004.
[2] A. Hoekstra, J. Kroc, P. Sloot: Introduction to Modeling of Complex Systems Using Cellular Automata, Springer, In book: Simulating Complex Systems by Cellular Automata (Understanding Complex Systems), Berlin, Heidelberg, 1-16, 2010.
[3] J. Kroc: Diffusion Controlled Cellular Automaton Viewed as Example of Self-Organization within Biology, https://www.researchgate.net/publication/280933644_Diffusion_Controlled_Cellular_Automaton_Viewed_as_Example_of_Self-Organization_within_Biology.

L.S.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 1, 2015

Czech Technical University in Prague

Faculty of Information Technology

Department of Software Engineering

Bachelor's thesis

Design and Implementation of Cellular Automaton Simulating Domain Growth

Martin Mužák

Supervisor: RNDr. Jiří Kroc, Ph.D.

9th May 2016

Acknowledgements

I would like to thank my supervisor RNDr. Jiří Kroc, Ph.D. for introducing me to this interesting topic, as well as for the support during the development process and writing the thesis text. Also, I would like to thank my family and friends for all the support during my studies.

Declaration

In Prague on 9th May 2016 ....

Citation of this thesis

Mužák, Martin. *Design and Implementation of Cellular Automaton Simulating Domain Growth.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstract

This bachelor thesis is devoted to design and implementation of the graphical user interface of cellular automaton simulating domain growth. User interface is implemented in C++ with usage of framework Qt. Important elements are intuitive controls of the application and clean code, which allows another program modifications. Application allows user to set initial location of the domains and their growth factor. During simulation user can perform selected actions.

Keywords desktop application, complex system, celullar automata, domain growth, self-organization, emergence, visualisation, C++, Qt

Abstrakt

Tato bakalářská práce se věnuje návrhu a implementaci grafického uživatelského rozhraní buněčného automatu simulující růst domén. Uživatelské rozhraní je implementováno v jazyce C++ s použitím frameworku Qt. Hlavním požadavkem je intuitivní ovládání a čistý kód, umožňující dalším úpravy programu. Aplikace nabízí možnost počátečního rozložení domén a jejich faktoru růstu. Během simulace lze provádět vybrané akce.

Klíčová slova desktopová aplikace, komplexní systémy, buněčné automaty, růst domén, samoorganizace, emergence, vizualizace, C++, Qt

Contents

List of Figures

Introduction

Very often scientists struggle with the problem of simulating various physical phenomena, for example spreading of the forest fires or organization of the ants in an ant colony. In 1940s, John von Neumann and Stanislav Ulam discovered the concept of cellular automata[1], which von Neumann defined as a method of complex systems simulation. Cellular automaton theory is described as a universe consisting of an homogeneous array of "cells"[2, 3]. Each cell has a finite number of states and according to its current state and rules, which are defined for the whole system, the cell reacts to specific situations. As computations of cellular automata run simultaneously for all cells at the same time, they are seen as a paradigm of distributed computations. Based on von Neumann's thesis, many scientists had contributed to this field.

Nowadays, we can find a number of programs where evolution of CAs can be simulated. However, the spectrum of possible physical phenomena described by cellular automata[3, 4] is so wide that creating a program which would cover the whole spectrum is impossible. Therefore, this thesis is dedicated only to a part of this spectrum, the self-organization, which is one of the key processes observed within complex systems.

The main goal of this thesis is to enable researchers to understand basics of complex system modelling on an example dealing with domain growth[5] and to conduct their own research motivated by the program. The application offers a great degree of freedom in configuration of simulation, with a possibility to save simulated models as an image file in several formats. Another goal of this thesis is to improve the accessibility of programming of the complex systems. The application offers to switch on the hints and prompter for every user action, so the less skilled users can use the program without being afraid of the lack of knowledge. The application is open-source, which means that anybody can modify the source code and develop his/her ideas on how to improve the program.

The first chapter is devoted to the introduction into the complex systems, especially in one way of their possible simulation - cellular automata. In the next chapter, the current state of the art is analyzed. The third chapter contains definition of the functional requirements, non-functional requirements and requirements on the graphical user interface. The next chapter continues with the analysis and describes typical users and use cases. The fifth chapter is about application design, technologies used for implementation and class design. Classes implemented in the resulting application are also described in this chapter. The next chapter contains a description of the realization of the program and the source code snippets, which implements important and complex parts of the application. In the chapter dealing with application testing, used techniques and actions that were taken after the test results evaluation are described. The last chapter is a summary of the whole thesis and it concludes whether the requirements were fulfilled.

CHAPTER 1

Introduction to Complex System and Cellular Automata

The basic knowledge of the complex systems and the ways how to simulate them is crucial for understanding of the CSs applications. This chapter is meant to provide such knowledge. It contains a brief introduction to this issue, as well as an introduction to the massive parallel computations, self-organization and emergence.

1.1 Complex Systems

1.1.1 Definition

There is a number of widely used definitions describing complex systems. A lot of researchers have their own explanation[1, 6, 7], but generally, we can describe them as subjects consisting of well-defined components. Together, they are forming a functioning whole, where behavior is dynamical and with responses to the environment.

"In complex systems we observe group or macroscopic behavior emerging from individual actions and interactions. One of the first biological observations of complex systems, if not the first one, is linked to ant-colony behavior. Each ant is simply following its internally encoded reactions to external stimuli. It is a well-known fact that ant species may have a set of 20 up to 40 reactions on any given external stimulus. Despite the lack of controlling entities, an ant-colony builds its ant-hill, feeds it, protects it, attacks other colonies, follows a foraging strategy etc. This emergent behavior observed in ant-colonies is prototypical for many other complex systems."[4]

"The components of a complex system are most commonly modeled as *agents* i.e. individual systems that act upon their environment in response to the events they experience. Examples of agents are people, firms, animals and molecules. The number of agents in the system is in general not fixed as agents can multiply or *die*. Usually, agents will react to a specific condition perceived in the environment by producing an appropriate action. The casual relation or rule connecting condition and action, while initially fixed for a given type of agent, can in some cases change, by learning or evolutionary variation.”[8]

Examples of a complex system can be found in the whole spectrum of scientific disciplines, from physics, medicine or chemistry to parts of human society like stock markets or the Internet network. A few concrete examples that can be observed within complex systems are such phenomena like cloud patterns, fluid velocity or immune system.

1.1.2 Self-organization

“A promising approach to the problem of dynamical emergence is provided by the recently developed models of self-organization. Self-organization may be defined as a spontaneous (i.e. not steered or directed by an external system) process of organization, i.e. of the development of an organized structure. The spontaneous creation of an ‘organized whole’ out of a ‘disordered’ collection of interacting parts, as witnessed in self-organizing systems in physics, chemistry, biology, sociology, is a basic part of dynamical emergence.”[9]

Even though sometimes it is hard to decide if the system is self-organizing because it always depends on the level of the observer’s abstraction, self-organizing systems have the following characteristics[10]:

- **Complexity** – Self-organizing systems are always complex systems, which are not organized centrally but in distributed manner.
- **Control parameters** – A set of parameters influences the state and behavior of the system.
- **Systemness** – Self-organization takes place in a system in a coherent whole that has parts, interaction, structural relationship, state, and a border, which delimits it from its environment.

1.1.3 Emergence

“Emergence is a classical concept in systems theory, where it denotes the principle that the global properties defining higher order systems or ‘wholes’ (e.g. boundaries, organization, control, ...) can in general not be reduced to the properties of the lower order subsystems or ‘parts’. Such irreducible properties are called emergent.”[9]

Another popular definition of emergence as simple as possible: “Order arising from chaos”[11]. But despite of that we can define some properties of emergence:

- **No leadership** – There is no conductor that organize other emergents.

- **Dynamic** – Emergents are always in process, continuing to evolve.
- **Coherence** – A stable system of interactions.

A great example of how to imagine emergence is a traffic jam. Defined behavior depends on which level we are thinking. If we see the traffic jam as a simple collection of cars, then each car of that collection is moving towards its final destination, which should mean that the whole collection is moving forward. But if we make the simulation of a traffic jam, someone might be surprised that the traffic jam is in fact moving backwards. This example shows how important it is, on which level of description we are trying to define the system with lots of interacting parts.

1.2 Cellular Automata

1.2.1 Definition

There are several ways how to model complex systems. One of the approaches to model them are cellular automata[3], which is not very common method but very important. Another method for complex system simulation is agent based modeling.[12, 13]

A cellular automaton is a model of a physical system, where space and time are discrete. It consists of a regular grid of cells, where each cell can acquire a finite number of states and also has a set of rules for evolving. Each cell has a defined initial state. Rules are applied at each time step, where each cell updates its status according to the status of the cells in its neighborhood.

