# Generating two-dimensional game maps with use of cellular automata

Michał Wolski

December 19, 2018

# Table of contents


**1 Introduction** **3**
- 1.1 Objectives . . . 4
- 1.2 Thesis scope . . . 4
- 1.3 Thesis structure . . . 5
- 1.4 Experimental setup . . . 5

**2 Research on 2D map generation methods** **6**
- 2.1 Maps and cartography . . . 6
- 2.2 Maps in games . . . 8
- 2.3 Interactivity and automation . . . 11
- 2.4 Existing solutions for planar map generation . . . 12
  - 2.4.1 Cellular automata . . . 12
  - 2.4.2 Generative grammars . . . 14
  - 2.4.3 L-systems . . . 14
  - 2.4.4 ... . . . 14
  - 2.4.5 Other methods . . . 14
- 2.5 Choosing a method of generation . . . 14
  - 2.5.1 Effectiveness . . . 14
  - 2.5.2 Accessibility . . . 14
  - 2.5.3 Cost . . . 15
- 2.6 Chosen approach: cellular automata for 2D map generation . . . 15

**3 Generating and visualizing maps - proposed solution** **16**
- 3.1 Analysis of requirements for a map generator . . . 16
  - 3.1.1 Functional requirements . . . 16
  - 3.1.2 Non-functional requirements . . . 17
  - 3.1.3 Constraints . . . 18
- 3.2 Design . . . 18
  - 3.2.1 Data structures and persistence . . . 18

3.2.2 Application logic . . . . . 18
3.2.3 User interface . . . . . 18
3.3 Basic cellular automata simulations . . . . . 18
3.4 Generating maps with CA . . . . . 21
3.5 Implementation . . . . . 21
3.6 Tests . . . . . 21
3.6.1 Performance test . . . . . 21
3.7 Deployment (?) . . . . . 21

**4 Conclusions** **22**
4.1 Evaluation of results . . . . . 22
4.1.1 Effectiveness . . . . . 22
4.1.2 Accessibility . . . . . 22
4.1.3 Cost . . . . . 22
4.2 Perspectives for usage . . . . . 22
4.3 Further work . . . . . 22

**Bibliography** **23**

**List of figures** **25**

**List of tables** **25**

**List of abbreviations and acronyms** **26**

**Attachments** **27**

# Chapter 1

# Introduction

During recent years, presence of computer games in human lives has increased. The amount of time spent on playing games by the modern society has shown that games are desirable both as a means for entertainment and a medium of expression. However, as the interest in games rises [1] and computer games become increasingly complex, the need for game content must also rise. Elements such as believable maps, textures, sound and models (among other types of content) are a necessary resource for production of games.

Studies such as [Hen+13] show where the evidence for insufficiency of manual content creation may be found. In the study, authors point to work of Kelly and McCabe [KM07], Lefebvre and Neyret [LN03], Smelik et al. 2009 [Sme+09] and Iosup 2009 [Ios09] as sources which reveal game content production as a time-consuming and expensive endeavour. Hence, it is logical to conclude that information contained in studies and statistics on the topic of game development suggest that most projects aimed at creating games or simulations could benefit from seeking new or more efficient automated means of content creation.

### Solving the inefficiency issue

In order to provide a solution to the inefficiency of manual content production, formal methods have emerged and are commonly referred to as procedural generation techniques, defined by the literature as processes or methods of automatic content creation, through algorithmic mechanisms [Tog+11] [YT15].

Scientific surveys such as [Hen+13] and [Sme+09] show why investigating

[1] The Interactive Software Federation of Europe compiles and publishes statistics which include frequency of gaming in European countries and show that demand for games is on the rise. `https://www.isfe.eu/industry-facts/statistics`

procedural generation is useful for the game industry, by providing examples of successful methods which can be used to generate content for games. Primary concerns which drive the interest in automated ways to create game content are the rising project costs and increasing development time.

In order to reduce the cost of game development, allow for greater replay value or provide a feeling of vastness to the game worlds that designers aim to create, procedural content generation techniques can provide an attractive solution to the problem of content creation. Surveys such as [Hen+13], [Tog+11] and [DC+11] show what types of game content can be generated and are a good starting point for seeking methods of procedural generation.

## 1.1 Objectives

This thesis focuses on automated creation of 2-dimensional game maps using a cellular automata approach to generate small map tiles and merge them into a bigger map. Such approach allows for a degree of control to the map designer – who may want to decide which tiles will be merged and at which locations in the map they will be present. Integrating manual editing or parametrization of desired results with procedural generation techniques has been proposed before in the works concerning procedural generation techniques [Bid+10], [Sme+10], [Sme+11].

Beginning experimentation with flat maps on 2-dimensional plane avoids the complexity that may arise when dealing with higher dimensions, hence the main aim is to develop a solution to the problem of automating planar map creation for games.

Specifically, maps created by the generator should be planar and their layout must have characteristics of shapes that can be helpful in design of believable environments. These characteristics may include irregularity, asymmetry and imbalance. The goal is to create structures that are interesting because of their unpredictability, somewhat resembling the look of results produced by natural processes like erosion, dissolution, deformation, tectonic fragmentation or weathering.

## 1.2 Thesis scope

The research carried out to support this thesis was focused on discovery of generative techniques presented in current literature describing methods of content generation for games.

Work on preparing the solution has been divided into the following parts, described in the remainder of this thesis:

- research on procedural generation of planar maps and method selection
- design of a map generator program and its implementation in C++
- extending the feature set of the generator

TO DO: review scope after thesis body is done » scope well defined? scope matches content?3

## 1.3 Thesis structure

This thesis includes introduction followed by three chapters. Chapter 2 serves as a study on possible mechanisms that could be used for procedural generation and specifically, for creation of 2D maps for games. Chapter 3 describes design and implementation of a solution to the problem along with performed experiments. Chapter 4 summarizes the findings and concludes the thesis.

