TU WIEN Informatics

Tour of Life

Eine Game of Life Learning Experience

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Patrik Szabó
Matrikelnummer 11811341

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Univ.Ass. Dipl.-Ing. Daniel Pahr, BSc

Wien, 2. Februar 2022

	Patrik Szabó	Eduard Gröller

Informatics

Tour of Life

A Game of Life Learning Experience

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Patrik Szabó
Registration Number 11811341

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Univ.Ass. Dipl.-Ing. Daniel Pahr, BSc

Vienna, 2nd February, 2022

Patrik Szabó Eduard Gröller

Erklärung zur Verfassung der Arbeit

Patrik Szabó

Wien, 2. Februar 2022

Patrik Szabó

Kurzfassung

Obwohl das Konzept hinter dem zellulären Automaten Game of Life in der Vergangenheit eingehend erforscht wurde, ist es immer noch schwierig, geeignete Ressourcen zu diesem Thema zu finden, die es Personen mit wenig oder keiner Erfahrung im Bereich der IT ermöglichen, es richtig zu verstehen. Es gibt keine einzige Lösung, die den Algorithmus hinter dem Automaten erklärt, den Benutzer mit der Simulation interagieren lässt und Erklärungen zu den gängigsten Bausteinen von Game of Life an einem Ort bereitstellt. Das Ziel dieser Bachelorarbeit ist es, ein Online-Lernwerkzeug zu entwickeln, das Menschen mit geringen Computerkenntnissen ermöglicht, sich mit dem Game of Life Automaten zu beschäftigen. Dies wird erreicht durch eine Reihe interaktiver Lektionen, die zum Game of Life angeboten werden, mit einem hohen Maß an Anpassbarkeit der Simulation selbst. Ein weiteres Ziel dieses Projekts ist es, eine 3D-Interpretation des Game of Life Automaten einzubeziehen, den Algorithmus entsprechend anzupassen und den Benutzern zu ermöglichen, mit der zusätzlichen dritten Dimension zu experimentieren.

Da das Game of Life kein neues Thema ist, wurden verwandte Arbeiten recherchiert und verglichen, und die Ergebnisse sind aufgeführt. Neben der entwickelten Webanwendung geht diese Arbeit tiefer in die Erforschung der Konzepte einer Turing-Maschine und der Touring-Vollständigen Systeme, wie dem Game of Life selbst, ein. Konkret werden Umsetzungsdetails vorgestellt sowie Ideen für die zukünftige Entwicklung dieser sogenannten Tour of Life und ihrer möglichen Merkmale.
Die Resultate dieser Arbeit sind öffentlich verfügbar, der Link zu der Tour of Life Webseite befindet sich am Ende dieser Arbeit.

Abstract

Although the concept behind the Game of Life cellular automaton has been explored deeply in the past, it is still difficult to find proper resources on this topic that would allow those with little to no experience in the field of IT to properly understand it. No singular solution exists that explains the algorithm behind the automaton, lets the user interact with the simulation, and provides explanations regarding the most common building blocks of the Game of Life, all in one place. The goal of this bachelor thesis is to create an online learning tool, that enables people with little computer knowledge to engage with the automaton. This is done by offering a range of interactive lessons on the Game of Life, together with a high degree of customizability of the simulation itself. An additional goal of this project is to include a 3D interpretation of the Game of Life cellular automaton, adjusting the algorithm accordingly, and allowing users to experiment with the additional third dimension.

As the Game of Life is not a resent topic, related work has been researched and compared, and the results are laid out. In addition to the developed web application, this thesis goes deeper in exploring the concepts of a Turing machine and Turing complete systems, such as the Game of Life itself. Implementation details are presented, as well as ideas for the future development of this so-called Tour of Life, and its potential features.
The results of this work are publicly available, the link to the Tour of Life website can be found at the end of this thesis

Contents

CHAPTER 1

Introduction

The first chapter of this thesis outlines the goals this project is trying to accomplish, as well as the reasons for doing so. It also sets up some ground rules that the finished project will have to follow, in order to be called a success once it is complete. These will be looked back at, at a later point in the thesis.
The concept behind the Game of Life will be more deeply explored in Chapter 3 of this thesis. In the meantime, a brief summary in order to understand the following segment is provided. The Game of Life is a cellular automaton, meaning a collection of cells on a mostly 2D grid, which evolve and change their state based on a set of predefined rules and the states of cells around them. A cell inside the game of Life can have only 2 possible states that it switches between: dead or alive. Certain configurations of living cells, which have been documented and tend to come up in the simulation are called patterns. The Game of Life needs an initial cell configuration as an input. This starting configuration then evolves based on the rules defined for the automaton.

1.1 Motivation

Interactivity in learning has been proven to help with information retention, especially when dealing with complex topics. One such seemingly very complex topic is the automaton known as the Game of Life, for which very few learning resources employing interactivity techniques exist online. A book called "The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge" by William Poundstone [37] covers the simplistic complexity of the automaton, as well as its history in great detail. This book was the main initial inspiration for this thesis, and it will be a major resource going forward. When the research phase on the subject of cellular automata and the Game of Life began, it turned out, that although the information exists online, it is often spread out among multiple sources, which present it with varying complexity.

Recent research referenced in Chapter 2 outlines the importance of interactivity in teaching/learning, and a website on the Game of Life automaton could not be found, that would offer it. This was an obvious gap, and addressing it provides a way to make the learning experience for anyone interested in researching the automaton in the future that much more pleasant.

1.2 Problem Statement

Research of existing solutions and learning resources regarding this topic shows a lack of opportunities for complete beginners, to properly get introduced to the concept of cellular automata, Turing machines, and the Game of Life. Most currently available tools are mainly a combination of simulation windows, where users can play the Game of Life, which however offer very little in the way of customization, or wiki-style information repositories and text-based explanations. Detailed examples for these will be provided in Chapter 2.
Very rarely has a web resource enabled the user to interact with the presented patterns or ideas, and if it did, these were isolated experiences, lacking any kind of logical progression, where users would be able to progress from the very basics to the more advanced concepts.

This is the gap this project fills, and in order to do so, the following criteria have been defined, that a project would have to fulfill:

- **Approachable to laypeople:** The concept of a cellular automaton, more specifically the Game of Life, is, on its own, not a very difficult one. Some online resources however describe it in a very confusing way, using terminology, not known to the general population. The lessons within the learning experience should start out simple, and increase their complexity and difficulty, as the user moves from lesson to lesson.
- **Interactivity:** It is proven, that higher engagement of students in the lesson leads to higher information retention rates [38]. By letting the user interact with the lessons, it is more likely that users will remember them, and the more complicated lessons will not feel overwhelming.
- **Proper documentation:** It is important that every major functionality, especially in the learning section, is properly explained. Only when the user engages with the full suite of the available features the website will offer can they have the best learning experience. An option should be provided at every step for the user to look into various topics by linking to external resources.
- **Self-containment:** The interactive platform will be developed as a web application, so it can be much more easily accessed by a large number of people. This eliminates the need to install potentially untrustworthy software on users' computers, and also makes it accessible from anywhere.

- **Customizability:** In order to make the platform accessible to many people, the learning tool should include a wide range of customization options. Especially the color adjustments within the simulation are important, as many existing Game of Life simulators alienate groups of people (e.g. with color-blindness) with their choice of color palette.
- **Proper visualization:** Proper visualization is important in presentation. This is especially true when working in three dimensions. It is crucial to find a coherent, natural way to visualize and interact with the 3D simulation of the Game of Life since existing solutions have arguably failed to do so.

A more detailed description of the criteria, especially how they have been implemented, will follow in Chapter 3.

CHAPTER 2

Research & Related Work

Since the Game of Life is such an old and deeply explored topic, one would not be hard-pressed to find countless simulators, papers, and blogs describing every little detail about the automaton. The problem with this statement is, that none of the information can be found, concisely and neatly in one place, wrapped inside a uniform UI with a focus on teaching the Game of Life in a playful, interactive, and highly customized way.

2.1 Related Work

The first result returned on Google if searching for the term "Game of Life" at the time of writing this thesis is a website by Edwin Martin, called "playgameoflife.com" [35]. This website was also used as a reference during the development of the web application, to check the proper functionality of the algorithm. It is a very simple rendition of the Game of Life simulator, with a limited way to interact with the platform. The platform is used as the basis for what will be this projects own learning experience, as when using the "playgameoflife" website, the lack of features is obvious, as can be observed in Figure 2.1. In addition to some very basic playback controls, a lexicon is also included, describing a lot of the discovered patterns. Some even come with a short description. These descriptions, however, mostly cover when and who discovered the displayed patterns, and not why they are interesting, or how they work.