"In the *cellular-automaton* model of a dynamical system, the 'universe' is a uniform checkerboard, with each square or *cell* containing a few bits of data; time advance in discrete *steps*; and the 'laws of the universe' are just a small look-up table, through which at each time step each cell determines its new state from that of its neighbors; this leads to laws that are *local* and *uniform.* Such a simple underlying mechanism is sufficient to support a whole hierarchy of structures, phenomena, and properties. Cellular automata provide eminently usable models for many investigations in physics, chemistry, and biology, as well as for experiments in combinatorics and for studies in parallel computation."[14]

Despite of the fact[15] that the concept of cellular automata was described decades ago, there are tools for modeling complex system only for a shorter period of time. Without the use of high-performance computers, cellular automata would not be able to solve real world problems. In the 80s and early 90s, the best way to achieve the best performance of the cellular automaton was to use a specialized hardware which has been designed for this cause. Examples of this machine is CAM (Cellular Automaton Machine[16]). There are also several applications dedicated to simulation of the behavior of the cellular automata (e.g. CAMEL or StarLogo[17]).

Eventhough applications of the cellular automata are very various systems, they can be described by following definitions[18]:

- **Discrete n-dimensional lattice of cells** – We can have one-dimensional, two-dimensional, . . . , n-dimensional CA. The atomic components of the lattice can be differently shaped: for example, a 2D lattice can be composed of triangles, squares, or hexagons. Usually *homogeneity* is assumed: all cells are qualitatively identical.

- **Discrete states** – At each discrete time step, each cell is in one and only one state, $\sigma \in \Sigma$, Σ being a set of states having finite cardinality $|\Sigma| = k$.

- **Local interactions** – Each cell's behavior depends only on what happens within its local *neighborhood* of cells (which may or may not include the cell itself). Lattices with the same basic topology may have different definitions of neighborhood, as we will see below. It is crucial, however, that "actions at a distance" not be allowed.

- **Discrete dynamics** – At each time step, each cell updates its current state according to a deterministic transition function φ: $\Sigma^n \to \Sigma$ mapping neighborhood configurations (n-tuples of states of Σ) to Σ. It is also usually, though not necessarily, assumed that (i) the update is *synchronous*, and (ii) φ takes as input at time step t the neighborhood states at the immediately *previous* time step t - 1.

1.2.2 Types of neighborhoods

Before the demonstration of the typical example of the cellular automaton, it is necessary to define the set of cells that are used for evaluation of the new updated cell's status during time step. The used grid is usually square, but it can also be triangular, hexagon or Voronoi. There are two fundamental types of neighborhood:

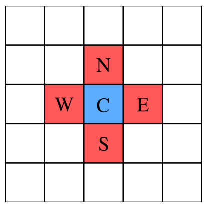
- **von Neumann**1.1a – Cells having influence on given cell are only those sharing whole side with it, e.g., 4 cells when radius equals to one (r=1). A definition of a neighborhood is often provided in the following way[19]. In the case of a von Neumann neighborhood arbitrary r:

$$\begin{aligned} N_N^r(i,j) &= \{\sigma_{k,l}|\ |i-k| + |j-l| \le r\} \\ &= \{\sigma_{i,j}, \sigma_{i-1,j}, \sigma_{i,j-1}, \sigma_{i+1,j}, \sigma_{i,j+1}\}. \end{aligned} \tag{1.1}$$

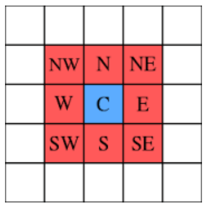
- **Moore**1.1b – Given cell is influenced by the cells sharing side or the corner e.g. 8 cells when radius equals to one (r=1). Radius of the neighborhood can be any positive value. Neighborhood can be defined by the following equation[19] for arbitrary r:

$$\begin{aligned} N_M^r(i,j) =& \{\sigma_{k,l}|\ |i-k| \leq r, |j-l| \leq r\} \\ =& \{\sigma_{i,j}, \sigma_{i-1,j}, \sigma_{i-1,j-1}, \sigma_{i,j-1}, \sigma_{i+1,j-1}, \\ & \sigma_{i+1,j}, \sigma_{i+1,j+1}, \sigma_{i,j+1}, \sigma_{i-1,j+1}\}. \end{aligned} \tag{1.2}$$

- **Random** – Given cell is influenced by a randomly chosen set of cells that can be quite distant from the cell but the size of neighborhood is limited compared to the size of the World.



(a) The von Neumann neighborhood with radius 1.[20]



(b) The Moore neighborhood with radius 1.[21]

Figure 1.1: Two basic types of neighborhood.

1.2.3 The Game of Life

The most famous model among two-dimensional cellular automata is the Game of Life[22], described by the British mathematician John Horton Conway in 1970. The cellular automaton consists of a certain number of cells, where each cell can acquire only two states and four rules which specifies the rules of updating status of every cell each time step. The reason behind the name "Game of Life" is a certain resemblance to living organisms.

The rules applied each time step are following:

- If a cell is alive and has two or three living neighbors, it stays alive for the next generation (survival).
- If a cell is alive and has fewer than two living neighbors, then it dies (under-population).
- If a cell is alive and has more than three living neighbors, then it dies (overpopulation).
- If a cell is dead and has exactly three living neighbors, then it becomes alive (reproduction).

The Game of Life represents a cellular automaton with emergent behavior. When a random initial states of the cells are set and several time steps are applied, a pattern behavior created by the living cells can be observed. Some of those structures are called “gliders”1.2. An structure called “blinker”1.3 consists of three living cells changing every time step from a horizontal cluster to a vertical cluster and back.

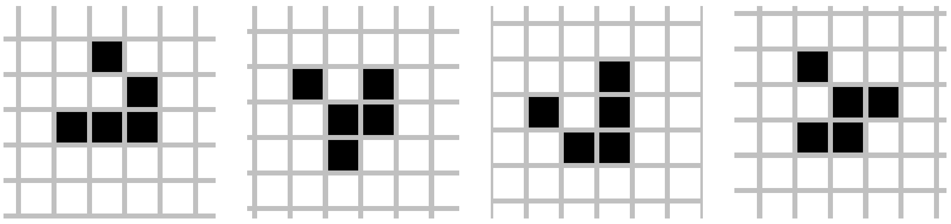


Figure 1.2: Demonstration of an emergent pattern called Glider, which is created by living cells.[23]

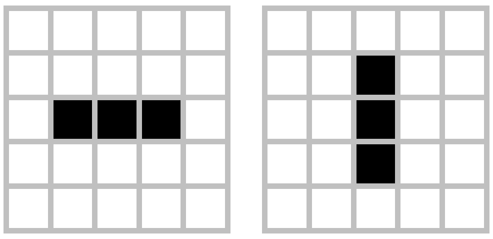


Figure 1.3: The emergent pattern called Blinker.[24]

1.2.4 Implementation

The Game of Life is not hard to implement, but one has to keep in mind the parallelism i.e. the evaluation of each cell is performed simultaneously. It means that the values from the old time step/layer have to be kept until all values from the new step/layer are evaluated. In such way, we easily store the old values separated from the new ones.

Sequential implementation can be written with usage of the two 2D arrays of bool variables, where value “1” demonstrates a living cell and value “0” a dead cell. Algorithm iterating through first array’s cells, counts the number of its neighbors, which is a totalistic rule, decide the new updated state of the cell according to the given rule and save the new value into the second array. When all cells are counted and their new states are saved in the second array, then pointers to these arrays are swapped, so the first pointer always references to the array with new states.

An abstract code follows:

- count_new_states(p_array1, p_array2) – Number of living neighbours for each cell in the first array is counted and new states of the cells are saved into the second array.
- swap_pointers(p_array1, p_array2) – Pointers are switched, so pointer *p_array1* references on the array with the new states.
- print(p_array1) – Voluntary step, cells in the array1 are printed, representing state of the cellular automaton at the end of the time step.

```
int main(void){

        ...    // Initialization of arrays
               // and setting up initial states

        for(i=0; i<NUMBER_OF_TIME_STEPS; i++){
                count_new_statuses(p_array1, p_array2);
                switch pointers(p_array1, p_array2);
                print(p_array1);
}
```

Chapter 2

Simulation of domain growth using cellular automaton

This chapter contains information about the cellular automaton used in the developed application. General information about the set of states and rules used for simulation of domain growth are described here. Author of the implementation of the automaton is Jiří Kroc, who is also the leader of this bachelor thesis. Subject of this thesis is to write a graphical user interface for the simulation. Originally, graphical output is implemented as generation of pgm figures that can be converted into another figure formats. Information provided in this chapter is a summary of the publication "Diffusion Controlled Cellular Automaton Performing Mesh Partitioning" [5] also written by Jiří Kroc.