## 1.4 Experimental setup

The solution that allowed to carry out experiments in this thesis was implemented using the C++ programming language and compiled with MSVC++ 14.0 compiler, natively included in the VS2015 IDE. Other tools and libraries used in the project:

- Dear ImGui, by Omar Cornut - to easily build an Immediate Mode user interface. Project homepage: `https://github.com/ocornut/imgui`
- GLFW 3.2.1 library - to create an OpenGL context and have direct access to texture functions. Project homepage: `http://www.glfw.org/`
- TO DO: SFML not used

In order to satisfy requirements for using OpenGL API function calls, it is recommended to use a dedicated graphics processing unit and install updated drivers. For the purposes of development and experiments *nVidia GeForce GTX 560M* was used and drivers were updated to latest available versions.

This thesis has been prepared with LaTeX system for document typesetting, included diagrams were drawn with *UMLet* - an open source modelling program.

# Chapter 2

# Research on 2D map generation methods

## 2.1 Maps and cartography

Historically, maps have been used by the human race since ancient times. The need for navigation in the world has been a driving force behind the evolution of maps. Starting with cave paintings and representations of stars on the sky, our kind had the need for capturing an abstract model of a territory, terrain shape, location of useful resources or some other aspect of surrounding environment in a useful way.

Making a model of the physical (or fictional) world with maps requires choice of the data types describing locations represented on the map. List of data types visualized with maps has been growing with the evolution of cartography and whenever new technologies have been introduced to the map making crafts. Some examples of data possible to represent on maps are:

- physical maps - terrain shape, elevation, forests, bodies of water, etc.
- political maps - borders around a territory, districts, states
- climate and weather maps - temperature, humidity, precipitation, wind currents
- geologic maps - terrain features, location of precious resources underground
- star maps - views of the distant cosmic objects measured by solid angles from a fixed point

- route maps - transport links, connections joining points on the modelled territory

Although early maps had the form of drawings or etchings on surface of solid materials, now there are other possibilities of representing the abstract model which a map aims to represent. The rise of digital maps and geographic information systems has opened new possibilities - maps have become dynamic entities, stored digitally, easily updated with new data and not limited to the boundaries of physical model. With digital maps, it is possible to show more than one layer of data, as chosen by the user, whereas physical maps are limited to the data and view scale chosen initially at the time when map was crafted. Despite the limitations, they still can serve well as a medium for storage of geographic information, requiring simpler processes during archival and conservation efforts [Bag17].

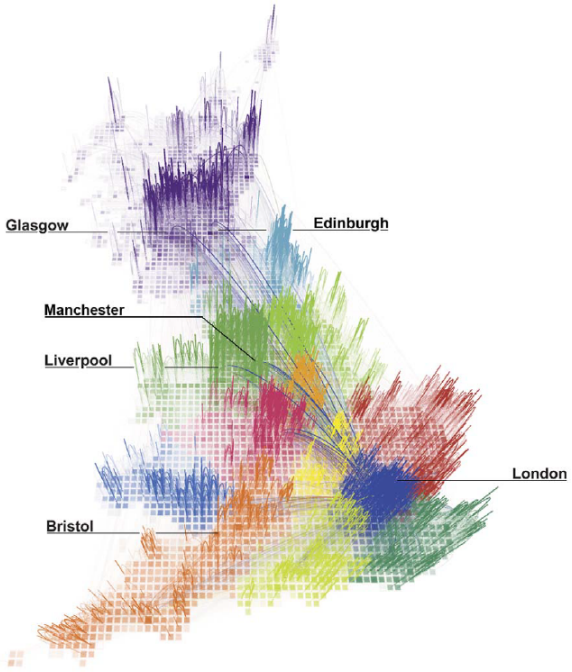



***Figure 2.1:*** *Pure data map showing the geography of talk in Great Britain. Authors measured the total talk time via communication networks between areas in Britain and used the data to produce the visualzation. Source article: [Rat+10]*

Digital technology has also brought interesting methods to the art of crafting maps, allowing for new types of maps to be crafted, which brought previously undiscovered insights into the nature of represented territories. For example, with

data maps such as the one described by article [Rat+10] it is possible to draw more useful borders around regions, fitting actual human interaction groups as opposed to those defined by past governments, as shown in figure 2.1. Maps like these are generated from vast data sets, collected and stored in databases - an approach that would not be possible without use of digital technology. Since maps can emerge out of pure data, the same approach can be used to create fictional maps, out of generated data.

The possibility of visualizing map layers which could not have been created and shown without digital data processing techniques and ease of experimentation with the information and algorithms used to create modern digital maps have also made it apparent that the source data for the layer itself do not necessarily have to measure some aspect of reality, but can be generated using mathematical methods. Such approach effectively allows for creation of fictional maps, representing imaginary territories. With that in mind, let us now move on to the sphere of fictional maps and their use in games, physical and digital alike.

