Generating two-dimensional game maps with use of cellular automata

Michał Wolski

June 12, 2018

Contents

Chapter 1

Introduction

During recent years, presence of computer games in human lives has increased. The demand for games has shown that playing games, both as a medium of expression and a means for entertainment, is a desirable form of activity. However, as the demand for games rises [1] and the games become increasingly complex, demand for game content must also rise – game 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 endeavor.

Solving the inefficiency issue

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

[1] The Interactive Software Federation of Europe compiles and publishes statistics which include frequency of gaming in European countries: `https://www.isfe.eu/industry-facts/statistics`

show what types of game content can be generated and are a good starting point for seeking methods of procedural generation.

Personal motivation

During two recent years, the author of this thesis took part in a small, after-hours independent game development project. Working with a group of friends, using Unreal Engine as a tool to develop a simple prototype of a game belonging to the *rogue-like* game genre. The project is still in development phase and finding a good method of map generation can potentially result in contribution of useful features.

1.1 Thesis structure

The overall structure of this thesis includes introduction followed by three chapters. The second chapter serves as a study on possible mechanisms that could be used for procedural generation and specifically, for creation of 2D maps for games. The third chapter describes the experiments, design and implementation of a solution to the problem. Last chapter summarizes the findings and concludes the thesis.

TODO: LATER: check if structure description matches content

1.2 Objectives

This work focuses on automated creation of 2-dimensional game maps using a cellular automata approach. We aim to do so by generating small map tiles, which can be later merged 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. Moreover, we could also allow for editing the tile before placing it in the map. An approach that integrates manual editing or parametrization of desired results with procedural generation techniques has been proposed before [Bid+10], [Sme+10], [Sme+11].

We focus on creation of maps for games, since literature shows map generation as an interesting area for experimentation, although personal motivation influenced the choice as well.

Beginning experimentation with flat maps on 2-dimensional plane avoids the complexity that may arise when dealing with higher dimensions.

We will investigate existing methods for procedural generation of game maps which resemble cave structures. Then, an approach that may be used for automated creation of such maps will be selected and examined with a focus on implementing a working map generator. Main points of focus for this project are as follows:

- research on procedural generation of maps
- selecting a promising approach to use
- designing a map generator program
- implementing the solution in a programming language of choice

TODO: is that all?

1.3 Thesis scope

TODO: scope - what we will do, what we dont do
TODO: short: what could be done instead

1.4 Technology and tools

The following paragraphs summarize what tools were involved during the project of thesis preparation and performing the experiments.

1.4.1 Hardware

All experiments in this thesis have been performed using a laptop with an Intel x64 2.0 GHz multi-core processor, 16GB RAM and an *nVidia GeForce GTX 560M* graphics card.

1.4.2 Software

Development environment for the purposes of thesis experiments and writing has been set up under Windows 10 operating system with the following software installed:

- Visual Studio 2015 Community IDE

- CMake for Windows
- TeXstudio editor with MikTeX backend
- Git version control system
- Notepad++
- Modelio 3.7 open source modeling environment
- TODO: ...

Other configuration details include:

TODO: environment variables, configuration specifics...

This thesis has been prepared with LaTeX system for document typesetting.

Programming languages

The program 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.

Libraries

The implementation uses following libraries:

- 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/`
- TODO: ...

1.4.3 Other tools

Design patterns

TODO: list used design patters, if any. Singleton? Command? Factory?

1.5 Related work (?)

TODO: think what could be included here

Chapter 2

Research on 2D map generation methods

2.1 Definitions

Before we start planning a solution to the problem of map generation, we must first define what we mean by maps. As stated in chapter 1, our context does not deal with projections of 3D objects onto a plane, like the fields of geography and cartography do [Sny93]. Our goal is simply to generate planar maps.

Map TODO: what is a map?

TODO: 2d map types?
TODO: what do we mean by generation?

2.2 Automation - reduction in development time and cost

TODO: write about PCG in general, short
TODO: PCG types of content
TODO: PCG methods
TODO: focus on maps

2.3 Existing solutions for 2D map generation

TODO: HOW it was done until now? options?
TODO: ref survey with table of 2d dungeon gen

2.3.1 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.
TODO: introduce new definitions

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

Moore's neighborhood Moore neighborhood includes the cell and its immediate neighbors - one to the north, south, east and west of the cell.

Von Neumann neighborhood Von Neumann neighborhood includes 8 closest neighbors of the cell - immediate and diagonal.

TODO: write on use of CA for generation of content and then specifically maps

2.3.2 Generative grammars

TODO: what is a formal grammar?
TODO: how can it be used for generating maps?
TODO: why we cant use them?

2.3.3 L-systems

TODO: what are lsystems?
TODO: can they generate maps?
TODO:

2.3.4 ...

2.3.5 ...

2.4 Evaluation of existing methods for project purposes

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.

TODO: 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.

TODO: 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.

TODO: how to measure cost?

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

2.4.1 Effectiveness

TODO: study on generation time

TODO: desired characteristics of generated content?