2.1 Introduction

A model performing mesh partitioning into computationally equivalent mesh parts on regular square lattices using a diffusion controlled cellular automaton is used for the simulation of domain growth. Algorithm is processing information in parallel on a single processor computer. We do not have access to a truly parallel computers. Hence, all computations are in the end processed sequentially on a single processor computer. Implementation of a parallel computer for a single processor computer is often called back-end. In our case back-end and local rule are mixed together. Each cell assigned as a domain seed has a pre-defined amount of growth factor. According to the rules used for migration, the borders of domains are changing each time step. The model uses self-organization principles ensuring convergence. Solution is achieved from any initial configuration.

2.2 General Concept

Each cycle is composed of five steps, which are applied in a specific order - growth of domain, two steps of diffusion of diffusive agents and two steps of migrations of domain boundaries. The initialization has to be done before the simulation starts. Each simulated domain is initialized as a single cell possessing the domain number and a certain amount of diffusive agent. If too low initial value is taken, then algorithm becomes unstable. Other cells are empty or marked as walls. The difference between walls and empty cells is, that a cell that represents a wall cannot be occupied by any domain. Domain can migrate into an empty cell or neighboring domain if difference between migrated cell amount of diffusive agents and amount of diffusive agents of the cell of the migrating domain is bigger than a given threshold. The threshold can be too high, if too low amount of diffusive agent is taken as the initial value.

2.3 Variables

Each cell has the following set of variables:

- **domain-number** – The unique identification of the domain which fills the given cell.
- **growth-factor** – Stores the amount of growth factor.
- **distribution[0], distribution[1], distribution[2], distribution[3]** – Store fractions of *growth-factor* split into the neighboring cells via diffusion - each variable for one neighboring cell.
- **protect** – Flag, if set, given cell is going to migrate and flag prevents the cell to be migrated by domain occupying neighboring cells and avoids mixing of diffusive agents from different domains.

2.4 Rules definition

Definition of all rules applied in the Diffusion Controlled Cellular Automaton:

- **Growth rule G** – Performs growth of domain seeds. For each cell, there are four cells in Von-Neumann neighborhood which are tested against possibility to grow. Cells defined as the wall cells are not tested because they cannot migrate as well as they cannot be migrated.
- **Migration rule M1** – Tests conditions of cell being considered for migration. Variable *protect* could be set to a value that lock the potential migration sub-step against being influenced by other cells until the migration sub-step is finished. A migration threshold is defined.

- **Diffusion rule D1** – At first, the growth factor of migrating domains is deposited into the possible directions. It is considered as a possible direction if the neighboring cell belongs to the same domain.
- **Diffusion rule D2** – Deposited growth factor is shifted into appropriate neighboring cells. This rule finalizes actions done in rule D1.
- **Migration rule M2** – Finalization of the simulation cycle. Movements of domain borders from protected cells are done by specific rules.

2.5 Demonstration

Following figures display the state of domains for given initial positions in specific time with wall cells covering part of the grid.

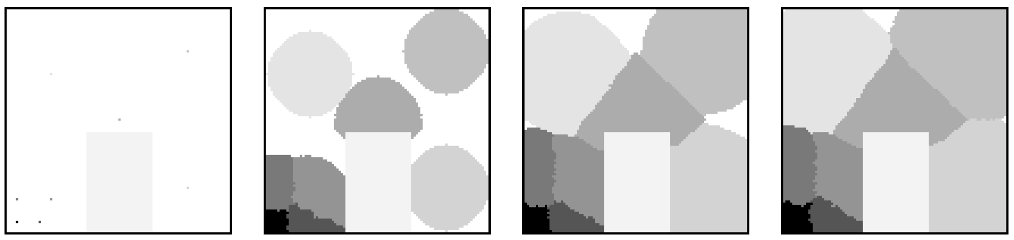
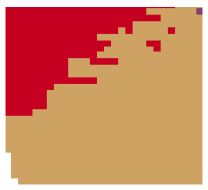


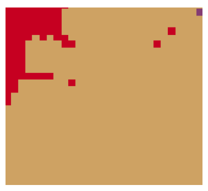
Figure 2.1: Demonstration of domain simulation with eight domain seeds and boundaries (times 0, 1k, 10k, 20k).

2.6 Possible improvement of algorithm/known bug

When the initial amount of growth factor of domains differs by hundreds or more, consistency of the domain can be disrupted by a domain. Very probably the scenario of orphans creation is the following. A domain having a lower amount of growth factor is “eaten” by that with greater amount of GF. Shrinking domain leaves behind “peninsulas”2.2a that can be scissored2.2b by the domain having a greater amount of GF.



(a) The red domain is "eaten" by the yellow domain and creates the peninsula.



(b) The peninsula of the red domain is scissored and an orphan is created.

Figure 2.2: The feature of the algorithm that was not recognized during its design because amount of diffusive agents within all domains was kept equal.

Chapter 3

Software requirements

Software requirements define the behavior of the application, its external interfaces, portability of the software across various platforms or language accessibility.

Requirements are split into three sections. First are functional requirements which define functions and functionality of the software. Second are non-functional requirements, those requirements are not related to the functional aspect of the software. Third, user interface requirements define the behavior of the graphical user interface.

3.1 Functional requirements

- Application shows the simulation model within the application window.
- Allow users to save their configuration into the file and load it from the file.
- Allow users to set the number of domains used in a simulation model and their initial position and growth factor.
- Allow users to draw model boundaries in the simulation window.
- Allow users to pause simulation and save a graph into the file as a picture in PNG, JPEG and BMP format.
- Allow users to save a set of pictures of a simulation model into the chosen directory.
- Application contains two modes, first for skilled users and the second one for beginners, where the prompter will be available for all input windows and simulation model.
- Application validates configuration before every execution.

3.2 Non-functional requirements

- Application is compatible with operating system Windows 7 and newer, and Linux 14.
- Application is prepared for the various language localization, e.g. all labels are in tags allowing translation.
- Release package contains all necessary files and libraries for executing the application.
- Application is available from websites `www.researchgate.net` and `www.sourcefourge.net`

3.3 User interface requirements

- When user violates the rules for drawing domains and walls, the application shows message box with an error message.
- Domain growth factor of each domain can be set from the main window.
- Scroll area is available when there are too many domains to fit them into the window without scrolling area.
- Each button shows tips with its functionality description.

Chapter 4

Analysis of the application

After the definition of the software requirements, it is now possible to proceed to the second part of an analysis of the program. The first half is about defining the users, which will most probably use the application to conduct their work or use it for their studies. Each user definition is complemented with the description of their expected knowledge and abilities. The second half is describing the most often actions, which will be taken by the users. Each use case contains scenario of the actions done in specific order.

4.1 Typical user

Users of this application are considered to have at least some basic knowledge of the issue of the complex systems. There are two types of the typical users:

- Student – Student of the university with a technical orientation, trying to understand complex systems, cellular automata and self-organization. He/She has great skills in operating computers and he/she can easily get familiar with various applications. He/She has a lack of some knowledge of the action he/she is taking by operating the program, but he/she is willing to learn the theory how the cellular automaton, used in this application, works. He/She has some basic knowledge about programming and is able to implement his/her own ideas to improve open-source applications.

- Researcher – Scientist with a great knowledge of the complex systems. He/She understands how cellular automata work and how the rules are applied. He/She has average computer skills. He/She does not have time to learn how to work with applications that are too complicated to control. He/She demands possibilities to save his/her current work or save an image of the cellular automaton to use it for some following research. He/She has no knowledge about programming.

- Teacher – Teacher with an average knowledge about complex systems and cellular automata. He/She is able to control the basic application but he/she has a minimal knowledge about programming. He/She wants to use useful programs as learning tools for his/her students.

4.2 Use cases

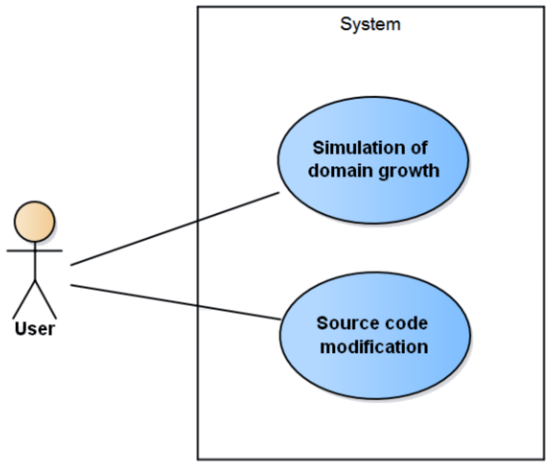


Figure 4.1: Role "User" and the actions (events) which can be done within the application.

- Simulation of domain growth with saving results of the work.

  1. Scenario starts at the moment when user starts the application.
  2. User marks cells in the drawing window as seeds of domains.
  3. User fills the amount of domain growth factor for each domain.
  4. User draws boundaries in the cell window.
  5. User starts the simulation.
  6. If user is not satisfied with the model, he/she modifies the position of the domains and its domain growth factor.
  7. User saves configuration of the model for further use.
  8. User saves images of the simulation.

- Modification of the source code.