## 2.2 Maps in games

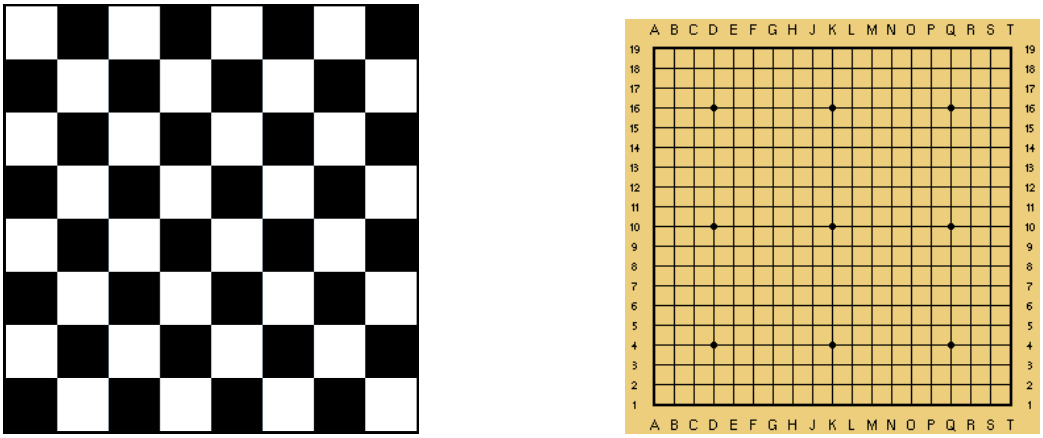
It is not clear what kind of game was the first one in history to use a map to represent the game world, however two notable examples may easily come into mind: Chess and Go, which are both widely known around the world.

Chess, a board tactical war-game developed before 6th century AD, uses black-white board as the map of its world. Although the environment represented by a chess board is very simple, it has some important features and rules. The map is composed of square cells, which are arranged on 8 by 8 grid, effectively creating a rigid boundary around the game world, which according to game rules - cannot be crossed. Each cell has 8 neighbours and can be occupied by only one game piece.

Another game, originating from ancient China, defines a similar, grid-based game world, effectively making a map of a uniform planar territory. Go is played on 19 by 19 board, however smaller board sizes are used as well. In Go, the goal is to capture more territory than the opponent, which is done by placing game pieces on line intersections, one piece per turn.

TO DO: expand on Go, comment on both, then introduce modern board games

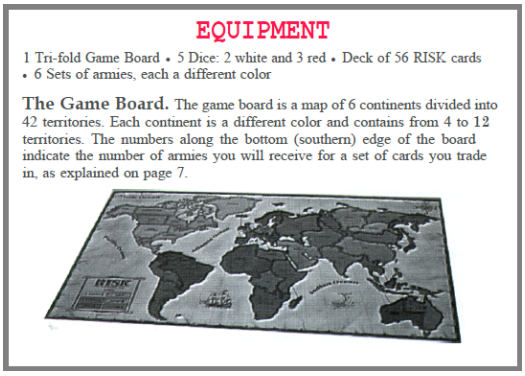
Another interesting example of a game world map is the multi-player strategy board game Risk, invented in 1957 by Albert Lamorisse, where the map represents a territory divided into regions, which must be captured by players in order to win. Risk game shows how a political world map with imagined region borders can be




***(a)*** *Chess board* ***(b)*** *Go board*

***Figure 2.2:*** *Chess and Go both use planar boards divided into tiles by a square grid - simple maps to represent the environment in which game is played.*

creatively used in a game, as a resource for the players to fight over. The game of Risk has been since published in many variations. Most of them share the same gameplay goal: to capture more territory than the opponents do, which is an example of how a game might use environments represented by maps as a limited resource for players to acquire.



EQUIPMENT

1 Tri-fold Game Board • 5 Dice: 2 white and 3 red • Deck of 56 RISK cards • 6 Sets of armies, each a different color

**The Game Board.** The game board is a map of 6 continents divided into 42 territories. Each continent is a different color and contains from 4 to 12 territories. The numbers along the bottom (southern) edge of the board indicate the number of armies you will receive for a set of cards you trade in, as explained on page 7.

***Figure 2.3:*** *Risk rule book fragment, containing a photo of the game board. The board shows a world map with fictional political borders, dividing the map into regions, which serve as a resource for players to capture. Image source: photo taken from original Risk rule book, copyright Hasbro 1993*

Modern board games have introduced many new ideas to the design of game boards. One example of such ideas is bringing modularity into the board design, composing pieces of the board similarly to how a jigsaw puzzle is composed of

singular pieces. Such arrangement allows for greater value in replaying the game, since the game world can be different at each time the game is played.

An example of such game is Carcassonne, published in 2000, designed by Klaus-Jürgen Wrede in Germany. The game of Carcassonne involves an interesting mechanism: rather than having a fixed game board, cards with tiles are used to construct the board during gameplay. The game rules around tile placement can be thought of as an algorithm of procedural generation: only one tile can be placed during each game turn, adjacent to other tiles, forming a connection with features that tiles represent - roads must connect to roads, fields to other fields, and cities to cities. The rules ensure that the players will develop the game board as the game progresses, which leads to an interesting observation: each turn, the players are presented with new territory to consider in their decisions - and since the game requires players to deploy their resources onto the constructed game map in order to accumulate score and eventually win, these decisions may often become quite challenging with increasing complexity of the board layout.

TO DO: carcassone web version at https://concarneau.herokuapp.com/game

***Figure 2.4:*** *Carcassonne game board during gameplay. Players place tiles adjacent to existing ones, making sure that each tile fits the others. Image source: https://deerfieldlibrary.org/2016/01/carcassonne-a-modern-board-game-for-adults-teens/*

TO DO: comment - board games: map creation mechanics, rules

However, some of the more complex game rules and ideas are better implemented using computer simulation, since most of the mundane tasks which do not contribute to gameplay can be automated. Random number generation, board preparation and arrangement, checking player moves against game rules - all those activities are good candidates for automation. The other reason to simulate games

may be to develop artificial intelligence algorithms which can simulate player behaviour at a chosen level of competitive play, effectively providing a way for beginners who want to learn the game they are interested in or for veterans who want to develop their skills further, as has been done for chess and other classic board games.