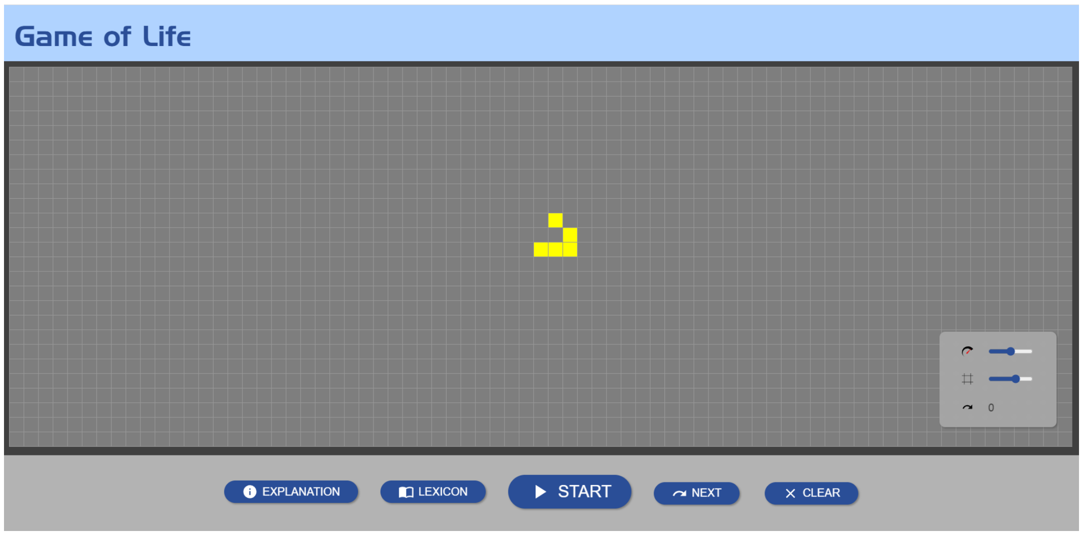
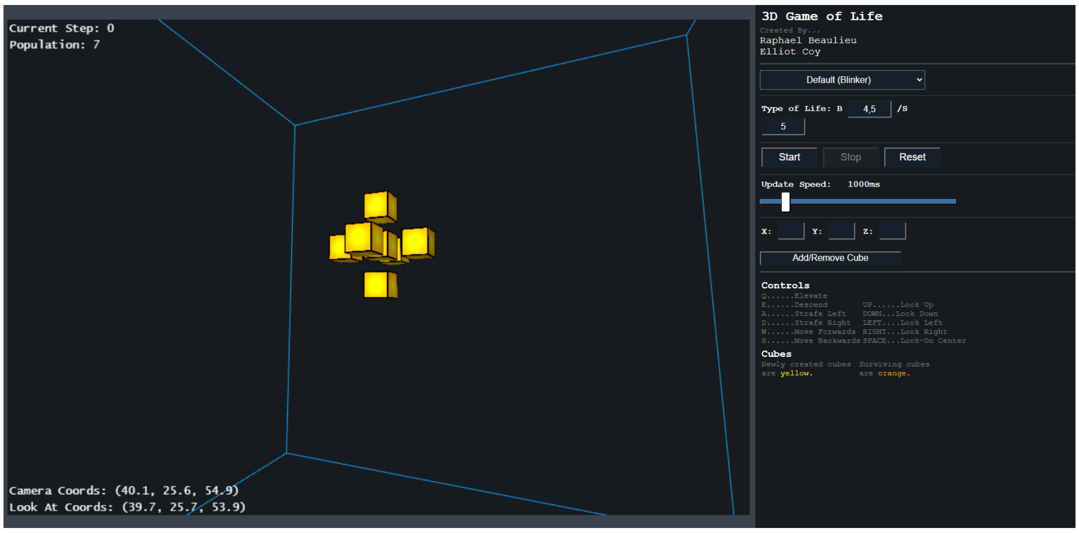


Figure 2.1: A screenshot of the playgameoflife website [35] by Edwin Martin.

The Game of Life also has its own wiki page, called "conwaylife.com/wiki" [33]. This website is the closest analog to the interactive lesson part of this project. It is structured like a generic Wikipedia page, with many of its basic explanations on the Game of Life, cellular automata, etc. being just simplified versions of their own Wikipedia entries. The "conwaylife.com" website will be referenced a lot within the actual lessons in this project, as it serves as the primary external resource for many of the patterns presented in them. Almost every pattern has a specific sub-page on the "conwaylife.com" wiki; most also contain an interactive window to simulate a pattern evolution. This functionality is, however, not embedded into the core of the page, and serves mostly as an additional way to examine the pattern, instead of as a key part of the learning experience.

Even the 3D rendition of the Game of Life is not a new concept, although online renditions of it are not as easily found compared to the 2D variant. The first result on Google when searching for a "3D Game of Life" at the time of writing this thesis will lead a user to the website called rbeaulieu.github.io/3DGameOfLife by Raphael Beaulieu & Elliot Coy [30], a screenshot of which has been provided in Figure 2.2. It features a simplistic interface with a render window and some options on the side. The rendered view cannot be interacted with using the mouse; keyboard keys have to be used. To add a new cell, or remove an existing one, its coordinates have to be entered. Playback options are equally limited as they only allow the user to start, stop or reset the simulation. The website leaves the impression of a mere proof of concept, instead of a finished 3D Game of Life sandbox, which is a goal of this project.



```
Current Step: 0
Population: 7

Camera Coords: (40.1, 25.6, 54.9)
Look At Coords: (39.7, 25.7, 53.9)
```

```
3D Game of Life
Created By...
Raphael Beaulieu
Elliot Coy

Type of Life: B 4,5 /S 5

Update Speed:   1000ms

Controls
Q......Elevate
E......Descend         UP......Look Up
A......Strafe Left     DOWN...Look Down
D......Strafe Right    LEFT....Look Left
W......Move Forwards   RIGHT...Look Right
S......Move Backwards  SPACE...Lock-On Center

Cubes
Newly created cubes  Surviving cubes
are yellow.          are orange.
```

Figure 2.2: A screenshot of the 3DGameOfLife website [30] by Raphael Beaulieu & Elliot Coy.

The projects introduced in this chapter are not the only iterations and renditions of the Game of Life in its many forms (e.g. dcode GoL [8], cuug GoL [7], creetah 3DGoL [6], Rithmschool 3DGoL [19], Chaos and Fractals [5], Wikipedia [27]), and describing them all would arguably be redundant, yet there is a clear space for a product like the one being developed here, that none of the existing solutions fill.

2.2 Importance of Interactivity in Teaching/Learning

As a slight detour before the Game of Life, the actual topic of this thesis, it was deemed important to outline some of the research and consensus that has been reached, regarding the significance of interactivity in teaching and learning. This is a relevant subject, since this project and the learning experience part of the website use just such interactive practices, to help the user better retain the information they will be presented with.
With this in mind, two papers focused on interactivity in learning/teaching have been examined.

"The students want to understand natural phenomena, to know scientific truths and to acquire knowledge to be applied in practice and for these reasons they are dissatisfied by the traditional education" [38]. According to the paper quoted here, most traditional methods employed in schools today are becoming less effective for the current generation of students. It is important to encourage discovery in learning and to let children experience things for themselves.
Traditional learning systems, that "put the focus on the teacher while students are the passive participants of the educational process" [28] are outdated. An engaged student is one that retains more information, and interactivity in the classroom keeps students engaged.

It is stated, that in comparison to traditional teaching methods, interactive learning uses ones own life experience. Students need to be able to tackle a specific problem in their own individual way. Interactive teaching helps people do just that [28].
"Interactive teaching styles are designed around a simple principle: without practical application, students often fail to comprehend the depths of the study material" [38].

Five guidelines have been designed by the ARMA International Center for Education [14], to encourage interactive teaching. Among them is the importance of student participation, using questions that stimulate a discussion, or the use of teaching aids, etc. These guidelines have been followed when designing the learning experience part of this platform.

CHAPTER 3

Cellular Automata, Game of Life & the Turing Machine

What follows is a brief explanation of the key concepts this project focuses on. Together, they form the core of the research, as well as the interactive lessons from the practical part of this work. These interactive lessons that are at the center of the developed website had to have a certain structure and progression. To show off the full potential behind an automaton like the Game of Life, in the later chapters of the lessons, the recursivity of the automaton is pointed out and explained. In order to understand why and how the Game of Life can recursively simulate itself, the concept of a Turing machine has to be introduced first. Throughout the rest of this paper, the terms "Game of Life", "Life" or "GoL" are to be considered interchangeable.

3.1 Game of Life

The Game of Life is a so-called cellular automaton. This is a grid of cells, each having a state. Cells inside this grid can only exist in a certain number of states, which they switch between. The number and/or type of states a cell can take on depends on the automaton. The cells in the Game of Life generally only have two states, dead or alive. Each cell has a specific neighborhood, based on which it transitions between states. This is the basic principle behind the evolution of a cellular automaton.

The Game of Life, as first imagined by the late English mathematician John Horton Conway in 1970, is often described as a zero player game, because the only input from the user happens at the very start, where they choose an initial cell configuration. After that, the simulation is run and the player can only watch. Conway is said to have always been intrigued by the unpredictability of similar imagined game universes devised by mathematicians of his time.

It was this unpredictability, in spite of seemingly very simple rules, he had in mind when he initially decided to create the automaton known today as the Game of Life. "Conway's name of life compares the growth of patterns to the growth of populations of living organisms - say, bacteria in a culture" [37] (page 15, chapter "Game of Life").
He experimented with many different rule sets, driving down their complexity at every iteration. What he came up with is, as he described it, a nongame. Everything except the initial input of the player is determined by his carefully devised, static rules. The situation at one moment directly determines the situation in the next one, therefore, everything in the Game of Life is predetermined.

Life generally gets cited as having four basic rules that determine the next evolution of the simulation. Rules determine, whether a living cell will die, or a dead cell is reborn. The rules are, in no particular order:

- A dead cell with at least three living cells in its neighborhood will be reborn.
- A living cell with more than three living cells in its neighborhood will die from overpopulation.
- A living cell with less than two living cells in its neighborhood will die from underpopulation.
- Any other living cell will keep living, unaffected.

The Game of Life is played on a 2D grid of cells, and each cells neighborhood is made up of the cells next to it, diagonally, vertically, and horizontally. This means that a cell in the Game of Life has eight possible directly neighboring cells, which it needs to examine, in order to determine its behavior in the next epoch.
The epoch refers to a state of the simulation at a particular point in time. Every time a specific configuration of cells evolves according to the 4 defined rules, this signifies a new epoch. Often, an epoch inside the Game of Life is called a period or generation.

3.2 Turing Machine & Turing Completeness

A Turing machine is an abstract model of a machine, that can simulate any given algorithm, despite its apparent simplicity [31]. It is a theoretical machine defined as having an infinitely long input tape, that binary values can be either read from or written to. Using simple instructions and binary operations, it is thus possible, given enough time, to simulate an algorithm of arbitrary complexity. The Turing machine is named after the British mathematician, Alan Turing, famous for inventing what is regarded as the first computer. Turing came up with the then called automatic machine in 1936, in order to prove that no machine can exist, which could determine if another machine ever prints out any given symbol, or if it freezes or stops computing.

If a collection of instructions is deemed Turing complete, it simply means, that it is capable of simulating a Turing machine. It is, therefore, possible for a Turing complete system to simulate all tasks accomplishable by current computers [22].

A core idea that this thesis heavily relies on is, that the Game of Life is also Turing complete. Or at least, that components like logic gates, the building blocks of a computer, can be simulated within Life, that make it so. And consequently, by nature of its Turing completeness, a Turing machine can and has indeed been built within the Game of Life. A fully-fledged, 8-Bit programmable computer, complete with the Arithmetic and Logic Unit (ALU, part of a CPU that carries out arithmetic and logic operations), memory, and even a display [34] has been created in Life, as seen in Figure 3.1.