  1. Scenario starts when a user opens a project in Qt Creator
  2. User adds his/her own methods in existing classes or create his/her own class in a separate file, in this case he made himself/herself sure, that the new file appears in a *.pro file.
  3. User compiles the program in debug mode and check if the program works correctly without crashes.
  4. User compiles the program in release mode.
  5. User makes release package by adding libraries (*.ddl files).

Chapter 5

Design of the application

Design of the application can be split into the two parts:

- Graphical user interface
- Cellular automaton

5.1 Design of graphical user interface

Thanks to the knowledge gained during the analysis, it was possible to design a prototype of the application's main window. For the modeling of the wireframe[5.1], Balsamiq[1] application was used. The main window was split into several parts:

- upper menu bar,
- left part of the window displaying all domains and their specifications,
- drawing window,
- buttons for controlling the simulation.

All elements can be reached directly from main window. Upper menu bar is split into the three parts, each part containing functionality creating logical unit.

Drawing widget has two modes, one for drawing domains and wall cells. The second mode is used for the simulation rendering.

[1]`https://balsamiq.com/`

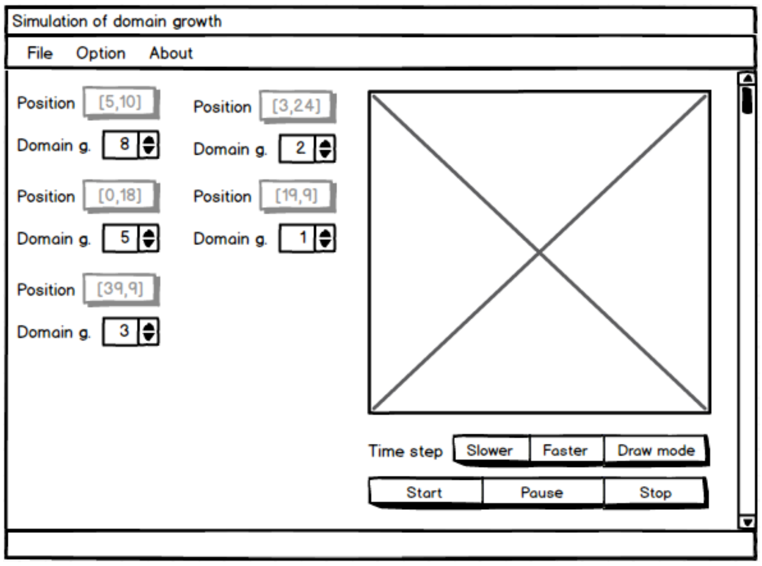


Figure 5.1: Original design of the main window.

5.2 Cellular automaton

The code implementing CA was programmed by Jiří Kroc. Originally, the programming language used for the implementation was C.

It was decided to rewrite the implementation of the cellular automaton to the C++ language. Modifications done in the source code of CA do not change its functionality. As a result, new class dedicated to the cellular automaton was created and performs all necessary actions (initialization, all steps of the simulation) needed for the simulation.

5.3 Class design

Before the beginning of the implementation classes were designed, which implements the application's functionality and splits it into logical parts.5.2 The application was implemented in accordance with this design.

- **MainWindow** – Represents the main window of the application. It is responsible for application start and creation of the graphical interface (e.g. menu bars, layouts, drawing window). It connects elements of GUI (e.g. buttons, labels) with appropriate methods.
- **DrawWidget** – A custom widget inherited from the basic widget (QWidget). It represents a drawing window displayed in the main window of the application. This class implements methods for domain and boundaries drawing. Values from the window are used in the cellular automaton as parameters for simulation.

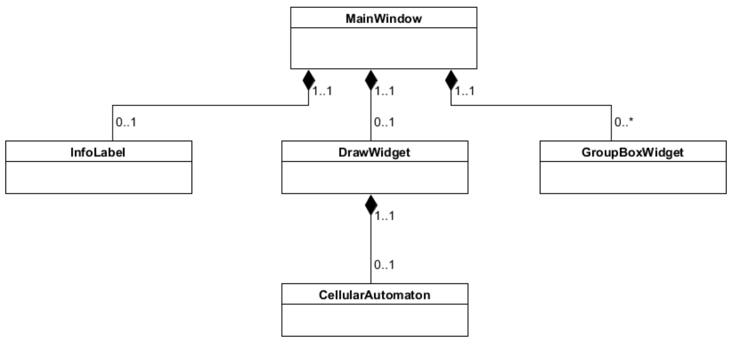


Figure 5.2: Class diagram showing how classes are composited.

- **GroupBoxWidget** – A custom widget inherited from Qt basic class QGroupBox. The class represents values for each domain drawn in the drawing window. It contains basic elements like label, buttons and line edits. In each instance of this class, information about one domain is stored.
- **CellularAutomaton** – Methods in the class were originally created by Jiri Kroc and refactored for the purposes of the application by the author of this thesis. The class represents the cellular automaton and back end part of this program. At the beginning of the simulation it loads values from GroupBoxWidget and DrawWidget.
- **InfoLabel** – A custom widget inherited from Qt class QLabel. It represents label with information displayed in the main window when there is no domains marked in the drawing window.

The design counts also with one additional header file containing the definition of the initial values. This file makes their further modification easier, so for the change of the value, user does not have to search throughout the whole project files.

5.4 Technologies

An important part of the software development process is to choose the right tools for the implementation and the way how to implement the resulting application. Both parts are examined in detail in the following sections.

5.4.1 Programming language

The decision which programming language will be used for the implementation was made by the leader of this thesis in the thesis assignment. The language is C++, same as the language used to create the back end.

- C++ – Object-oriented language, originally developed as a extension for the language C in 1970s. C++ is a popular language with many open source libraries and frameworks.

5.4.2 Frameworks and libraries

The extensions were used to help to implement graphical user interface, suitable extensions for this cause are:

- **Qt**[2] – Qt is a multi-platform framework for creating graphical user interface in C++. It is very popular thanks to its enormous documentation[25]. To facilitate work, users can use Qt Creator and Qt Designer for designing their applications.
- **GTK+**[3] – GTK+ is a multi-platform toolkit written in C, formerly known as GIMP Toolkit. GTK+ contains a set of widgets suitable for creating graphical user interface. It is licensed under the terms of the LGPL i.e. GTK+ is open source toolkit. For easier design implementation there are several applications which offer GUI Designer.

The decision, that used framework will be Qt was made by the leader of the thesis.

5.4.3 Development environment

Based on the fact, that the application will be written in Qt, **Qt Creator**[4] will be the development environment.

Qt Creator is developed by *Qt Project* as well as the whole framework Qt. It includes debugger, syntax highlighting and autocompleting. Qt Creator includes Qt Designer, which is an application for designing graphical user interface from Qt widgets.

[2]`http://www.qt.io`
[3]`http://www.gtk.org`
[4]`http://www.qt.io/ide/`

Chapter 6

Realisation

In this chapter, the important and interesting parts of the implementation of the resulting application are covered. The chapter doesn't contain any information about regularities or general rules for implementation in language C++ or in Qt. The information about classes and methods needed during the implementation were found in the official documentation[25] of the framework Qt.

6.1 Drawing window implementation

The drawing window6.1 is implemented in the class DrawWidget. This class represents, in Qt terminology, a *custom widget*, which means that the class inherits most of its behavior from standard Qt class (in this case it inherits from QWidget class), but there are some changes in functionality. It is used to represent a 100x100 grid. Each cell can represent a domain, a wall or an empty cell. For implementation, it was crucial to reimplement the following methods:

- **mousePressEvent** – The method called when user press the left mouse button on the top of the widget. The method at first saves the position of the cursor to the class variable *firstPos*, which is an instance of the QPoint class, a class suitable for saving coordinates. The current state of pixels in image is saved to the backup array of pixels. This array is used for rendering the rectangular boundaries.

- **mouseMoveEvent** – The method called every time when user holds the left mouse button and moves the cursor over the drawing widget. The first thing which is done, is to revert changes in pixels to the state when the mouse button was pressed. Then, a rectangle is drawn from the initial position with coordinates saved in variable *firstPos* to the position where mouse cursor is located.

- **mouseReleaseEvent** – This method is called when the mouse button is released. At first, method checks if the initial position is not the same as the mouse release position. If the position is the same and the cell is empty, a domain is created. If the position is the same but the cell is not empty, a message box with a warning is displayed and no action is done.

  If the initial position and the mouse release position differs, then the cells which are being between the initial position and the mouse release position are checked whether there are empty or representing the domain. Then, a rectangle is drawn into the drawing widget.

- **mousePaintEvent** – This method repaints the whole widget. It is called at the end of all previous methods, where it is invoked by the command *update()*.

For the initial visualization of the widget, it was decided to use white and yellow pixels, thanks to that, it is easy to recognize the grid and it makes marking an individual cell easier. The black color is used for representing domains and the boundaries can be recognized by its red color. Each color can be easily changed by rewriting its value in the class constructor.