There are many more examples of modern board games which involve interesting mechanics, but their description lies beyond the scope of this thesis project. Moving on to the next section, let us investigate a few examples of computer games and simulations, where maps are used to construct some aspects of gameplay.

## 2.3 Interactivity and automation

TO DO: write about PCG: General PCG - types of content, methods. Maps as content type. Then: Methods suitable for map generation (next section).

The evolution of personal computers has allowed players to enjoy a new form of entertainment - video and computer games. Possibility of performing real-time simulations on computers and development of computer graphics rendering techniques have created a new medium of expression in the form of computer software. At the time when early forms of interactive simulations were created, first computer games were also developed.

In the past decades since the first computer games emerged, an industry focused on the craft of game development has emerged. The efforts of modern game designers and developers have lead to creation of multiple game genres and have driven the evolution of mechanics and challenges that modern games offer to players.

As stated in chapter 1, the context of this thesis does not deal with projections of 3D objects onto a plane, like the fields of geography and cartography do, as shown by works similar to [Sny93]. The goal is to generate planar maps, which makes games that include them of particular interest for finding examples of working solutions to the problem.

One of classic examples involving procedurally created maps is Nethack, developed in 1987.

expand

TO DO: examples from games: Diablo, NetHack (1987), Ultima Ratio Regum, Dwarf Fortress (2006),

TO DO: `http://www.roguebasin.com/index.php?title=Major_roguelikes`

TO DO: http://ascii-patrol.com/map.png + source?

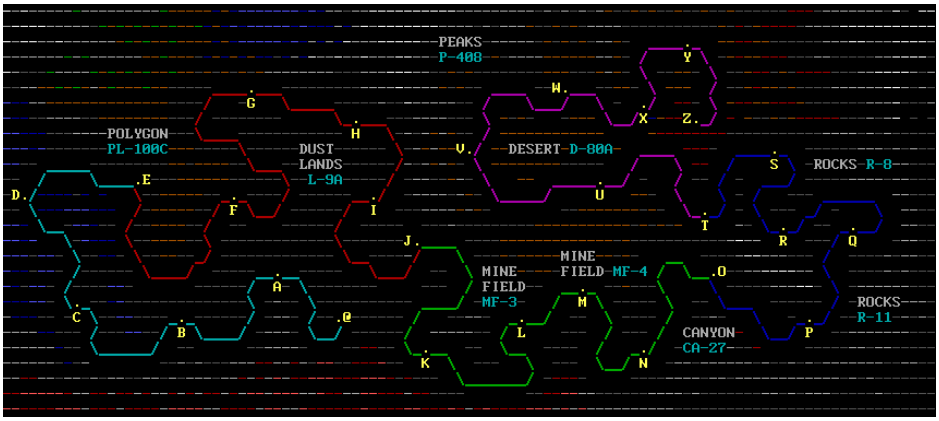



***Figure 2.5:*** *A game map using ASCII art*

## 2.4 Existing solutions for generating planar maps

In scientific surveys on PCG methods, we find approaches to map generation employed in the past. As listed by Hendrikx et al. [Hen+13],

TO DO: list map procgen methods

TO DO: HOW it was done until now? options?

TO DO: ref survey with table of 2d dungeon gen

### 2.4.1 Cellular automata

A cellular automaton is a simulation in which every object in a mathematically defined space is being updated at every step of a simulation. Historically, cellular automata and their properties have been studied since the time of first electronic computers [Sar00]. One of the most complete sources on cellular automata is a book summarizing research on CA carried out by Stephen Wolfram since 1980s [Wol02], where a classification of cellular automata is shown along with examples for each kind of CA.

Specifically, 2-dimensional automata operate on a grid of cells with arbitrary discrete dimensions. Each cell in the grid has neighbours, which may be relevant to the simulation rules. Depending on the type of rules which are used by a particular CA, a different type of cell neighbourhood may be used. To present this concept concisely, a short list of definitions follows.

**Cell** A cell is simply one unit positioned in CA simulation space. Cells have state, which can be simple - for example, a binary digit, an integer - or more

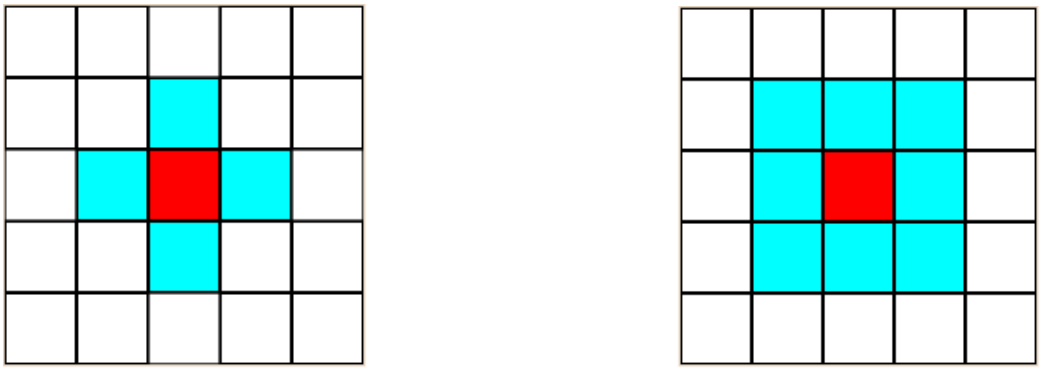
complicated - a real number with constraints, a complex number, or other.

**Cell neighborhood** In a context of a 2D square grid of cells, neighbourhood is a collection of nearest cells to the selected one.

**Moore's neighbourhood** Moore neighbourhood includes the cell and its immediate neighbours - one to the north, south, east and west of the cell, as shown in figure 2.6.