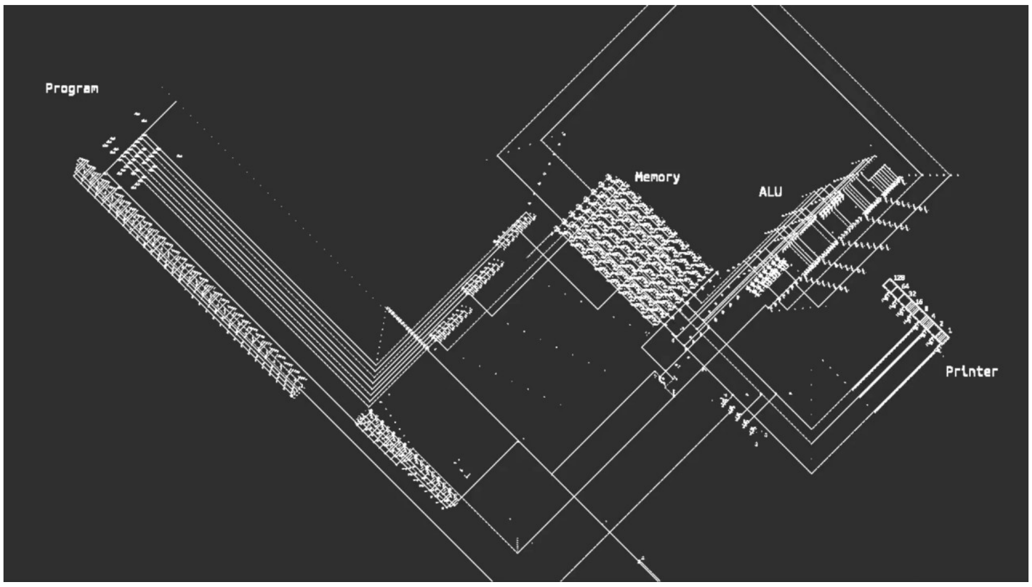


Figure 3.1: A basic programmable computer inside the Game of Life created by Nicolas Loizeau [34].

If taken one step further, it can also be said that the Game of Life is not only a Turing machine but a universal Turing machine. Whereas a standard Turing machine is capable of simulating any arbitrary algorithm, one would have to create a specific machine for each algorithm that is being simulated. The universal Turing machine, however, takes the standard input tape that has already been explained, along with a description of a machine that would solve a certain problem. With this, the universal Turing machine can take the input machine and, using the content of the tape, simulate it, and with it the algorithm itself. A universal Turing machine can simulate any other machine, and so can the Game of Life. For clarification, the machines that are being talked referenced are not actual physical machines. Rather, they are mathematical concepts similar to state machines or the already familiar automata.

3.3 Recursivity of the Game of Life

The fact that a computer can be built inside the Game of Life might not seem very interesting at first glance. After all, there are countless Turing complete systems, and thousands of examples online, of people, building simple computers out of unexpected things.

A rather surprising example of this is the trading card game "Magic: The Gathering" [15], which is also considered Turing complete. Meaning, using a certain subset of the total cards, the rules of the game allow for a Turing machine to be constructed. This means that a "computer" could be constructed out of "Magic: The Gathering" cards, that would itself be capable of playing "Magic: The Gathering". Another, more common example is the game Minecraft [16], with hundreds of tutorials and videos of people online, building computers inside the game.
These are examples of what is called unintentional Turing completeness. The creator of "Magic: The Gathering", although he was a mathematician, or the developer of Minecraft did not set out to create a Turing complete system within their games. It happened as a byproduct of the games' mechanics. The same goes for John Conway and the Game of Life, who did not initially set out to create a Turing machine.

Stemming from the fact that Life is Turing complete, it is provably possible to build a Turing machine, that can run Life itself. One can therefore simulate the Game of Life, inside the actual Game of Life. Figure 3.2 shows a grid of nine cells, with three of the cells lit up, representing the very common blinker pattern [4] documented in the Game of Life. The miniature grid itself is made up of cells, within a Game of Life simulation. This figure therefore shows Life being simulated inside Life.
Due to technical limitations of the browser platform, this was not included as an real-time simulation among the interactive lessons. Rather, a recording serving as a proof of concept was made, inside a program called Golly [39], which uses the so-called HashLife algorithm (first described by Bill Gosper, who discovered many of the most famous patterns in Life, like the Gosper Glider Gun [11], a very famous pattern, and also the first documented pattern with infinite growth) for improved simulation speeds and scale. The algorithm works by leveraging the considerable amount of repetitive behavior in many large cell clusters and patterns, inside the automaton. This is done by recording common patterns in a hash table so that the resulting evolution calculations do not need to be re-done if they turn up again [32].

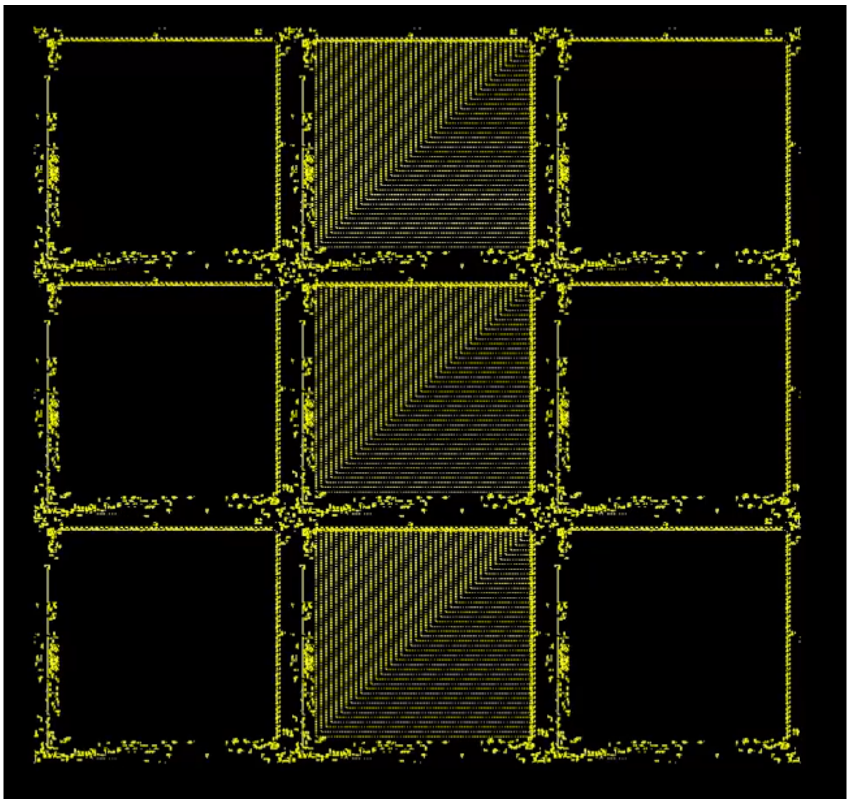


Figure 3.2: Life being simulated inside Life itself, using the open-source software Golly [39].

CHAPTER 4

Designing the Tour of Life

This chapter provides a description of the three major sections of the web application named the Tour of Life, their basic and advanced functionalities, and also gives insight into the major design decisions made during development. In a nutshell, this includes a 2D simulator of the Game of Life itself, where users can experiment with the algorithm, the Tour of Life learning experience, and also a 3D rendition of the Game of Life simulator.

4.1 2D Game of Life Simulator

Our proposed solution learning application for the Game of Life includes an interactive Game of Life simulation environment, where users can experiment with their newly gained knowledge about the automaton. To enable them to do this properly, users should be able to customize certain parts of the experience, like simulation parameters, UI, or even the algorithm itself. Most existing solutions present the user with a grid where they can place cells for the starting configuration and run or rewind the simulation, without many customization options. By looking at multiple online simulators for the Game of Life, the following list of basic, necessary features for playback control has been compiled for later implementation:

- **A way to start and stop the simulation** (Figure 4.1.a), once a starting configuration has been selected. The user has the option to change the state of cells on the grid at any time, even while the simulation is active. Depending on the playback speed, this will have varying effects, as the added cells might not have enough time to form into patterns, causing them to die immediately due to the lack of other cells in their neighborhood.

- **A way to progress the simulation by one step** (Figure 4.1.b), either forward or backward. This option allows the user to step through the simulation in a more controlled fashion compared to the playback introduced above, to properly experience all the changes that are happening. The option to step backward naturally only becomes available, after the simulation has already progressed. The number of saved steps backward is not unlimited, so as to save resources, but can be adjusted in the options menu which will be discussed later in this chapter.
- **An option to clear the whole grid** (Figure 4.1.c), changing the state of all living cells in the simulation to being dead. This self-explanatory feature allows the user to "kill" all cells at once, in case a pattern grows too big, and manual single-cell state change would prove too tedious.
- If the user has the option to clear all living cells at once, one must also account for the situation of accidental clearing. For such cases, an option needs to exist, to **restore the last cleared cell configuration** (Figure 4.1.d).
- Most of the researched Life simulators contain no instructions of background information, with no proper explanations for their features or working principles. This is why a sort of guide is needed, which would include a basic **description of the platforms features**, as well as the basic controls necessary to interact with the website.
- **The option to draw a cluster of cells**, instead of having to place them one by one is crucial to the user experience and ease of use. Similarly, a way to control the view, preferably without having to use the keyboard, gives the user freedom to easily explore the whole simulation.

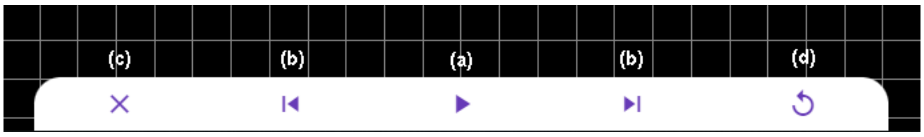


Figure 4.1: A snippet from the website showing the playback controls setup.

The features listed above already form a basic Game of Life simulator and allow the user to interact with the Game of Life in a restrictive manner. An important aspect of visualization is to give the user freedom to adjust the view-port to their liking, as well as change the parameters of the visualization. Not only does this help improve the user experience [12], it also helps with understanding the algorithm itself, especially when adjustments of the underlying simulation are possible.