One of the class variables is a pointer to the instance of the *CellularAutomaton* class. The instance of that class is created during the start of the application. Before the beginning of the simulation, the properties of cell filled their counterparts in CA class.

The colors used in the simulation are defined in the method named *setColors*, the first twenty colors are hardcoded.

Listing 6.1: Example of hardcoded values.

```
colArray[0] = QColor(255,179,0);        //vivid_yellow
colArray[1] = QColor(128,62,117);       //strong_purple
colArray[2] = QColor(255,104,0);        //vivid_orange
```

Additional colors are set by counting rgb values, however the colors created by this generator differ so little that it was necessary to use hardcoded values. The number of additional colors is set by a parameter. Its default value is 40 but can be easily modified in a data file.

6.2 Simulation of the domain growth

The simulation starts when user press the button “Start”. To allow user to control the simulation while computations of each time step are running, the usage of the threads was considered. Because of the goal to keep the application as simple as possible, Qt class “Timer”[5] was used instead.

[5]http://doc.qt.io/qt-5/qtimer.html

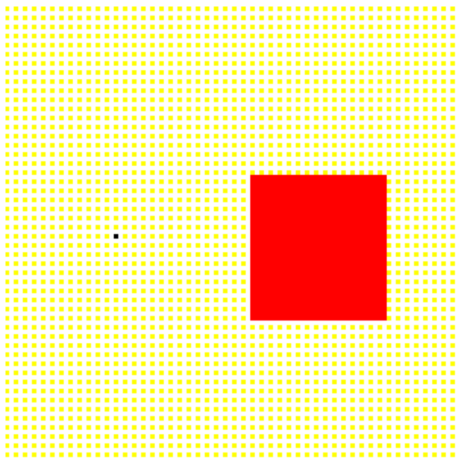


Figure 6.1: The look of the drawing window with domain and boundaries

Timer class handles a repetitive calling of the method to which is connected. After the creation of the connection, the timer can be started. An argument taken for calling "timer->start(int)" sets how often the connected method will be called. In the example 6.2, the method "startSimulation()" would be called every 1000 milliseconds.

Listing 6.2: Example of QTimer initialization.

```
QTimer *timer;
timer = new QTimer(this);
connect(timer, SIGNAL(timeout()),
                this, SLOT(startSimulation()));
timer->start(1000);
```

At the beginning of the simulation at least one domain has to exist. Then values representing domains in the cellular automaton are filled. These values (e.g. coordinates of the domains and value of domain growth factor) are taken from all group boxes and stored in variables of the CA, where they are used afterwards. Other CA's variables needed for the simulation are set as well. Then, the timer can be started.

The timer handles calling the method of the DrawWidget class, values of the CA are loaded into the values of the drawing window after every time step is done by the automaton. As the timer calls the method only once per given amount of time, we can use buttons to control the simulation. Every time the simulation is paused or stopped the timer is stopped as well.

6.3 Domain adding

The implementation of the domain adding shows the complexity of Qt. The creation of the domain starts when a user marks a single cell in the drawing window. It is checked whether the cell is not already occupied by another cell or wall, if not, signal "createDomain(int x, int y)", where integer x and y represents a position of the domain in the drawing window, is sent out. This signal is caught in MainWindow class which causes calling of the AddDomain method.

At first, a box with the specification of the domain is created and its line edits are filled with the value and the erase button is connected with the appropriate method. When the beginner's mode is on, the value of the growth factor is filled with the recommended value 6.3. At the end of the method, the size of the scroll area is checked and changed if needed.

Listing 6.3: Calculation of the recommend value of the growth factor. This value is used when the beginner's mode is on. Default value of GF_PER_CELL_VALUE, DRAW_WINDOW_X_SIZE and DRAW_WINDOW_Y_SIZE is 100.

```
int calculatedValue  = (GF_PER_CELL_VALUE
                         * DRAW_WINDOW_X_SIZE
                         * DRAW_WINDOW_Y_SIZE)
                         /number_of_domains;

    calculatedValue  = calculatedValue
                         - GF_PER_CELL_VALUE
                         * (2*DRAW_WINDOW_X_SIZE - 2
                         * (DRAW_WINDOW_Y_SIZE-2));
```

At the beginning of the resize scroll area method height of the current group boxes is counted. The group boxes are displayed in two columns, so their amount must be divided by 2 and because of the rounding, the amount of the domain must be incremented by 1. Because of the margin size of the group boxes, value 9 is added to their height. Then, the counted value is compared with the size of the main window and the bigger value is used for the new size of the scroll area. If the counted value of the group boxes height is used, then value 30 is added because of the top margin of the layout, where group boxes are stored.

Listing 6.4: Implementation of the scroll area resize.

```
void MainWindow::resizeScroll(){
    int groupBoxHeight =
        (currentDomains+1)/2*(GROUP_BOX_HEIGHT+9);

    if(groupBoxHeight>SCROLL_HEIGHT){
        ui->scrollAreaWidgetContents
                ->setMinimumHeight(groupBoxHeight+30);
    }else{
        ui->scrollAreaWidgetContents
                ->setMinimumHeight(SCROLL_HEIGHT);
    }
}
```

6.4 Configuration saving

Saving of the configuration can be invoked within the main window of the application from the menu bar named *File*. At first, the file dialog appears, where user can choose the directory in a file system where the configuration file will be saved. The default extension for the saved file is ".ca". After validation that the string where the absolute address is supposed to be is not empty, the program proceeds to the next step. The output file stream is open for the chosen file. The first item saved into the file is a number of the domains. The following items are the specifications of each domain (e.g. their position and domain growth factor), if the value of a domain growth factor is empty, then 0 is saved into the file. The information about one domain is always on a single line and is saved in format "xx,yy:gf", where:

- xx – X coordinate of the domain.
- yy – Y coordinate of the domain.
- gf – Value of the growth factor.

After the domain specification is saved, the method from the drawwidget is invoked. The return value of this method is one string with the information about all pixels. The empty cells are marked with a value of 0 and not empty cells are marked as 1.

The configuration file is the text file that allows user's modification from any text editor. Unfortunately, this option can cause that user will corrupt the structure in which the data are saved. For this reason the validation of the file during the load is implemented. The validation should prevent the application to crash or to load corrupted data.

Listing 6.5: Example of the configuration file.

```
3
13,12:737
38,38:5373
22,65:5775
000000001111111111111100000000000000...
000000001111111111111111111111000000...
000000001111111111111111111111000000...
...
```

6.5 Saving image of the simulation

Saving image is used to save the picture of the simulation in JPG, PNG or BMP format. This action can be invoked by clicking on "Save as picture" button, which is in Option tab bar.

This feature was easy to implement thanks to the Qt's possibilities. At first, save file dialog is displayed and user is asked to write the name of the image file and choose an extension for the file. The default extension is JPG. After validation that the name of the file is filled, the image is saved.

Listing 6.6: Whole method implementing saving picture

```
bool MainWindow::savePicture(){
    QString fileName = QFileDialog::getSaveFileName
        (this,
         tr("Save Image"), ".",
         tr("JPEG (*.jpg);;PNG (*.png);;BMP (*.bmp)"));

    if (fileName.isEmpty())
        return false;

    ui->drawwgt->grab().save(fileName);

    return true;
}
```

In the application, there is also implemented a function for saving multiple pictures in particular time steps, this function is working similarly to the function for saving a single picture but it invokes run of the automaton from the initial position.

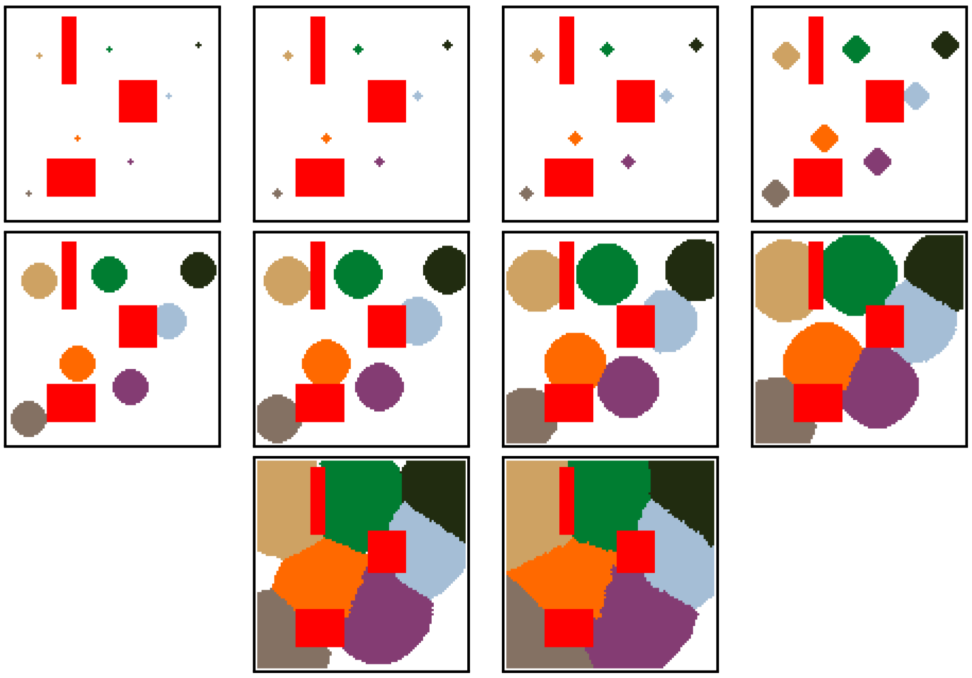


Figure 6.2: Set of the 10 pictures saved in defined time steps via the implemented function in the application (time step 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048).