**Von Neumann neighbourhood** Von Neumann neighbourhood includes 8 closest neighbours of the cell - immediate and diagonal, as shown in figure 2.6.

**Other types of neighborhood** It is possible to imagine other types of cell neighbourhoods, possibly including more cell rings around a cell or only a selection of them arranged in a custom pattern. Those cases are beyond the scope of this thesis.



***(a)*** *Moore neighborhood* ***(b)*** *Von Neumann neighborhood*

***Figure 2.6:*** *Two basic types of cell neighborhood*

There are ...

Every CA simulation also consists of rules which drive the process of cell evolution to its next stage. Typically, such rules define how board elements must be changed once a specific element arrangement is recognized.

TO DO: ca basics - game of life

TO DO: using CA for simulations

TO DO: using CA for generation of content

### 2.4.2 Generative grammars

TO DO: Sportelli, Francesco & Toto, Giuseppe & Vessio, Gennaro. (2014). A Probabilistic Grammar for Procedural Content Generation. 10.13140/2.1.3820.4163.

### 2.4.3 L-systems

### 2.4.4 ...

### 2.4.5 Other methods

## 2.5 Choosing a method of generation

In order to effectively judge the value that a working map generator may bring to a game development project, we need to consider what characteristics should be evaluated. First, a useful generator must be effective at map generation.

TO DO: how to measure effectiveness? time of map generation, map shape, desirable map features?

Another point to consider is how easy to use such generator can be. Game designers may ultimately decide to use manual methods of map creation if the method of map generation requires too much effort to include in their project.

TO DO: how to measure such ease of use? accessibility?

The third aspect of choice what a generation method could be used is to consider how much value it brings to the designer versus what development costs it can reduce.

TO DO: how to measure cost?

The following subsections describe how each of the mentioned aspects can influence the choice of a generation method.

### 2.5.1 Effectiveness

TO DO: study on generation time

TO DO: desired characteristics of generated content?

### 2.5.2 Accessibility

TO DO: study on what makes generation easy to include in game development projects

TO DO: integrating manual editing AND procgen

### 2.5.3 Cost

TO DO: examples of development costs - human resources, machine resources

TO DO: which of these costs can be reduced by PCG

## 2.6 Chosen approach: cellular automata for 2D map generation

One of possible proposed approaches is the work of L. Johnson, G. Yannakakis and J. Togelius from IT University of Copenhagen [JYT10].

Authors describe rules of a cellular automaton which are able to transform a tile filled initially with random distribution of cells into a tile which has interesting properties for a map designer.

TO DO: authors describe a process - 1 random image 2 apply CA steps as in article cave gen 3 merge tiles, result: maps!

TO DO: short paragraph on the choice of CA for game maps

TO DO: why we chose CA for mapgen?

TO DO: what are pros and cons of such choice?

# Chapter 3

# Generating and visualizing maps - proposed solution

To describe the developed solution concisely, this chapter consists of five sections: definition of required features, design of simple CA simulation and map generator models, followed by implementing them in C++ programming language, experiments and finally, tests.

## 3.1 Analysis of requirements for a map generator

Having gathered the abstract constructs needed to build a CA map generator in chapter 2, we may proceed to state the requirements formally. In the following sections, we will:

- define features required of a map generator program to automate creating maps,
- state the desired properties of such generator,
- discover to what constraints such program may be required to conform.

### 3.1.1 Functional requirements

First, we must define the desired functions which a useful map generator must provide to its user. Since we have selected an approach to map generation based on cellular automata in chapter 2, we have to include simulation of CA states. As described in [JYT10], our approach consists of generating a random image, which after several transformations performed by CA rules becomes structured

with island-like features. Such image can then be used as a tile for larger maps. Showing each step of image transformation could be useful to the designer, allowing them to control what kind of generated tiles will be chosen to compose the map.

Another potentially useful feature would be manual editing of tiles before they are placed in the map, which would allow for an even greater degree of control over tile contents. Further expansion of such feature could be to provide the designer with tools to make their own rules for tile generation, while also allowing them to define types of single cells composing the tile.

Finally, a map generator without a mechanism for saving the work done by a designer would certainly not be a useful tool. Such program needs a way to export generated maps to a file format that later can be used by a game engine of choice, possibly with procedures written by other programmers. Although saving or exporting maps is not really necessary for experimentation and testing, a production version of the program should have such functionality.

To summarize desired functions of a map generator program, a list follows:

1. The program must implement a cellular automaton to generate map tiles
2. The program must show generation stages graphically
3. The program must allow building maps from components (tiles)
4. The program may allow manual editing of map tiles
5. The program should have a mechanism for exporting generated maps
6. TO DO: expand desired functions

### 3.1.2 Non-functional requirements

TO DO: nf-req intro

1. The program must have a graphical user interface.
2. The GUI must allow to change generation parameters, which should be grouped and ordered.
3. The program must be responsive to user input and avoid crashing.
4. TO DO:

### 3.1.3 Constraints

1. Map tile generation time must not exceed 5 seconds.
2. User interface elements must not invoke non-existent mechanisms.
3.

## 3.2 Design

### 3.2.1 Data structures and persistence

TO DO: how do we store data?

TO DO: diagrams of cell, board

TO DO: exporting data from generator?

TO DO: how designers can get a complete map model?