This is why, based on more competition research and the state of the art, a second list of advanced features has been devised, included in a special options menu. The list of features includes:

- **An indicator for how many cells survived, died, or were born in the current epoch**. This feature gives the user a very important overview of the state of the simulation.
- **Options to adjust the underlying algorithm** for cell birth/death/survival. In order to let the users experiment and better understand the simulation and the algorithm for cell state transition, they are allowed to change the number of cells that are needed for a living cell to die, survive, or for a dead cell to be reborn. These options exists in the form of simple input fields, where the maximum allowed value is 8, and the minimum value is 0, since the neighborhood of a cell can have at most eight cells and at least zero. Changes to the rules of the Game of Life can be made even while the simulation is running.
- To help make the state of a specific pattern at the next iteration clear to a layman, a visualization is necessary, which contains a **preview of the changes that are about to happen** to the living cells, between the current epoch and the next one. This feature is called the "prediction mode", and it exists in the form of a toggle that enables such a view. This view gives cells a specific color-coding, based on what is about to happen to them. This coding, and why it was chosen will come up in more detail in Chapter 4, Section 4.
- Just as important as giving users freedom when exploring the simulation, is to limit the inexperienced ones, in order not to overwhelm them. By default, there are limits, set on the zoom and pan of the camera within the view-port. These limits need to be toggle-able, to give users freedom, should they explicitly request it. The "**developer mode**" toggle does just that.
- **A way to control the speed of the simulation**. Using a slider, users should be allowed to adjust the speed of the playback. This changes the delay between epochs and makes the evolution of the cells faster or slower. This feature obviously affects performance, so different simulation speeds may work variously on different systems.
- **The option to change the background and/or cell color**. An intuitive way is needed, preferably using sliders, to adjust the color of the board and the cells within it, on the fly. This helps especially color-blind users to better interact with the simulation.
- The sandbox includes a **helper grid** in the background, as a visual aid to allow for easier cell placement. An option should exist, to toggle this grid, in case users want to examine the cells without the grid, as in cases when the camera is zoomed out too much in developer mode, it can cause distracting flickering artifacts.

This set of options and features gives the user full control over the simulation and makes interacting with it very intuitive. By letting users change the look of the simulation, and even the algorithm itself, it is ensured, that most easily solvable questions and problems with the application can be solved by the users themselves (e.g. visibility, understandability, visual clutter).

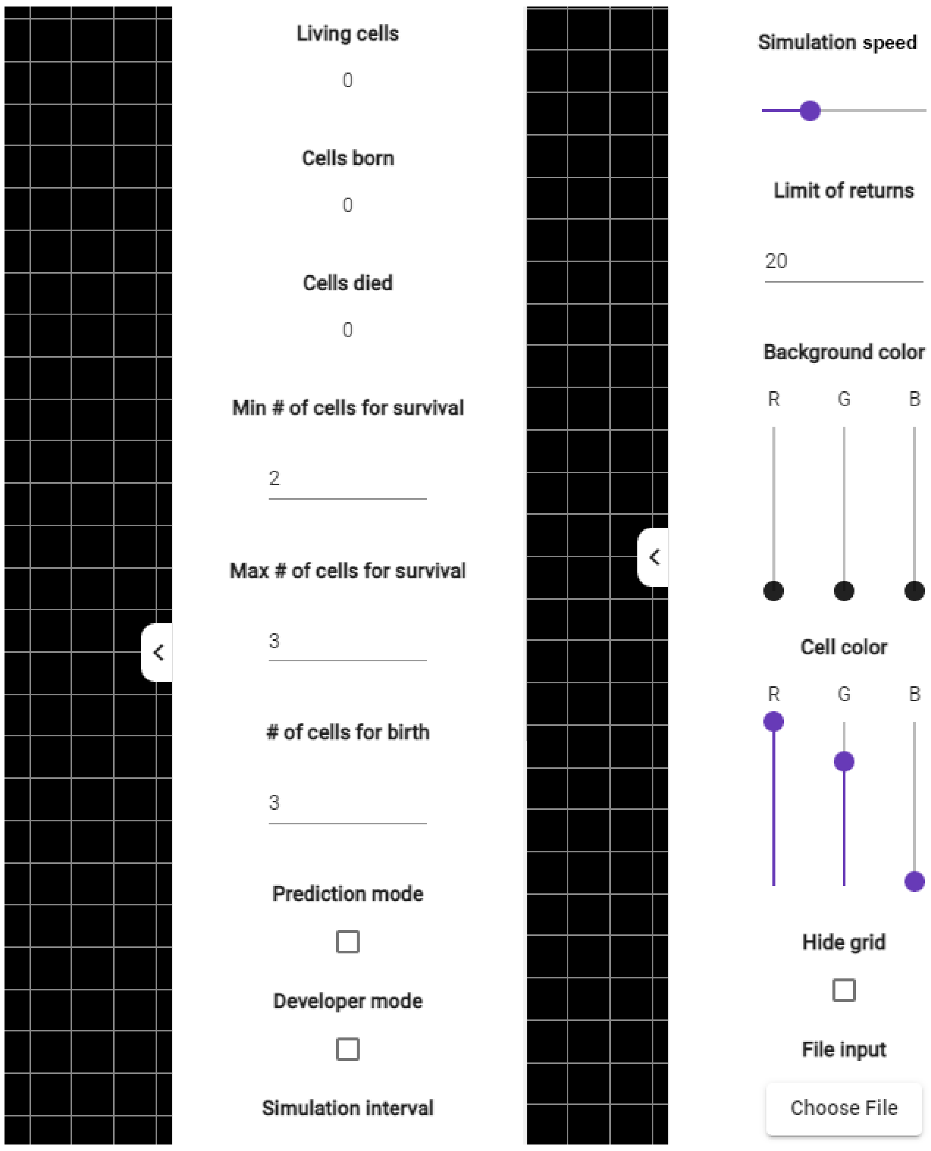
One additional feature will be added to the 2D simulator, as an way for users to personalize their initially drawn cells, and also as an additional way to differentiate this platform from competitor solutions. The very last feature in the options menu allows a user to upload an image in the format jpg or png, which then gets automatically converted into a starting cell pattern. The process will work by extracting the bright spots of an image above a certain threshold and drawing a cell in this position in the simulation grid.

With customizability in mind and including the features described in this Section, the simulator sections of the application have been designed to be as customizable as possible, to prevent the feeling, the user is watching a pre-recorded evolution of a specific pattern, but instead that they can directly interact with what is shown, and how it is shown.

4.2 Tour of Life

The Tour of Life section of the platform is the core premise of the whole project. It is a series of interactive lessons, where users, less familiar with the concepts of automata and computing can experience the complexity of Conway's Game of Life, from its very basic rules, all the way to the fact that it is Turing complete, and what that means.
We propose for the lessons to have the form of small, self-contained blocks of information, that slowly increase in complexity as the user completes them. This segmentation serves to make the amount of presented information the user is confronted with less overwhelming. Originally, only theoretical lessons were planned, where users can interact with the presented cell pattern, yet user interaction was not necessary to complete the lesson. The exercises were later introduced to allow users to engage with the Game of Life in a more thorough manner by having them complete a pattern by themselves, as it provably helps with information retention. The Tour of Life is also intended for younger audiences, around the high-school age, where interactivity is highly desired to keep the users' attention [28].

The Tour of Life is split up into individual chapters, each chapter consisting of two lessons or exercises. These chapters have been designed to include thematically related lessons, to prevent jumping from topic to topic at random. Once a chapter introduces a theoretical topic or pattern from the Game of Life, there may later come up an exercise, which tests the user on said topic. This means, that every exercise has a corresponding theoretical lesson connected to it, and should therefore be easily solvable.


Living cells
0
Cells born
0
Cells died
0
Min # of cells for survival
2
Max # of cells for survival
3
# of cells for birth
3
Prediction mode
☐
Developer mode
☐
Simulation interval

Simulation speed
Limit of returns
20
Background color
R G B
Cell color
R G B
Hide grid
☐
File input
Choose File

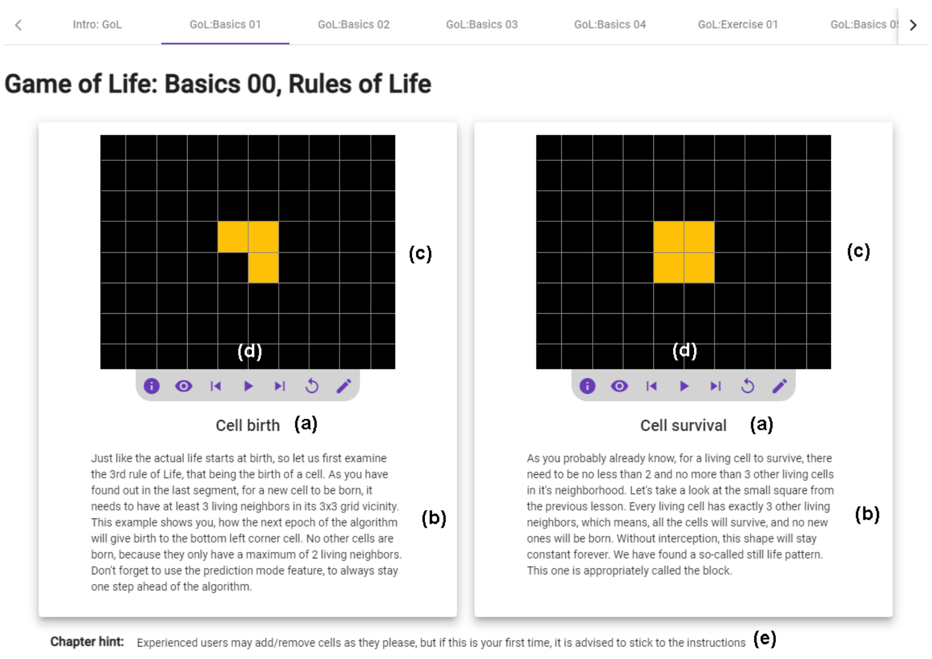
Figure 4.2: Design and arrangement of the options listed in Chapter 4, Section 1, inside the implemented options menu.

The content of the lessons is as follows. First, a brief introduction to the idea behind and the functionality of the Tour of Life is presented. What follows are lessons on the Game of Life itself, its basic rules, and the most important patterns, that help the user understand the automaton more deeply. The lessons also gradually increase in complexity. After that, the focus switches to computer logic, and logic gates inside the Game of Life. This is a brief departure on the way to the final chapters of the lessons that have to do with the Turing machine.