2.4.2 Accessibility

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

TODO: integrating manual editing AND procgen

2.4.3 Cost

TODO: examples of development costs - human resources, machine resources

TODO: which of these costs can be reduced by PCG

2.5 Chosen approach: cellular automata for 2D map generation

TODO: short paragraph on the choice of CA for game maps
TODO: why we chose CA for mapgen?
TODO: what are pros and cons of such choice?

Chapter 3

Generating and visualizing maps - proposed solution

3.1 Map generator - analysis of requirements

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

3.1.1 Functional requirements

- user interface allowing playing with parameters
- rendering each generation step
- exporting generated maps
- ...

3.1.2 User requirements

- allow changing parameters by user
- format of exported maps must be easy to understand and use
- ...

3.1.3 Constraints

3.2 Design

3.2.1 Data structures, persistence

TODO: how do we store data?
TODO: exporting data from generator?
TODO: how designers can get a complete map model?

3.2.2 Application logic

TODO: how a generator will work
TODO: diagrams

3.2.3 User interface

TODO: mention Bret Victor talks
TODO: imgui immediate mode user interface

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, 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,
- Neighborhood - each cell needs to know the state of its neighbors 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 the user. Figure 3.1 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.

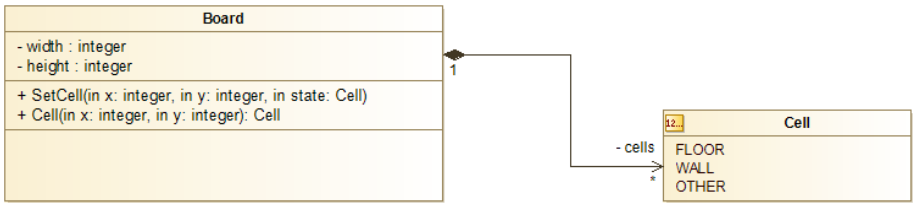

Figure 3.1: A possible model of a Board Class which holds cell states in its block of memory and lets its user change their states

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 neighborhood is a common 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 3.2.

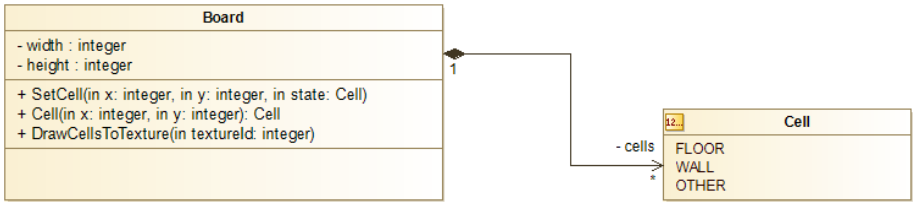

Figure 3.2: Revised Board abstraction - added methods for neighbor sums and translation of cell states to texels

TODO: what else to include?

TODO: more on CA, 2dim CA?

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 needed to simulate cellular automata [Fou15].

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. 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 tranform a tile filled initially with random distribution of cells into a tile which has interesting properties for a map designer.

TODO: 1 random image

TODO: 2 apply CA steps as in article cavegen

TODO: 3 voila, maps!

3.5 Implementation

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

4.3 Further work

Chapter 5

Full implementation source code

5.1 main.cpp

```
#include <iostream>

#include <GL\gl3w.h>
#include <GLFW\glfw3.h>

#include "UserInterface_MapGenerator.h"

GLFWwindow* window;

void glfw_error_callback (int error, const char* description)
{
  std::cerr << "GLFW Error " << error << ": " << description << std::endl;
}

bool glfwSetupWindow (unsigned int width, unsigned int height, const char* title)
{
  glfwSetErrorCallback (glfw_error_callback);
  if ( glfwInit () )
  {
    glfwWindowHint (GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint (GLFW_CONTEXT_VERSION_MINOR, 2);
    glfwWindowHint (GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    glfwWindowHint (GLFW_MAXIMIZED, GLFW_TRUE);
#if __APPLE__
    glfwWindowHint (GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif
    window = glfwCreateWindow (width, height, title, NULL, NULL);
    glfwMakeContextCurrent (window);
```

```
29    glfwSwapInterval (1); // Enable vsync
30    gl3wInit ();
31    return true;
32   }
33   return false;
34 }
35
36 int main (int, char**)
37 {
38   if ( !glfwSetupWindow (800, 600, "Cellular Automata Map Generator 152017") ) return 1;
39   else
40   {
41     UserInterface_MapGenerator missionControls = UserInterface_MapGenerator (window);
42     while ( !glfwWindowShouldClose (window) ) // Main loop
43     {
44       glfwPollEvents ();
45       {
46         missionControls.Update ();
47         missionControls.Render ();
48       }
49       glfwSwapBuffers (window);
50     }
51   }
52   glfwDestroyWindow (window);
53   glfwTerminate ();
54   return 0;
55 }
```

Listing 5.1: main.cpp Source Code

Bibliography

[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.

[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.

List of Figures

List of Tables

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.

PCGG Procedural Content Generation for Games.

TODO: (?)

Attachments

1. Załącznik 1
2. Załącznik 2
3. Załącznik 3