### 3.2.2 Application logic

TO DO: how a generator will work

TO DO: behavior diagrams

### 3.2.3 User interface

TO DO: OpenGL immediate mode paradigm

TO DO: imgui immediate mode user interface library

TO DO: diagram of texture class, used by Component MapGen, uses OpenGL

TO DO: mention Bret Victor talks - why we choose Immediate Mode

## 3.3 Basic cellular automata simulations

Having chosen cellular automata as a method for generating maps, we need to have a clear idea about how to approach building a program that could simulate a cellular automaton. One of the helpful resources on the topic of building cellular automata simulations is chapter 7 in Nature of Code, a book by Daniel Shiffman [Shi12], where we can find a short tutorial to build our first CA simulation. There,

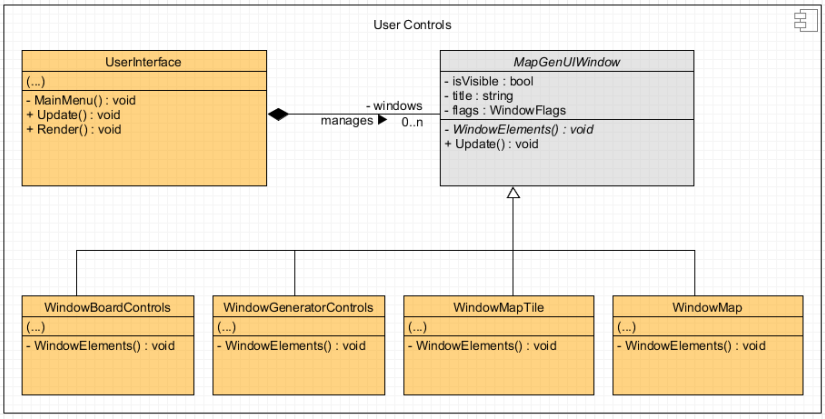



***Figure 3.1:*** *Example model of classes to be used to construct user interface in the map generator program*

author describes elementary concepts needed to construct a basic CA, explains how to implement a working simulation and provides helpful exercises. The tutorial is quite useful as a guide, since examples presented in New Kind of Science [Wol02] are implemented in the Wolfram language and would require familiarity with it. As stated in Nature of Code [Shi12], a 2-dimensional CA would need the following key elements to be simulated:

- Cell state - every cell has a state updated on each simulation step,
- Grid - a space on which cells are placed,
- Neighbourhood - each cell needs to know the state of its neighbours to update its state.

In order to represent the cells of an automaton, a primitive data type is sufficient. However, we could design a class which will act as a collection of cells and provide additional utility to its user. Figure 3.2 presents an example model of a class that would encapsulate a collection of cell states while also preserving information about the board on which those cells are placed.

We can also assign a number to each cell

TO DO: why?

as shown in table 3.1.

Such abstraction creates an easy to use interface for further development and is also sufficient to access the values of neighbors to the selected cell. However, in some CA simulations summing the values of cells in neighbourhood is a common

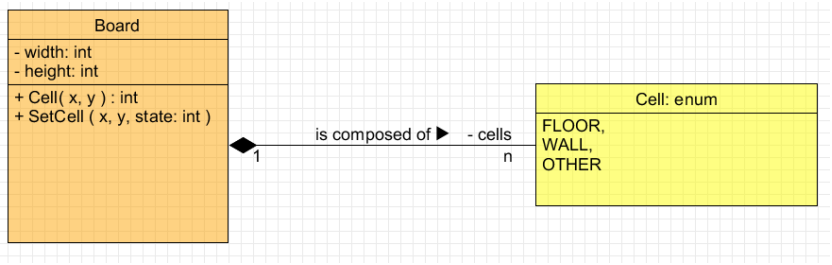



***Figure 3.2:*** *A possible model of a Board Class which holds cell states in its block of memory and lets its user change their states*

| | | |
|---|---|---|
| 0 | 1 | 2 |
| 7 | S | 3 |
| 6 | 5 | 4 |

***Table 3.1:*** *Cell neighbours, numbered. S denotes selected cell. Cells marked with odd numbers are members of Moore neighborhood of selected cell and all numbered cells are members of Von Neumann neighbourhood of it.*

operation, so we can include variations of it for convenience. Similarly, a method to translate cell states into texture points would be welcome, since we may possibly need a way to display the state of CA board on screen. Adding those elements to our abstraction yields a class presented on figure 3.3.

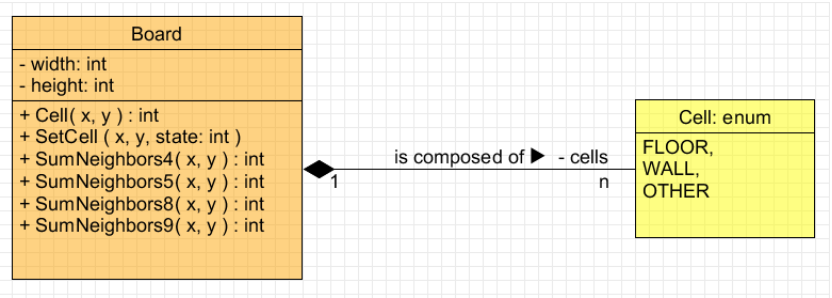



***Figure 3.3:*** *Revised Board abstraction - added methods for neighbor sums and translation of cell states to texels*

TO DO: add neighbor methods to board2

TO DO: result of what it all does?