The user will experience what a Turing machine is, and how it can be built inside the Game of Life, using the presented logic gates as building blocks, as they can form a Turing complete system. Finally, using this knowledge, it is presented to the user, that Life can indeed simulate itself. This is done using a pre-recorded video taken from the software "Golly" [39], as such a simulation would be too costly for most browser applications.
Each lesson usually consists of a simulation window, simulation controls, a lesson title, and the lesson itself. A detailed description of the individual parts of a lesson is as follows:

- **Lesson title:** (Figure 4.3.a) Generally describes the core idea of the lesson. This can be related to the algorithm of the Game of Life, specific patterns that emerge within the automaton, or even the basics of computer logic.
- **Lesson description:** (Figure 4.3.b) This part includes the actual lesson, specifically, what the user should remember, and how to interact with the presented pattern inside the simulation. It outlines why a pattern was chosen, categorizes it, and describes how it behaves. The lesson description is intended to help users understand the pattern presented to them before they actually start interacting with the simulation window.
- **Simulation window:** (Figure 4.3.c) The simulation window within any given lesson works analogously to the 2D sandbox described in Chapter 4, Section 1. Users can draw and delete cells by drawing on it, as well as zoom and pan the camera to a degree. For the initial exercises, the zoom on the camera is locked, to force the user to focus on the pattern being introduced to them. Later lessons allow the user to zoom/pan, as the patterns presented get bigger and more complex. The zoom settings and restrictions are carefully chosen to not make the simulator inside the lesson environment confusing, and also to direct attention to the important parts of the lesson.
- **Simulation controls:** (Figure 4.3.d) In addition to the already examined basic playback controls, the lesson simulator includes a button to reset the simulator. With this, the user can reset the pattern that is being discussed in the lesson, as well as any camera adjustments, back to its original state. A prediction mode button has also been added directly to the controls bar, taking it out of the options side menu in the 2D Game of Life simulator. Its functionality has also been described in Chapter 4, Section 1. A button is added, which links to every lesson a related online resource, allowing the user to more deeply research the topic at hand, should they find something unclear. Lastly, the user has the option to transfer the pattern of any given lesson into the 2D sandbox environment, where they can experiment with it using the full suite of options discussed in Chapter 4, Section 1.

In addition to these basic features, some chapters may contain a hint at the bottom of the screen (Figure 4.3.e). This short suggestion serves to give users additional information regarding the chapter or help them during an exercise.



Game of Life: Basics 00, Rules of Life

Cell birth

Just like the actual life starts at birth, so let us first examine the 3rd rule of Life, that being the birth of a cell. As you have found out in the last segment, for a new cell to be born, it needs to have at least 3 living neighbors in its 3x3 grid vicinity. This example shows you, how the next epoch of the algorithm will give birth to the bottom left corner cell. No other cells are born, because they only have a maximum of 2 living neighbors. Don't forget to use the prediction mode feature, to always stay one step ahead of the algorithm.

Cell survival

As you probably already know, for a living cell to survive, there need to be no less than 2 and no more than 3 other living cells in it's neighborhood. Let's take a look at the small square from the previous lesson. Every living cell has exactly 3 other living neighbors, which means, all the cells will survive, and no new ones will be born. Without interception, this shape will stay constant forever. We have found a so-called still life pattern. This one is appropriately called the block.

Chapter hint: Experienced users may add/remove cells as they please, but if this is your first time, it is advised to stick to the instructions

Figure 4.3: Standard layout of a lesson chapter inside the Tour of Life.

Around 25% of the currently available lessons do not include a simulation grid, and instead consist of simple text. These special lessons serve as introductions to certain topics, especially when transitioning between lessons about the Game of Life, to the introduction to computer logic. No fitting visual example has been found for these isolated lessons, so the simulation window has been omitted entirely.

Another special case of lesson that has been mentioned already is the exercise. It has the same format as the theoretical lesson, with the addition of a cell counter, displaying cells available for placement. The user has to complete the exercise using only the number of cells available to them for that exercise. It is still possible to delete existing cells, yet this will in most cases not lead to the desired solution. No system exists to check if the provided solution is correct, because usually, more than one acceptable solution to a given exercise exists. The user has to have the ability to discern, whether the provided solution depicts the required pattern, as outlined in the individual exercise description. The user can continue on with the lessons even when an exercise has not been completed.

To navigate the lessons, there is a navigation carousel with lesson IDs at the very top of the screen. The user can switch between lessons at will, later lessons are not locked until the lessons before them are completed. This is so that experienced users do not have to complete the whole course in order to get to the information they actually want and to enable a fast and quick refresh of forgotten information without the need to first complete the previous chapters.

4.3 3D Game of Life Simulator

The last section of the Tour of Life website enables the curious user to explore how Conway's Game of Life would behave in a 3D environment. Regarding the UI, the 3D Game of Life simulator may look very similar to its 2D counterpart. This is true when it comes to available features and options. The playback and options menus were left virtually unchanged, as the additional dimension does not add much in terms of functionality, and also to enable easier comparison between the 2D and 3D views. The most challenging part of constructing the 3D interactive Life environment is the interactivity with the cells themselves.

One additional problem the 3rd dimension introduced to the visualization of cell patterns is with the clear visualization of the prediction mode, which finds a use in the 3D simulator as well. The final implementation of this feature is further explored in Chapter 4, Section 4.

A feature that distinguishes the two kinds of simulators is the file upload. For the 3D simulator, the user can upload an obj file of a 3D mesh, which in turn gets converted into a cluster of cells inside the simulation. Just as with the 2D image upload, here too, the bigger the input mesh, the longer the upload.

4.4 Visualization & Accessibility

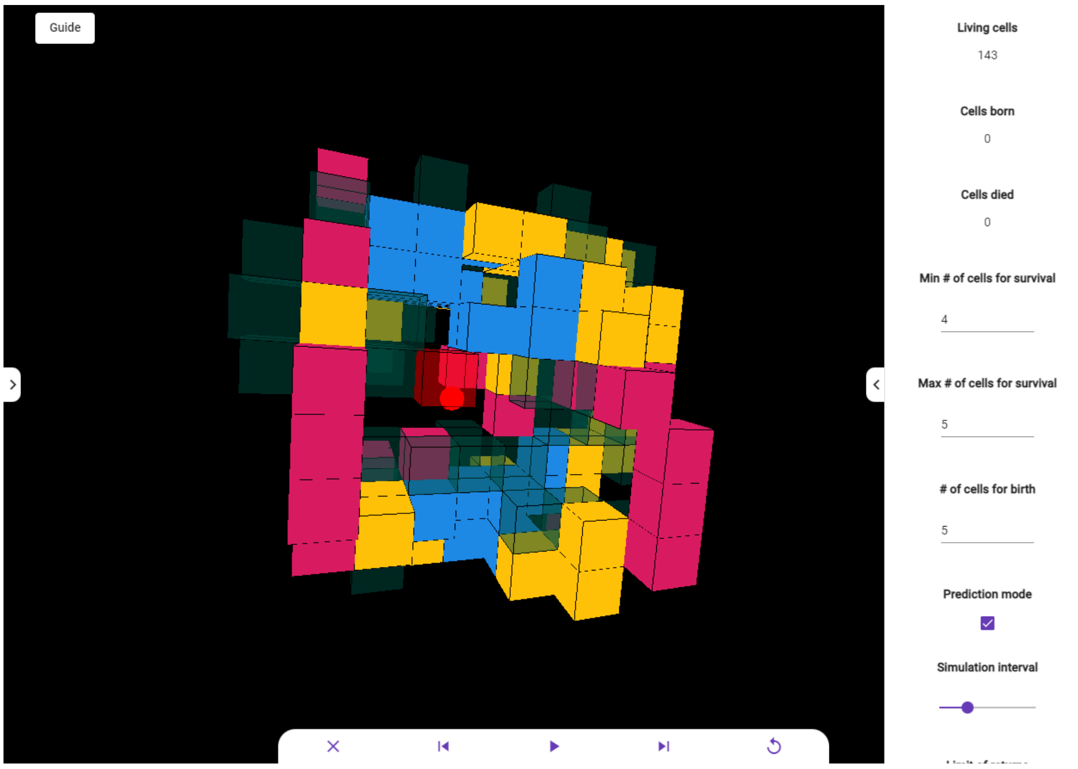
There are multiple visualization problems that have to be tackled during the development of the website, to make it accessible and usable for the majority of people. When analyzing existing solutions, it was always interesting to see what colors were used to depict the simulation grid as well as the cells themselves. In most cases, no option was provided to change these colors, meaning if they did not fit for any reason, there was nothing the user could do.

As the main color for the cells of the Tour of Life website, a dark yellow was chosen. This is because it contrasts well with the black background, and also because yellow is the brightest saturated color possible, and it is clearly distinguishable from the selected background even for colorblind users.
For the issue of visualizing multiple types of cells in the prediction mode of both the 2D and 3D sandbox, specific colors were chosen, that would exhibit a clear distinction from one another, even to people who suffer from colorblindness [36]. Specifically, users with protanopia, deuteranopia, and tritanopia should have no problem distinguishing between the chosen colors. The yellow used throughout this project is actually part of the prediction mode color palette, to ensure that even with the prediction mode enabled, there still is a clear distinction between all the visible colors. There was an attempt to logically color-code the individual states of the predicted cells. Yellow cells would indicate no change. Green means a new cell will be born. A blue cell dies of overpopulation and a pink one dies of underpopulation.

This encoding, although it might make sense to us (green means birth, red means death, etc.), is not carried over into the colorblind-safe palette. Nonetheless, there are still clear distinctions between the different types of cells.

A second major issue regarding visualization is figuring out a proper way to present a 3D cluster of cells to the user, ensuring a high level of interactivity. Since the user is dealing with a 3D shape, the basic camera pan from the 2D sandbox is not sufficient, as cells would remain obscured along the depth axis. Therefore, camera controls are expanded upon here with the addition of camera orbiting. More specifically, the camera always orbits around a fixed point in the coordinate system, that being the absolute center, unless the camera is moved via panning.

A final obstacle that will have to be overcome is the visualization of the prediction mode in 3D space. This poses a problem since due to the opacity of the cells, the state of cells inside any 3D shape would naturally be obstructed. Here as well, a few approaches are possible. Having every cell fully opaque clearly is not sufficient, due to the issue described. A test with all affected cells being made semi-transparent also has not worked out well, since the transparency leads to great confusion, due to visibility issues, and unintended color mixing. It was finally decided, that it makes sense for the cells that will be born in the next epoch, to be made semi-transparent. This removes a lot of the visual clutter, especially in small patterns, and allows for better viewing of the cells underneath. The color scheme remains.