Chapter 7

Testing

Even during the implementation, the application was continuously tested, every time new functionality was added. This chapter contains information about how were the final user tests handled.

7.1 Testing environment

Tests were run on a computer with the operating system Windows 10. Before the start of the test, each testing subject received the list of tasks7.2, which are supposed to be done in a defined order. These tasks cover the main functionality of the application.

When the tests start, the testing subject cannot communicate with the tests supervisor. The supervisor is observing the tester and writing notes of whether the tasks are successfully completed. If a critical defect appears in the application, the test is stopped and starts again after the successful bug correction.

After the end of the test, the testing subject is interviewed:

- if he/she had any difficulties to understand the tasks,
- if any task was hard to complete,
- if the testing subject has any suggestions for the application improvement.

7.2 Scenario

At the beginning of the test, the sheet with following tasks was distributed to the testers. Goal of the testers were to complete the tasks in the given order. Tasks are precisely described to avoid any misunderstanding.

1. Create six domains.
2. Draw boundaries in the model. The area of the created boundaries has to be bigger than twenty cells.
3. Start the simulation.
4. Decrease the speed of the simulation.
5. Pause the simulation.
6. Save a single picture of the current state of the simulation window.
7. Stop the simulation.
8. Save the configuration of your model into a file.
9. Clean your workspace, so there are no domains or boundaries. Do not delete the domain by erase button.
10. Load the configuration from the file, so your workspace will look like during the saving operation.
11. Delete two out of six domains.
12. Erase half of the boundaries, so the cells are empty.
13. Save a set of the pictures.

7.3 Test evaluation

The application was tested by five users. Each user saw the application for the first time, a few moments before the test started, so he/she did not have an opportunity to get familiar with the application nor to read the user's manual where it is possible to find help for each task in the test scenario.

A part of the users described the first task as difficult, because they did not understand how to mark the cell in the drawing window. A label informing about the domain creation is already present in the application, for that reason more details about the domain creation are described in the user's manual.

In the ninth task, testers described the label *New* on the button in menu bar as confusing. This button is used to clear the workspace, e.g. to erase all domains and all wall cells. For that reason the label on the button was changed to *Clear workspace*.

As it was expected, the twelfth task was the most difficult to fulfil. Users found it hard to find the button for changing modes between drawing boundaries and drawing empty cells. The actions taken to improve the visibility of the button are described in the next section.

The rest of the task were completely fulfilled and users found them easy to complete. To avoid users confusion, the application will be distributed with the user's manual, where each functionality is described in detail.

7.4 Changes in the application

After the test evaluation it was clear that a switcher between drawing modes has to be highlighted, otherwise users will have a problem to understand this functionality. Other two cases, which were causing problems to several testers, where fixed by changing the label text.

To highlight the switcher button, icons were added7.1, so it is now easier to distinguish in witch mode the application is. Every time the button is pressed the icon is changed, signalizing which mode is currently used. Also the status tip, which is located on the bottom of the application window was modified, now it is displaying more precise information which mode is used.7.2

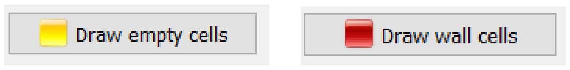


Figure 7.1: New look of the switch button.

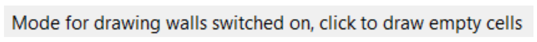

Mode for drawing walls switched on, click to draw empty cells

Figure 7.2: The prompter for the switch button.

Conclusion

The goal of this thesis was to design and implement a graphical user interface for a cellular automaton simulating domain growth. Complex systems and cellular automata are difficult fields of the science, so it was important to make the graphical interface simple enough to facilitate the work for the people which are interested in this issue. I believe that the application fulfilled this requirement. After the discussion with the leader of the thesis we wrote down all requirements necessary to make the application accessible for students, researchers and even teachers. According to the requirements, a wireframe representing the resulting application was created and classes needed for the implementation were designed. The implementation was done without any bigger issues; problems like how to involve the implementation of the cellular automata into the application or how to allow users to control the application while the simulation is running were solved in a few days. Subsequent user testing were done without discovering any critical defects, only a few minor bugs were discovered, but they were easily fixed by changing a few lines in the source code.

Currently, the source code of the application is placed into the ResearchGate and the SourceForge website and can be freely downloaded. Thanks to the application, users can easily simulate the domain growth and use the images of the simulation in their work.

The application fulfills all the requirements defined in the chapter 3 “Software requirements”. It allows user to define his model, set domains and to draw boundaries. Image of the simulation can be saved as a single picture or as a set of the pictures. Configuration of the cellular automaton can be saved or loaded from the file. User can switch between the beginner mode and professional mode, so the amount of the domain growth factor is set to the recommended value.

Future work

Even though the application is finished and the requirements are fulfilled, there are still several possibilities how to improve the application. The source code is free to modify so anyone with the knowledge of the programming in C++ can improve the application.

- When the difference between the amounts of the diffusive agents of two domain is big enough, “weaker” domain can be tore apart and the orphan cell is created. This bug is described in 2.6 “Possible improvement of algorithm/known bug”.
- To allow non-English speakers to use the application, it would be nice to offer various language localizations. During the implementation of the application this option was considered and each label displayed in the application is written in *tr tags* allowing easy translation. By using the Qt function *lupdate*, translation form can be created.
- To allow users to set the size of the drawing window. Currently, the size of the drawing window is 100x100 cells. The idea is to allow users to modify this values according to their needs.

Bibliography

[1] Mitchell, Melanie. *Complexity: a guided tour, 2009.* New York, Oxford University Press.

[2] Vichniac, Gérard Y. *Simulationg physics with cellular automata: Physica 10D, 1984.* Amsterdam, North-Holland.

[3] Ilachinski, Andrex. *Cellular automata: a discrete universe., 2002.* Singapore, World Scientific.

[4] Hoekstra, Alfons G., Jiri Kroc, and Peter M. Sloot. *Simulating Complex Systems by Cellular Automata, 2010.* Berlin, Springer.

[5] Kroc, Jiří. *Diffusion Controlled Cellular Automaton Performing Mesh Partitioning, Lecture Notes in Computer Science, volume 3305, 2004.* Helsinki School of Economics, Mikkeli, 1-6.

[6] Meadows, Donella H. *Thinking in systems: a primer, 2009.* London, Earthscan.

[7] Gabbay, Dov M. *Philosophy of complex systems, 2011.* Oxford Waltham, North Holland.

[8] Heylighen, Francis. *Encyplopedia of Library and Information Sciences: Complexity and Self-organization, 2009.* Free University of Brussels, Brussels.

[9] Heylighen, Francis. *Self-organization: emergence and the architecture of complexity, 1989.* Free University of Brussels, Brussels.

[10] Arshinov, Vladimir, and Christian Fuchs. *Causality, emergence, self-organisation, 2003.* Moscow, NIA-Priroda, 6-8.

[11] Holmann, Peggy. *Engaging Emergence: Turning Upheaval into Opportunity, 2010.* San Francisco.

[12] Cossentino, Massimo, et al. *Handbook on agent-oriented design processes, 2014.* Berlin, Springer. Print.

[13] Wilensky, Uri, and William Rand. *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo, 2015.* Cambridge, Massachusetts London, England, The MIT Press, 2015. Print.

[14] Toffoli, Tommaso. *Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics: Physica 10D, 1984.* Amsterdam, North-Holland, 117-127.

[15] Talia, Domenica. *Cellular Automata + Parallel Computing = Computational Simulation, 1997.* Universita della Calabria, Rende.

[16] Toffoli, Tomasso. *CAM: High-performance Cellular-automaton Machine: Physica 10D, 1984.* Amsterdam, North-Holland.

[17] Begel, Andrew and Klopfer, Eric. *StarLogo: A Programmable Complex Systems Modeling Environment for Students and Teachers: Artificial Life Models in Software, 2005.* London, Springer.