At this point, we could also observe a common property of cellular automata - whenever cell states need to change (the simulation moves to a later step), the state change is applied to every cell in the grid before simulation step ends [Wol84]

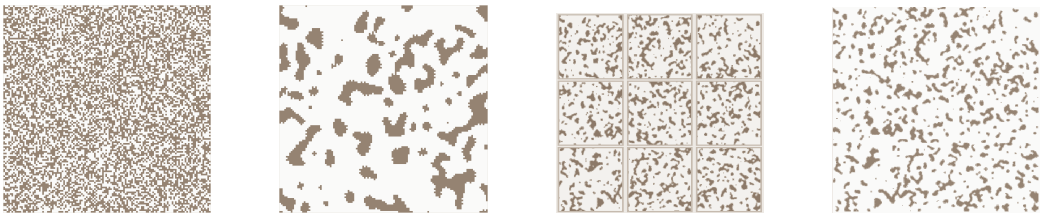
and cells do not need to be updated in sequence if only all cells will be changed before the next step. Hence, cell state updates could be applied in parallel to reduce the time needed to compute the simulation step. One way to do so would be to apply the findings presented by Reno Fourie in his thesis about applying CUDA technology to reduce time to compute next state of the board in case of 2-dimensional cellular automata [Fou15].

TO DO: what else to include?

TO DO: more on CA, 2dim CA?

## 3.4 Generating maps with CA

Since the goal of this work is not to just implement a working cellular automaton simulation, we need to find a way to generate maps using CA simulation.



***(a)*** *Step 1 - random noise*

***(b)*** *Step 2 - generated tile*

***(c)*** *Step 3 - tiles in a grid*

***(d)*** *Step 4 - complete map*

***Figure 3.4:*** *Four stages of map construction*

## 3.5 Implementation

TO DO: describe how it all works now, with object diagrams

TO DO: refer to code itself

## 3.6 Tests

### 3.6.1 Performance test

## 3.7 Deployment (?)

# Chapter 4

# Conclusions

## 4.1 Evaluation of results

### 4.1.1 Effectiveness

### 4.1.2 Accessibility

### 4.1.3 Cost

## 4.2 Perspectives for usage

TO DO: map generator will be used in game project codenamed 'UW'

## 4.3 Further work

TO DO: future: expanding the generator

TO DO: future: adding more rules

TO DO: future: user defined cell types

TO DO: future: non-discrete cell states or more complex cell types

TO DO: future: 3d maps?

# Bibliography


[Bag17] L. Bagrow. *History of Cartography*. Taylor & Francis, 2017. ISBN: 9781351515580.

[Bid+10] Rafael Bidarra et al. "Integrating semantics and procedural generation: key enabling factors for declarative modeling of virtual worlds". In: *Proceedings of the FOCUS K3D Conference on Semantic 3D Media and Content, Sophia Antipolis-Méditerranée, France*. 2010.

[DC+11] Daniel Michelon De Carli et al. "A survey of procedural content generation techniques suitable to game development". In: *Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on*. IEEE. 2011, pp. 26–35.

[Fou15] Ryno Fourie. "A parallel cellular automaton simulation framework using CUDA". PhD thesis. Stellenbosch: Stellenbosch University, 2015.

[Hen+13] Mark Hendrikx et al. "Procedural content generation for games: A survey". In: *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 9.1 (2013), p. 1.

[Ios09] Alexandru Iosup. "Poggi: Puzzle-based online games on grid infrastructures". In: *European Conference on Parallel Processing*. Springer. 2009, pp. 390–403.

[JYT10] Lawrence Johnson, Georgios N Yannakakis, and Julian Togelius. "Cellular automata for real-time generation of infinite cave levels". In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM. 2010, p. 10.

[KM07] George Kelly and Hugh McCabe. "Citygen: An interactive system for procedural city generation". In: *Fifth International Conference on Game Design and Technology*. 2007, pp. 8–16.

[LN03] Sylvain Lefebvre and Fabrice Neyret. “Pattern based procedural textures”. In: *Proceedings of the 2003 symposium on Interactive 3D graphics*. ACM. 2003, pp. 203–212.

[Rat+10] Carlo Ratti et al. “Redrawing the Map of Great Britain from a Network of Human Interactions”. In: *PLOS ONE* 5.12 (Dec. 2010), pp. 1–6. DOI: 10.1371/journal.pone.0014248. URL: https://doi.org/10.1371/journal.pone.0014248.

[Sar00] Palash Sarkar. “A Brief History of Cellular Automata”. In: *ACM Comput. Surv.* 32.1 (Mar. 2000), pp. 80–107. ISSN: 0360-0300. DOI: 10.1145/349194.349202. URL: http://doi.acm.org/10.1145/349194.349202.

[Shi12] Daniel Shiffman. *The Nature of Code: Simulating Natural Systems with Processing*. Daniel Shiffman, 2012.

[Sme+09] Ruben M Smelik et al. “A survey of procedural methods for terrain modelling”. In: *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*. 2009, pp. 25–34.

[Sme+10] Ruben Smelik et al. “Integrating procedural generation and manual editing of virtual worlds”. In: *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. ACM. 2010, p. 2.

[Sme+11] Ruben Michaël Smelik et al. “A declarative approach to procedural modeling of virtual worlds”. In: *Computers & Graphics* 35.2 (2011), pp. 352–363.

[Sny93] J.P. Snyder. *Flattening the Earth: Two Thousand Years of Map Projections*. University of Chicago Press, 1993. ISBN: 9780226767468. URL: https://books.google.pl/books?id=MuyjQgAACAAJ.

[Tog+11] Julian Togelius et al. “Search-based procedural content generation: A taxonomy and survey”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011), pp. 172–186.

[Wol02] Stephen Wolfram. *A new kind of science*. Vol. 5. Wolfram media Champaign, 2002.

[Wol84] Stephen Wolfram. “Cellular automata as models of complexity”. In: *Nature* 311.5985 (1984), p. 419.

[YT15] Georgios N Yannakakis and Julian Togelius. “Experience-driven procedural content generation”. In: *Affective Computing and Intelligent Interaction (ACII), 2015 International Conference on*. IEEE. 2015, pp. 519–525.