Living cells
143

Cells born
0

Cells died
0

Min # of cells for survival
4

Max # of cells for survival
5

# of cells for birth
5

Prediction mode
☑

Simulation interval

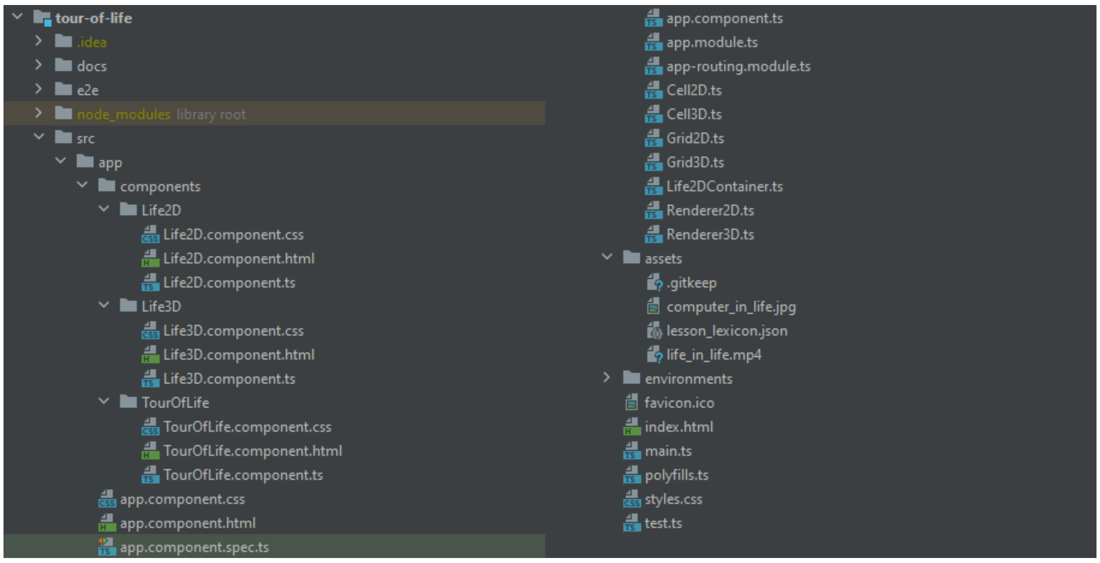
Figure 4.4: A random cluster of cells, displayed with the prediction mode enabled. What is visible is the colors used, as well as the way the mode has been realized in 3D space.

CHAPTER 5

Implementation

What follows is a detailed description of some of the development challenges and design decisions that determined the overall structure of the project. Core ideas and functionality of the main components of the website are outlined here, with a detailed explanation regarding the algorithms used, and programming techniques employed.

5.1 General Project Structure



```
tour-of-life
  .idea
  docs
  e2e
  node_modules library root
  src
    app
      components
        Life2D
          Life2D.component.css
          Life2D.component.html
          Life2D.component.ts
        Life3D
          Life3D.component.css
          Life3D.component.html
          Life3D.component.ts
        TourOfLife
          TourOfLife.component.css
          TourOfLife.component.html
          TourOfLife.component.ts
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
      app-routing.module.ts
      Cell2D.ts
      Cell3D.ts
      Grid2D.ts
      Grid3D.ts
      Life2DContainer.ts
      Renderer2D.ts
      Renderer3D.ts
    assets
      .gitkeep
      computer_in_life.jpg
      lesson_lexicon.json
      life_in_life.mp4
    environments
    favicon.ico
    index.html
    main.ts
    polyfills.ts
    styles.css
    test.ts
```

Figure 5.1: Tour of Life project structure inside the IntelliJ IDEA IDE [13].

The Angular framework mainly relies on reusable components for parts of the website that are shown to the user. For this project, three basic components were needed.

The `Life2D` component handles the 2D sandbox part of the website, the `TourOfLife` component holds the code necessary for displaying the lessons, and `Life3D` is used for the 3D sandbox. These three components are all saved inside individual folders, all within a folder called `components`. Functionality not directly related to the displaying of components is housed outside the `components` folder, inside the parent folder called `app`.
Since the basic functionality of the 2D Game of Life gets reused twice, once for the sandbox, and once for the lessons, it was extracted into a separate class called `Life2DContainer`. This class holds the basic logic behind the 2D Game of Life, which can be reused at multiple points in the project, and to which additional functionality can be added.
Additional classes that do not belong to any component are `Cell2D` & `Cell3D`, `Grid2D` & `Grid3D`, and `Renderer2D` & `Renderer3D` for the respective simulators. It was unclear in the beginning if the 3D rendition of the Game of Life was going to be included in the final version of the project; that is why the two parts were developed independently, and their logic is contained in separate files. Since there were only two instances of each class, the use of an interface was also deemed unnecessary.

5.2 2D & 3D Rendering

In order for the 2D/3D graphics to be displayed, a component needs to have a Three.js renderer object, together with a Three.js scene and a camera. The scene houses any objects that should be displayed at any given render pass. So as to not make a component class too cluttered, the renderer functionality has been extracted into its own class, called `Renderer2D` & `Renderer3D` for their respective uses. These classes would house the root scene object, as well as the camera and the renderer itself, in addition to any camera settings and canvas dimensions. The renderer class would take care of any zoom and pan restrictions.
When a new `mesh` object needs to be rendered, it simply gets added to the active renderers scene. Within the next render pass using the `animate` method, the scene gets drawn onto the canvas element inside the DOM. The canvas element needs a container element, based on which the size of the actual canvas gets calculated. For this purpose, a simple `div` container was used, with the ID "render_window". The container then acts as a parent element to the renderer canvas.
The camera used for this project is a simple Three.js `PerspectiveCamera`. It is used for both the 2D and 3D versions of the Game of Life simulation, as in reality, the 2D Game of Life grid is also a projection on a 2D plane.

5.3 Cell Placement/Deletion

A set of three event listeners have been set up to check for mouse button presses, releases, and general mouse movement using the `addEventListener` method on the render window, with the types `mousedown, mouseup & mousemove` respectively. When a mouse button press event is detected, first, a `mouse_down` flag is set to true. The release of the button sets it back to false. Only if this flag is set, does the movement of the mouse have any effect on the cell states. This approach allows the user to paint in the cells by moving the pressed mouse, instead of having to individually place/delete each cell via a button-press.

The actual cell creation/deletion is handled by their respective methods `checkAddition() & checkDeletion()`. Both methods are based on the same principle. An invisible plane the size of the whole grid is placed at a depth of zero onto the grid. This plane is called the `raycast_plane`. When a click event is detected, left-click meaning a cell is to be added, right-click meaning a cell is to be deleted, a raycast is performed, with the ray pointing from the position of the camera, towards the x/y position of the mouse, with the depth set to zero. The position of the collision with the `raycast_plane` is then recorded, and a cell position is calculated from it. This is a necessary step, because the origin of the displayed grid is the same as the world origin, meaning a cell cannot be placed on either x=0 or y=0 (Figure 5.2). Therefore, if the x or y value of a ray collision is negative, it gets further decreased by 1, whereas if the value is positive (or 0), it gets increased by 1.

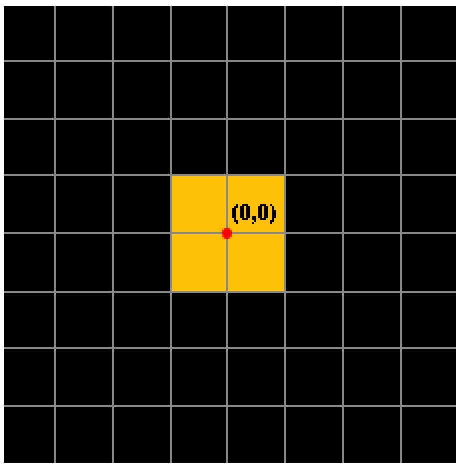


Figure 5.2: This graphic depicts the center of the coordinate system to demonstrate, that no cell can be placed at the coordinates (0,0).

The way a cell gets deleted works analogously to cell placement, the only change being the target of the raycast. Instead of checking for collisions with the plane, the ray is set to collide with any existing cell. Should such a collision be detected, its coordinates get extracted, and using the same call position calculation as described above, the cell corresponding to the newly calculated coordinates is deleted. What was described here is the process of adding/removing cells from a 2D grid, although the process is almost identical when working inside the 3D simulator. All the changes between these approaches are outlined in Chapter 5, Section 8.

The way cells are saved and interacted with is all saved within the respective grid classes, as outlined in Chapter 5, Section 1. There are two variables that store the existing cells. The first one is a 2D array called `coords`, which holds a reference to an existing cell at the coordinates of the cell itself. This approach was chosen to easily determine if a cell has neighbors and to quickly edit a single specific cell. A second, simple 1D array called `active` saves references to all the currently existing cells, in the order of their creation. This variable is used for quick editing of all existing cells, and also to quickly pass existing cells to the scene for rendering.

5.4 2D Game of Life Algorithm

In its very simplest form, the algorithm for the Game of Life consists of two steps. Constructing a list of all the cells that need to be deleted before the next epoch, and constructing a second list of all the cells that need to be created before then. Logically, the addition/deletion of cells cannot happen as the algorithm is being executed, since this would influence the outcome. All changes have to first be saved, and only as the last step can the corresponding cells be added/deleted.
The way this project tackles the issue is by having two additional arrays called `to_birth & to_die`, that hold the cells that are determined to be born/die in the next epoch. A void method called `advance()`, which is triggered at the end of every epoch, first resets these arrays to be empty. Next, all currently living cells are examined by looping over the `active` array and checking their neighbors using the proprietary `checkNeighbors(x:number, y:number)` method. This method examines the 3x3 neighborhood of a specified cell. If a dead cell is detected inside the neighborhood, the method `cellBirth(x:number, y:number)` is called, to check if it should be reborn. This is done by counting the living cells in the dead cells neighborhood. If this number is equal to, or greater than three, a new cell object is created and saved inside the `to_birth` array.
Finally, the `cellDeath(cell:Cell)` method determines, whether the currently examined cell itself should die or continue living, by again counting the neighboring cells. If the number of neighbors is between two and three, the cell survives. Otherwise, it is added to the `to_die` array.

As a final step, both the arrays are looped over, and the individual cells are either added to the grid or deleted from it. The new state of the grid is examined by the parent class, and the currently living cells are added to the rendered `scene`.

5.5 Prediction Mode

The prediction mode is one of the more important features of the Tour of Life experience, as it helps visualize the differences between epochs and helps the user better understand the algorithm. The core idea behind this mode is to split the existing set of living cells into four groups; survivors, cells that will die of overpopulation, underpopulation, and a fourth group of cells that will be born in the next epoch. To make this separation obvious to the user, these cells have to be color-coded. The specific color-coding and its meaning have already been previously discussed in Chapter 4, Section 4.
Because of this grouping of cells, additional arrays had to be introduced to store them. Namely `lonely & overcrowded`, which would store the under- & overpopulated cells respectively. To recolor the cells that are yet to be born, the `to_birth` array could be repurposed, as it already stores cells that will be created in the next epoch. All the remaining cells would stay unchanged.
Once the prediction mode was toggled by calling the method `predictionMode()`, it itself would call another method, called `predict()` which holds the actual logic for coloring the cells.