[18] Berto, Francesco and Tagliabue, Jacopo. *The Stanford Encyclopedia of Philosophy: Cellular Automata [online], 2012.* [cit. 16.2.2016] Available from: `http://plato.stanford.edu/entries/cellular-automata/`

[19] Hoekstra, Alfons G., Jiri Kroc, and Peter M. Sloot. *Simulating Complex Systems by Cellular Automata: Introduction to Modeling of Complex Systems Using Cellular Automata, 2010.* Heidelberg, Springer.

[20] Berto, Francesco and Tagliabue, Jacopo. *Von_neumann_neighborhood_with_cardinal_directions.svg* [online] [cit. 16.2.2016] Available from: `https://commons.wikimedia.org/wiki/File:Von_neumann_neighborhood_with_cardinal_directions.svg`

[21] Contributors of Wikipedia. *Moore_neighborhood_with_cardinal_directions.svg* [online] [cit. 16.2.2016] Available from: `https://en.wikipedia.org/wiki/Moore_neighborhood`

[22] Gardner, Martin. *The fantastic combinations of John Conway's new solitaire game "life", 1970.* Scientific American 223, 120-123.

[23] Contributors of Wikipedia. *Game_of_life_animated_glider.gif* [online] [cit. 16.2.2016] Available from: `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`

[24] CONTRIBUTORS OF WIKIPEDIA. *Game_of_life_blinker.gif* [online] [cit. 16.2.2016] Available from: `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`

[25] QT PROJECT. *Qt Documentation.* [online][cit. 21.2.2016] Available from: `http://doc.qt.io/`

Appendix A

Acronyms

GUI	Graphical user interface
CA	Cellular automaton
CAs	Cellular automata
CSs	Complex systems
GF	Growth factor
MPC	Massive parallel computation

Appendix B

User's manual

B.1 Introduction

This document is a user's manual for the application "Cellular automaton simulating domain growth". It was created as a part of the bachelor's thesis. The application is written in C++ with usage of Qt framework, it is an open source application and it is free to use or modify. The application can be used in Windows and Linux and does not require the Internet connection for its execution.

The primary use of the application is to create a simulation of the domain growth. The application allows to mark cells as the domains with a definition of their amount of diffusive agents and to draw boundaries. The application also allows you to save a configuration of the created model, a single image of the drawing window or the whole set containing of 10 pictures. The application offers a brief explanation of the basic elements after switching on the school mode.

B.2 Program installation

On **Windows**, it is possible to download a precompiled application from the author's profile in ResearchGate. After unpacking the archive, you can open a file **CellGen.exe** to run the application. If you intend to build the application on Windows by yourself, the process is not that simple. The best way is to download and install *Qt Creator*, which is a freeware IDE, suitable for programming Qt applications.

On **Linux**, unzip the archive, which can be downloaded from author's profile in ResearchGate as well. In order to compile and run the program, it is necessary to have Qt 5.5 (or higher) with qmake and g++. Then, after unzipping the package, it should be possible to open a terminal in the folder that contains the program and to write the following commands:

```
qmake
make
./DomainGrowth
```

B.3 Initial screen

B.3.1 Main elements

After start of the application, the main window should look like this.B.1

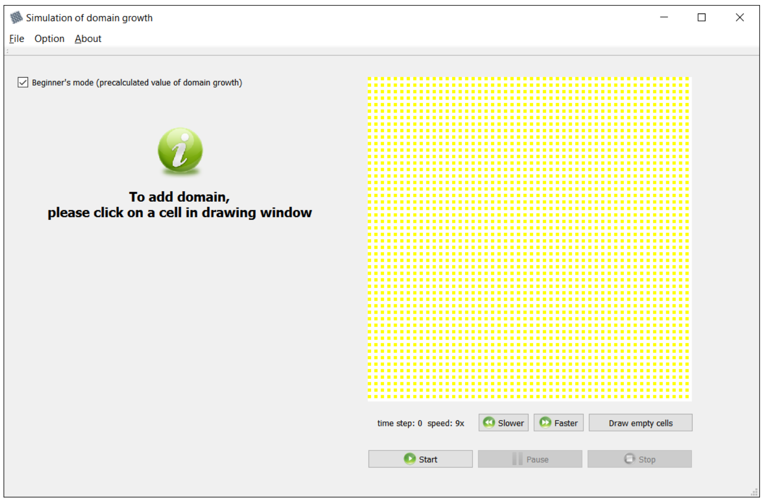

Simulation of domain growth
File Option About
Beginner's mode (precalculated value of domain growth)
To add domain,
please click on a cell in drawing window
time step: 0 speed: 9x
Slower
Faster
Draw empty cells
Start
Pause
Stop

Figure B.1: Visage of the window after the application start

The left half of the screen is used to display the domains specifications after they are marked in the drawing window.

By the drawing window it is meant the widget on the right side with yellow-white grid. Each square represents one cell/pixel. Below the drawing window there are several simulation control buttons.

The check box in the top left corner controls whether the value of the domain growth is filled with the recommended value or if it is filed with zero.

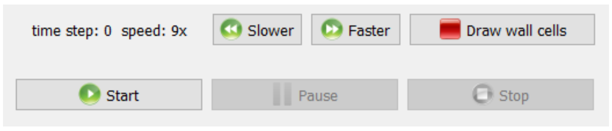


Figure B.2: Control buttons before the simulation starts

- **Time step, Speed** – Information label about a state of the simulation. "Time step" is showing how many time steps were done since the beginning of the simulation. "Speed" signalizes how fast the simulation is,

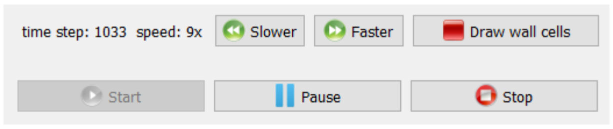


Figure B.3: Control buttons during the simulation

the default speed of the simulation is 9, and the speed can be changed from 1 to 10, where 1 is the slowest speed.

- **Slower** – Button to decrease the speed of the simulation.
- **Faster** – Button to increase the speed of the simulation.
- **Draw empty cells/ Draw walls cells** – A switch between modes, the default mode allows you to draw boundaries in the drawing window. After pressing the button, it is possible to draw empty cells.
- **Start** – Starts the simulation, it is possible when at least one domain is marked. After pressing the start button, pause and stop buttons become available.
- **Pause** – When enabled during the simulation, then it pauses the simulation. During the simulation or during the pause, it is not allowed to modify the drawing window.
- **Stop** – Stops the simulation and returns the drawing window into the drawing mode.

B.3.2 Menu bars

There are three tabs in the main menu bar: File, Option and About.

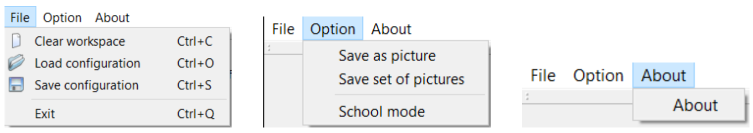


Figure B.4: Menu bars

- **Clear workspace** – Open a new clean application. The user is warned if the application is going to erase his changes.
- **Load configuration** – Load configuration from *.ca file.
- **Save configuration** – Save configuration to the *.ca file.
- **Exit** – Closes the application.

- **Save as picture** – Saves the visage of the drawing window.
- **Save set of pictures** – Saves 10 pictures into the chosen directory. Pictures of the drawing window are captured in various part of the simulation.
- **School mode** – Turns on help buttons for the domain specification.
- **About** – Displays a window with a brief information about the application and the authors.

B.4 Setting up the simulation

B.4.1 Domains drawing

To run the simulation, at least one domain has to be set. The domain is created by clicking on one single cell. Then, the cell changes the color to black and on the left side of the window a box with the information about the domain appears.B.5

B.4.2 Boundaries drawing

To draw the wall cells, it is necessary to mark more than one cell. When the left mouse button is pressed on the top of an empty cell, it is necessary to move the cursor on the top of another cell. After releasing the mouse button, boundaries are drawn. It is allowed to draw a wall cell or a domain only on the top of the empty cells.

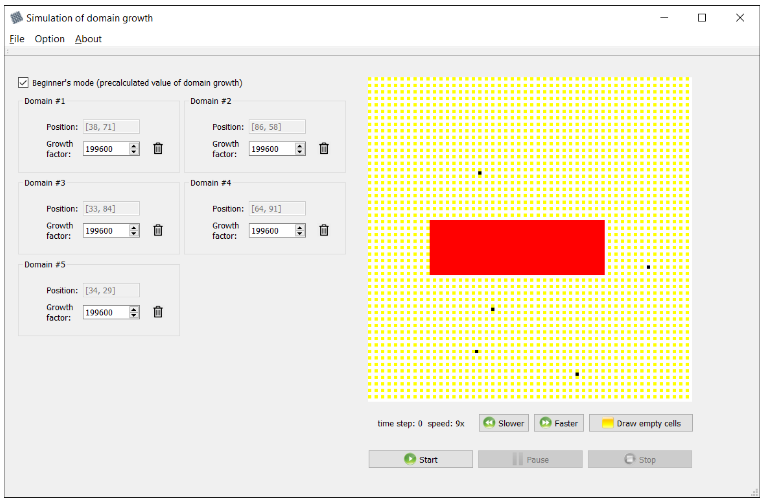
It can happen that wrong cells are marked as walls. It is possible to hit the button "Draw empty walls" which changes drawing mode and instead of drawing wall cells, the empty cells are drawn.