# List of Figures


2.1 Pure data map showing the geography of talk in Great Britain. Authors measured the total talk time via communication networks between areas in Britain and used the data to produce the visualzation. Source article: [Rat+10] . . . . . 7
2.2 Chess and Go both use planar boards divided into tiles by a square grid - simple maps to represent the environment in which game is played. . . . . 9
2.3 Risk rule book fragment, containing a photo of the game board. The board shows a world map with fictional political borders, dividing the map into regions, which serve as a resource for players to capture. Image source: photo taken from original Risk rule book, copyright Hasbro 1993 . . . . . 9
2.4 Carcassonne game board during gameplay. Players place tiles adjacent to existing ones, making sure that each tile fits the others. Image source: https://deerfieldlibrary.org/2016/01/carcassonne-a-modern-board-game-for-adults-teens/ . . . . . 10
2.5 A game map using ASCII art . . . . . 12
2.6 Two basic types of cell neighborhood . . . . . 13

3.1 User Interface model . . . . . 19
3.2 A possible model of a Board Class which holds cell states in its block of memory and lets its user change their states . . . . . 20
3.3 Revised Board abstraction - added methods for neighbor sums and translation of cell states to texels . . . . . 20
3.4 Four stages of map construction . . . . . 21

# List of Tables

3.1 Cell neighbours, numbered. *S* denotes selected cell. Cells marked with odd numbers are members of Moore neighborhood of selected cell and all numbered cells are members of Von Neumann neighbourhood of it. . . . . 20

# List of abbreviations and acronyms

The following terms, abbreviations and acronyms have been used in the thesis.

**CA** Cellular Automaton. A simulation consisting of cell objects.

**PCG** Procedural Content Generation. An automated process of creation.

**GUI** Graphical User Interface

TO DO: (?)

# Attachments

1. TO DO: include thesis defence documents
2. TO DO: ?
3. TO DO: ?
4. TO DO: ?

# Todo list

TO DO: review scope after thesis body is done » scope well defined? scope matches content?3 . . . . . 5
TO DO: SFML not used . . . . . 5
TO DO: expand on Go, comment on both, then introduce modern board games . . . . . 8
TO DO: carcassone web version at https://concarneau.herokuapp.com/game 10
TO DO: comment - board games: map creation mechanics, rules . . . . . 10
TO DO: write about PCG: General PCG - types of content, methods. Maps as content type. Then: Methods suitable for map generation (next section). . . . . 11
expand . . . . . 11
TO DO: examples from games: Diablo, NetHack (1987), Ultima Ratio Regum, Dwarf Fortress (2006), . . . . . 11
TO DO: `http://www.roguebasin.com/index.php?title=Major_roguelikes` 11
TO DO: http://ascii-patrol.com/map.png + source? . . . . . 11
TO DO: list map procgen methods . . . . . 12
TO DO: HOW it was done until now? options? . . . . . 12
TO DO: ref survey with table of 2d dungeon gen . . . . . 12
TO DO: ca basics - game of life . . . . . 13
TO DO: using CA for simulations . . . . . 13
TO DO: using CA for generation of content . . . . . 13
TO DO: Sportelli, Francesco & Toto, Giuseppe & Vessio, Gennaro. (2014). A Probabilistic Grammar for Procedural Content Generation. 10.13140/2.1.3820.4163. . . . . 14
TO DO: how to measure effectiveness? time of map generation, map shape, desirable map features? . . . . . 14
TO DO: how to measure such ease of use? accessibility? . . . . . 14
TO DO: how to measure cost? . . . . . 14
TO DO: study on generation time . . . . . 14
TO DO: desired characteristics of generated content? . . . . . 14

TO DO: study on what makes generation easy to include in game development projects . . . 14
TO DO: integrating manual editing AND procgen . . . 14
TO DO: examples of development costs - human resources, machine resources 15
TO DO: which of these costs can be reduced by PCG . . . 15
TO DO: authors describe a process - 1 random image 2 apply CA steps as in article cave gen 3 merge tiles, result: maps! . . . 15
TO DO: short paragraph on the choice of CA for game maps . . . 15
TO DO: why we chose CA for mapgen? . . . 15
TO DO: what are pros and cons of such choice? . . . 15
TO DO: expand desired functions . . . 17
TO DO: nf-req intro . . . 17
TO DO: . . . 17
TO DO: how do we store data? . . . 18
TO DO: diagrams of cell, board . . . 18
TO DO: exporting data from generator? . . . 18
TO DO: how designers can get a complete map model? . . . 18
TO DO: how a generator will work . . . 18
TO DO: behavior diagrams . . . 18
TO DO: OpenGL immediate mode paradigm . . . 18
TO DO: imgui immediate mode user interface library . . . 18
TO DO: diagram of texture class, used by Component MapGen, uses OpenGL 18
TO DO: mention Bret Victor talks - why we choose Immediate Mode . . . 18
TO DO: why? . . . 19
TO DO: add neighbor methods to board2 . . . 20
TO DO: result of what it all does? . . . 20
TO DO: what else to include? . . . 21
TO DO: more on CA, 2dim CA? . . . 21
TO DO: describe how it all works now, with object diagrams . . . 21
TO DO: refer to code itself . . . 21
TO DO: map generator will be used in game project codenamed 'UW' . . . 22
TO DO: future: expanding the generator . . . 22
TO DO: future: adding more rules . . . 22
TO DO: future: user defined cell types . . . 22
TO DO: future: non-discrete cell states or more complex cell types . . . 22
TO DO: future: 3d maps? . . . 22
TO DO: (?) . . . 27
TO DO: include thesis defence documents . . . 28
TO DO: ? . . . 28

TO DO: ? . . . . . 28
TO DO: ? . . . . . 28