The method first clears the values for the necessary arrays described above, after which it calls the `checkNeighbors(x:number, y:number)` function, described in Chapter 5, Section 4. The `predict()` method additionally also saves cells inside the respective arrays based on the cause of their "death". After all the necessary arrays are filled, they are looped over, and the cells stored in them are colored accordingly.
Once the prediction mode is turned off, all newly added cells are removed again, as they are not really part of the active cells yet, and all cells saved in the `to_die` array are simply re-colored with the default yellow. Users can even run the simulation with the prediction mode enabled.

5.6 Image Files as Input for Starting Cell Configuration

The method called `loadImage(e:event)` is responsible for taking the input file of an HTML input element, which only accepts png and jpg file types, and converting it into a starting cell configuration, as can be seen in Figure 5.3. It does this, by first defining an isolated `canvas` element with a `context`, onto which a downscaled version of the input image is drawn. The downscaling is done to prevent a crashing of the website when images with a too high resolution are uploaded. Once the image is fully loaded and drawn inside the `context`, its image data is extracted. This is an array containing values from 0 to 255, in each of the R, G, B color channels, and the additional alpha channel. Inside a loop, this data gets averaged for each pixel, as a simple brightness estimate. If the

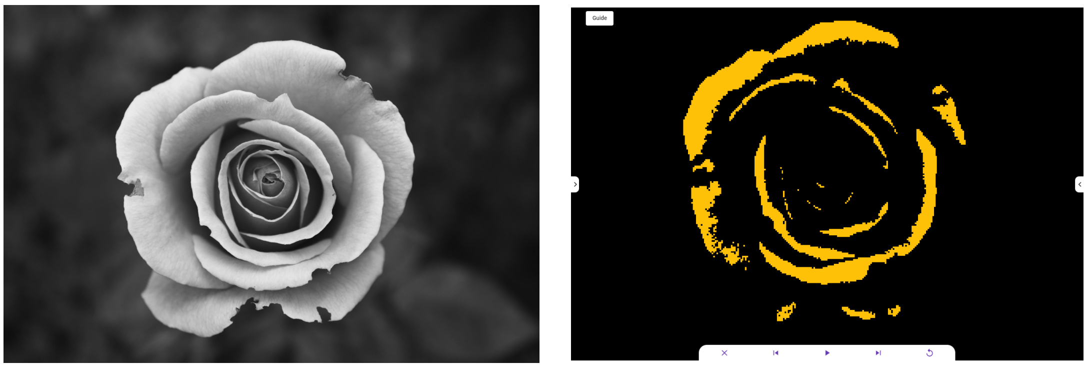


Figure 5.3: The input image (left), and its representation as a pattern of cells inside the Game of Life (right).

average value exceeds 200, the x/y position of the pixel gets calculated, and in its place, a cell is placed on the grid. The value of 200 has been determined through testing, as it eliminated a lot of the brighter pixels, making the calculation faster, yet it still preserved enough details for the input image to be recognizable.

5.7 Lesson Lexicon

The `lesson_lexicon` is a JSON file, holding the textual descriptions and configurations for all the standardized lessons and exercises of the Tour of Life. Instead of hard-coding the lessons directly in HTML, making the whole structure of the HTML part of the component unreadable, a file holding custom JSON lesson objects was created, and imported into the `TourOfLife` component. Thanks to Angular's ability to use loops to dynamically create HTML elements, the framework for each individual chapter is created based on the number of elements inside the lesson lexicon. In order to save resources, and due to the limitation of the Angular material tab component used, the content of a tab is not loaded automatically on creation, and therefore cannot be referenced. This means that the actual lesson resources are only loaded once the corresponding chapter is opened. This happens by fetching the ID of the tab opened, and using the ID as the index to search for the correct chapter in the lesson lexicon.
In addition to the lesson titles and descriptions, a lesson entry in the lesson lexicon contains the starting cell configuration for the pattern that is being presented, in the form of x/y coordinates saved in an array. This pattern is then displayed in a simplified grid, based on the 2D Game of Life simulator. Zoom settings and the speed of the simulation are also hard-coded here, as they have been individually determined for each lesson.

5.8 3D Game of Life interaction and visualization

The user has to be able to intuitively work with the existing cells and even more difficult, they have to be able to delete existing cells and add new ones. This is very difficult to achieve, using a 2D screen to depict a 3D environment. Several approaches have been examined. One such attempt involves separate controls for the depth axis, with which the user could determine at what depth a ray-cast would place the cell. This means that two hands were necessary to perform the basic process of adding/removing cells, which is highly unintuitive, especially compared to the 2D sandbox. The natural way to solve this issue is to bind depth to the scroll wheel, which is, however, already bound to zoom here. It has to be assumed, that the user does not have more than three mouse buttons to work with, so a compromise had to be made. A so-called "edit mode" will be introduced, bound to the E key on the keyboard, which switches between the depth at which cells are being currently drawn and zoom on the mouse.

To indicate depth visually, a 3D spherical cursor is added. This opaque sphere reflects the future position of the to-be-placed cell. Being a sphere, however, it could be unclear where a cell would get placed, especially without any neighbors as reference. This led to the addition of a semi-transparent cell that follows the spherical cursor and calculates the corresponding cell position to display exactly where the next cell would be placed. Additionally, being slightly larger than a normal cell, and having a different color from the default yellow for cells, the preview cell highlights an existing cell red, when the player hovers over it. This also simplifies cell deletion.

5.9 3D Cursor Movement and Position Calculation

To make the interaction with the 3D version of the Game of Life feel natural, and intuitive, a 3D cursor was designed, which allowed for movement along all three of the existing axes using just the mouse. It was, however, important, to preserve the users ability to zoom and orbit the camera around the center of the coordinate system. The movement of the camera would, without any changes to the way the position of the cursor is calculated, simply leave the cursor in its place in world space. This would lead to incorrect movement of the cursor, as it would solely depend on the mouse coordinates, and not take the camera position and perspective into account.
To achieve depth movement of the cursor, a vector is calculated, by subtracting the current position of the camera from the position of the cursor. The cursor is then moved along this vector, using the scroll-wheel.
The difficult part was the movement of the cursor in the x/y direction, following the movement of the mouse on the 2D computer monitor. This was solved by adding an invisible plane to the scene, similar to the one in the 2D simulator, also called `raycast_plane`. This plane was set up to always face in the direction of the camera every time the mouse was moved. A raycast is then performed, and the position of the collision becomes the new position of the 3D cursor.

This way, as the mouse moves on the monitor, each time, a ray is cast onto the `raycast_plane`, and the cursor position is updated accordingly.

5.10 3D Game of Life Algorithm

There is no difference between the algorithm for the 3D Game of Life, and the original 2D one, other than the added third dimension. Every function where the neighborhood of a cell was examined had to be extended by one additional axis, and the `coords` array was changed from a 2D array to a 3D one. Besides that, no changes were necessary, due to the way the algorithm was set up in the 2D version so that an easy extension into the third dimension was possible. The only notable change that was made, was to the basic rules of Life themselves. Since the neighborhood grew by a factor of three, due to the added dimension, the rules had to be adapted accordingly.
Multiple different configurations were experimented with, starting with the obvious analog of the standard 2D rules, used in 3D. For a cell birth, the standard 2D algorithm requires at least three out of the eight possible neighboring cells to be alive. For the 3D version of the algorithm, this would mean that at least nine out of the possible 26 neighboring cells are needed for a cell birth. Carrying on as in the original algorithm, a cell with less than eight, or more than nine neighbors would consequently die of underpopulation or overpopulation, respectively. The small gap between the two types of deaths meant however, that very few cells would survive an epoch. A researched paper [29] focused on the Game of Life in 3D and necessary changes to its rules explores exactly this problem. According to the paper, the ideal rule-set for the three-dimensional Game of Life could be achieved, by adding two, to the existing rule-set from the original algorithm. This meant, that five cells would be needed to birth a cell, and for a cell to survive, it would need at least four, and at most five neighbors. As cited in the paper, this configuration meant, cells required more time to form stable patterns, resulting in a greater possibility for interesting patterns to emerge. "Perhaps the most fascinating feature of [this adjusted rule-set] is that there exist an abundance of small stable and oscillating forms that usually exhibit symmetry of some sort" [29], the paper determined. After some amount of experimentation with this rule-set, and accidentally finding a period-4 oscillator as shown in Figure 5.4, the rules were kept. Just like in the 2D sandbox, however, the rules can be adjusted at any time, using the options menu.

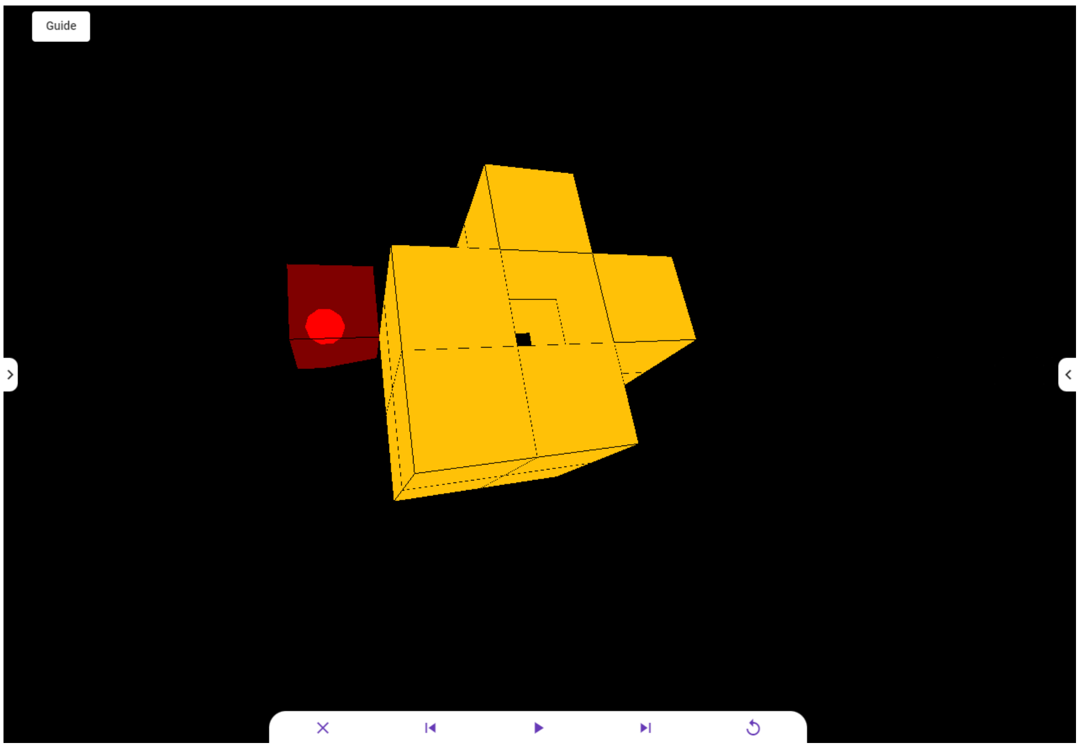


Figure 5.4: Depicted here is an oscillator that repeats after every 4 evolutions, within the 3D Game of Life, discovered by accident when testing various rule-sets.