B.4.3 Setting up the domain

After the domain is drawnB.6, it is possible to set its value of the domain growth by writing the value in the appropriate line edit. The first line edit is not allowed to modify and shows the coordinates of the domain. By clicking on the picture of the trash bin, the domain is erased.

B.5 Running the simulation

The simulation is started by clicking on the button "Start"B.2, then the drawing window switches from the drawing mode into the simulation mode. It is possible to control the speed of the simulation, to pause it and to stop it.



☑ Beginner's mode (precalculated value of domain growth)

Domain #1
Position: [38, 71]
Growth factor: 199600

Domain #2
Position: [86, 58]
Growth factor: 199600

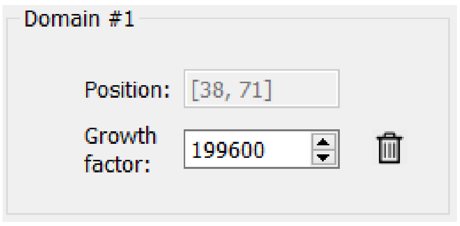
Domain #3
Position: [33, 84]
Growth factor: 199600

Domain #4
Position: [64, 91]
Growth factor: 199600

Domain #5
Position: [34, 29]
Growth factor: 199600

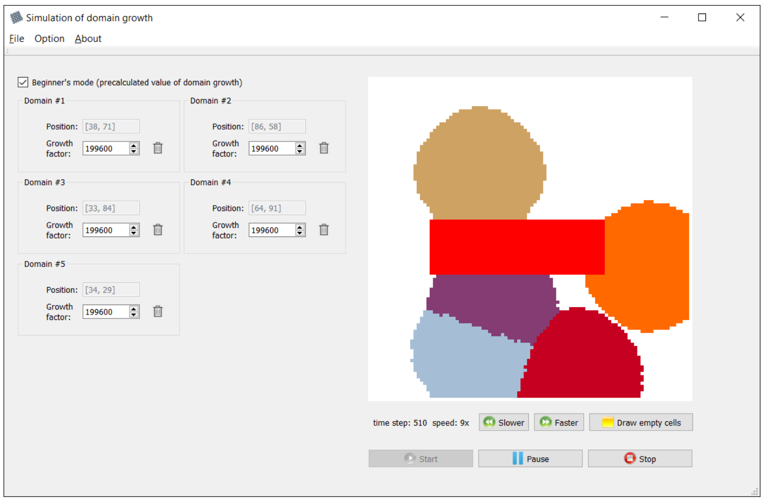
time step: 0 speed: 9x

Figure B.5: Screen with domains and drawn wall cells



Domain #1
Position: [38, 71]
Growth factor: 199600

Figure B.6: Detail of the domain



☑ Beginner's mode (precalculated value of domain growth)

Domain #1
Position: [38, 71]
Growth factor: 199600

Domain #2
Position: [86, 58]
Growth factor: 199600

Domain #3
Position: [33, 84]
Growth factor: 199600

Domain #4
Position: [64, 91]
Growth factor: 199600

Domain #5
Position: [34, 29]
Growth factor: 199600

time step: 510 speed: 9x

Figure B.7: Screen during the running simulation

B.6 Use case stories

This section contains several examples how the simulation of the domain can be seen.

- **ants colonies** – Domains can be seen as ant-colonies. The initial position (seed) sets the place where an ant colony is created. During the time, colonies are trying to grow accordingly to their values of domain growth factor, which can represents size and vitality of the colony. During their attempts to grow, they can meet with obstacles, inappropriate soil (walls), which cannot be used for spreading the colony. Ants can also meet other ants from a different colony and of course only the stronger colony can hold their soil and occupy the soil of the other colony.

- **balloons with water** – Let's imagine a big carton box and several balloons filled with water. When you put the balloons into the box, for some time they are competing with each other on the bottom of the box. After some time of competition, movement of the borders of balloons is minimal and the balloon with the biggest amount of water occupies the biggest area.

- **bacterial communities** – At the beginning, each bacterial community is spreading from its initial position (seed). The speed of the spreading is in accordance with its vitality (growth factor in the application). The more vital bacterial community growths at the expense of the less vital one.

Appendix C

Programmer's manual

C.1 Introduction

This document is a programmer's manual for the application "Cellular automaton simulating domain growth". It was created as a part of the bachelor's thesis. The application is written in C++ with usage of Qt framework, it is an open source application and it is free to use or modify. The application can be used in Windows and Linux and does not require an internet connection for its execution.

This manual contains information how the source code is written. It gives information about the used notation for comments, new methods or classes. One section is dedicated to the description of the structure of the program. The next chapter contains information about how to compile the modified program and how to prepare the release package.

The primary use of the application is to create a simulation of the domain growth. The application allows to mark cells as the domains with a definition of their amount of diffusive agents and to draw boundaries. The application also allows you to save a configuration of the created model, a single image of the drawing window or the whole set containing of 10 pictures. The application offers a brief explanation of the basic elements after switching on the school mode.

C.2 Programmer's convention

C.2.1 Comments

The comments are written in English. They contain a brief information about each method and each class. The comments are written in the header files, in the source files, they are written only if it is necessary for the code intelligibility.

C.2.2 Classes

Each class comment has to contain a brief information about what is the class representing. If it is appropriate, there is an additional information about the class functionality. The name of the class should be chosen with accordance to its functionality with the first capital letter, e.g. "NameOfTheClass". Each class is defined in the separated header file.

Listing C.1: Example of the class comment

```
/*!
 * \brief Custom widget representing domain.
 *
 * Implements custom widget inheriting from QGroupBox.
 * Each object of this class represents
 *              one domain and its position
 *              and amount of diffusive agents.
 */
class GroupBoxWidget: public QGroupBox
```

C.2.3 Methods

Like the class, each method contains a brief information about its purpose. Also if the method has any parameters, it is necessary to write a short note, about what each parameter represents. The name of the method is written with a small first letter, e.g. "nameOfTheMethod".

Listing C.2: Example of the method comment

```
/*!
 * \brief Creates domain with given coordinates.
 *
 * \param x X coordinate of the domain.
 * \param y Y coordinate of the domain.
 * \param gf Amount of growth factor.
 */
 void addDomain(int x, int y, std::string gf);
```

C.3 Structure of the program

The program is currently split into five classes (CellularAutomaton, DrawWidget, GroupBoxWidget, InfoLabel, MainWindow). Detail description about what each class implements can be found in the bachelor thesis.

C.4 Compilation & Deployment

C.4.1 Windows

On Windows, the best way is to install **Qt Creator**[6]. It is developed by *Qt Project* as well as the whole framework Qt. It includes debugger, syntax highlighting and autocompleting. The Qt Creator includes Qt Designer, which is an application for designing graphical user interface from Qt widgets. It has also many built-in libraries and contains a documentation for each class and its methods.

In the QtCreator, a user can open the project by choosing the *.pro file in the project folder. Then, the working space appears. The user can easily modify the program and then choose to compile the program in the release mode. After the compilation, a new directory named "release" is created in the project folder. Unfortunately, the directory, where the release version of the application is stored, does not contain all libraries that are necessary to run the application. However, Qt has a program which is able to add the libraries into the folder. The program is called "windeployqt" and is located in the folder "\Qt\5.5\mingw492_32\" bin (path can be slightly different according to the type and version of the compilator which the user is using). Windeployqt can be executed from the command line, as a parameter it takes the path to the application binary file.

Listing C.3: Command for creation of the release package

```
windeployqt <path-to-app-binary>
```

The windepoyqt creates a folder with an application file and libraries. It can happen that a few libraries are still missing, they have to be put into the folder manually.

Another way how to prepare the release package is to download the archive with the old original version of the application from the research gate website, unzip the archive and repleace the old executable *.exe file with the new one.

C.4.2 Linux

For the compilation of the applications source code and execution, it is neccessary to have installed Qt 5.5.1 or newer. Then the compilation is done by following commands:

```
qmake
make
./DomainGrowth
```

[6] http://www.qt.io/ide/

Appendix D

Contents of enclosed CD

```
readme.txt ...................... the file with CD contents description
exe ..................................... the directory with executables
src ..................................... the directory of source codes
    DomainGrowth ............................... implementation sources
    thesis .............. the directory of LaTeX source codes of the thesis
text ......................................... the thesis text directory
    BT_Martin_Muzak_2016.pdf ........... the thesis text in PDF format
```