5.11 Mesh Files as Input for Starting Cell Configuration

The method called `loadMesh(e:event)` handles the processing of an input obj file and turns this data into the initial 3D cell configuration.
First, the actual mesh data gets extracted from the input file using the Three.js `OBJLoader()`. Next, the bounding box of the mesh is calculated, using the method `computeBoundingBox()`, also provided by Three.js, which can be used on a geometry object. It is often the case, that the origin of a mesh, especially when extracted from third-party 3D editing software, does not correspond to the world origin. For this reason, once the bounding box was calculated, the whole mesh was moved based on a vector derived from the position of the bounding box.

Two approaches have been examined over the course of the development process, for extracting cell positions from the mesh file. The first approach involved testing every position inside the bounding box with a very small step size, to preserve a lot of the detail of the mesh, and subsequently casting a ray from this position straight up. If the number of collisions of said ray with the mesh was odd, this meant, that the starting position was inside the mesh, and a cell was to be placed in its spot.

This approach worked, however, it treated every object as solid, meaning it also rendered the inside of each mesh. This approach was ultimately discarded, as the calculation of cell positions would grow uncontrollably with increasing size of the input meshes, and also because it was ultimately useless to render the inside of any given mesh, as due to the rules of the Game of Life, all the inner cells would die out within the next epoch due to overpopulation.
The second approach, the one used in the final version of the project, involved casting a ray from every position along all six faces of the bounding box, in the direction of the inverted normal of the box. If the ray collided with the mesh, the position of the collision was recorded and from it, a cell was generated. The result of this operation can be seen in Figure 5.5.

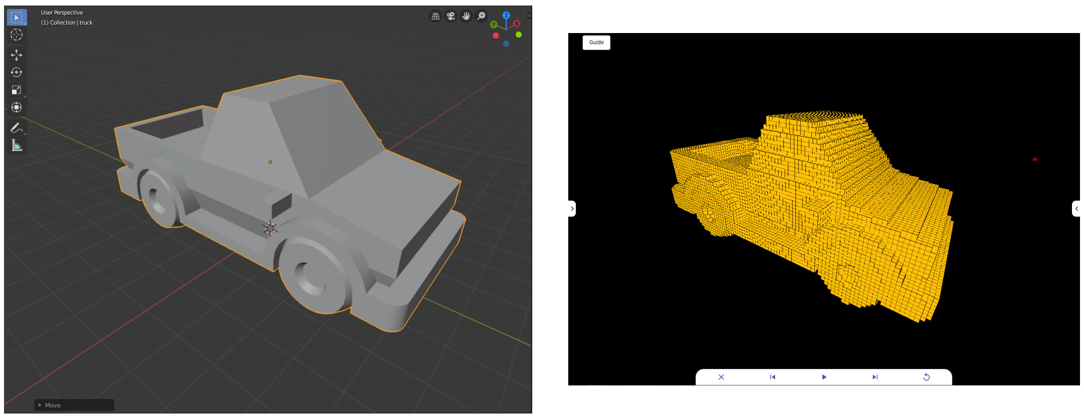


Figure 5.5: The input mesh (left), and its representation as a cluster of cells in the 3D Game of Life (right).

CHAPTER 6

Website Components

This chapter describes the actual building blocks of the Tour of Life platform, and how it was set up, as well as what technologies were used to bring it all together.

6.1 Node.js & NPM

Node.js is an open-source, cross-platform runtime environment for the execution of JavaScript code outside the browser. According to the Node.js website [17], it is built on Chrome's V8 JavaScript engine. It mostly finds use in building back-end services, or APIs, for client-side applications. It was chosen for this project because of prior knowledge and familiarity with the environment, in addition to its ease of use and scalability. Additionally, it boasts one of the largest collections of available open-source libraries [25].

Paired with Node.js is the package management system called NPM [18]. It was created in 2009 as an open-source way to help JavaScript developers easily share packaged code modules. It stands for Node Package Management, and it is the default package manager for Node.js. It includes a command-line interface, with which a huge repository of mostly free packages or modules can easily be accessed and adapted for any application. Instead of the developer having to write everything themselves, NPM can be used to quickly and easily implement existing code modules into a project [26]. NPM is automatically installed with every installation of Node.js and it was used for dependency management in this project.

6.2 TypeScript Programming Language

TypeScript is a strict superset of the JavaScript programming language, developed by Microsoft. According to the TypeScript website [24], it is a "strongly typed programming language that builds on JavaScript, giving you better tooling at any scale". Whereas JavaScript is a dynamic language, where some errors can go undetected all the way up until run-time with a browser crashing, TypeScript prevents this from happening, by extending JavaScript with types. In this case, TypeScript behaves like a compile language, with JavaScript as the compilation target. This means that with the TypeScript compiler, a TS file gets transformed into standard JavaScript [23].
The primary goal of TypeScript is to enable static typing, so variables can be annotated directly with static types. Custom types and interfaces are also possible. Strongly typed code means auto-complete is available everywhere in the project. This, along with a familiarity with TypeScript from past university courses, led to its use in this project.

6.3 Angular Website Framework

The Angular [3] framework was mainly chosen for this project because of prior experience with it. It is a TypeScript-based framework, developed by Google, for the building of user interfaces. When first generated, the Angular project comes out of the box, preconfigured with a testing framework, routing, etc. At its core, Angular is a component-based UI library [1]. Components are reusable parts of a website, that have been, among other things, used to display the simulation window within the Tour of Life lessons in this project. Angular also offers conditional logic or the ability to loop over iterable values. Common functionality and data can be shared between components using a service, although this feature has not been used in this project. To design the user interface, the Angular material [2] design components have been used.

6.4 Three.js WebGL API

In order to render the actual 2D and 3D graphics, a JavaScript library/API called Three.js was used. It uses WebGL to render GPU-accelerated computer graphics and animations without relying on third-party browser plugins [20]. It works by generating a canvas element inside the document object model, in which it renders specific geometry or meshes. In the case of this project, plane geometry was used to render the simple 2D cells, and the so-called box geometry was used for 3D cubes. It is very easy to manage the camera controls thanks to the provided TrackballControls object. This holds the configuration and control scheme for the camera movement, which can be adjusted at will. A scene object holds all the geometry or meshes that are to be rendered. The rendering itself happens within an animate loop, which runs continuously, and is framerate-independent.

6.5 GitHub Version Control

GitHub is arguably the most popular provider of internet hosting for software development and version control [10]. It is based on Git, which is a software for tracking changes in a set of files, mainly used by programmers for collaborative development. Git & GitHub were used for this project as it is an industry-standard, and it has been used with many previous projects. As no collaborative development was done in this project, changes have simply been committed to the main branch of the repository every time, and no branching architecture was needed. If any feature has not been fully implemented, the necessary additions were marked with a "TODO" comment inside the commit message. No versioning or branch structure was used.

6.6 GitHub Pages Deployment

GitHub Pages allows every owner of a GitHub account to publish a website, built from a select repository [9]. This option is available to free users of GitHub only if the repository in question is public. GitHub Pages generally requires no additional setup, and since the project was already using GitHub for version control, using Pages was the easiest path to publishing the website, without having to rent server resources and buy a domain. The website is served from a personal URL, tied to the user account.

CHAPTER 7
Results

This chapter serves as a look back at the development process, in order to outline some of the aspects of the project that might require more work, and/or could be changed or improved in the future. The goal, to create an interactive learning experience for Conway's Life, together with two separate sandboxes to allow for user experimentation, has been reached. The ground rules set up at the beginning of the thesis were all kept in mind when designing the platform. The lessons are designed to help even laypeople to understand the core concept of the website, the sandboxes are highly interactive, with many options to customize, and website features are explained either directly or inside the guide. The website is live and fully web-based, so no third-party add-ons or software is needed, and an intuitive way to visualize both the 2D and 3D sandbox has been implemented. The platform in its current form is a solid foundation, upon which improvements can be made, and future features added.

7.1 Web App Performance

The bulk of the web app has been developed and tested on a locally hosted, Angular dev server, which was run on a computer with the following relevant specifications:

- **CPU:** Intel i7-6700K, 4 GHz, quad-core
- **GPU:** Nvidia GTX 1070, 8 GB GDDR5 VRAM
- **RAM:** 16 GB 2133 MHz

To evaluate performance in highly populated scenarios, a test was performed, using a modified version of the algorithm, where the number of cells necessary for a cell birth was set to one, and the remaining rules had been adjusted so that no cell would ever die. This setup results in the continuous growth of a pattern with 4 living cells in a $2 * 2$ neighborhood.
It was observed, that on the machine described above, the simulation with its default speed settings and outside of prediction mode, would start slowing down slightly once around 10,000 living cells were observed. At around the 25,000 mark, the simulation would need one second to complete a step in the evolution, which is an over three-times increase in time needed, compared to the initial 0.3 seconds. At 30,000 cells, the web-page starts becoming unresponsive, and a refresh is needed.
An idea for speeding up the simulation has been developed, which could be implemented in a future iteration of the application. It will be discussed further in Chapter 8, Section 2.

7.2 Multi-platform Use

The web application was primarily developed for use on a standard computer/laptop, controlled via an external mouse, instead of a trackpad. Especially as a learning tool, with the potential to be presented in schools, a computer was the obvious target platform of choice. The advanced features and functionalities, like the painting of the cells and special camera controls, are lost if accessing the website using a smartphone or tablet, without an external pointer device and keyboard. The Tour of Life portion of the website, containing the actual lessons was also designed to display multiple lessons per chapter, rendering it unusable on mobile devices in its current form. The options menu in its current configuration also is not compatible with a mobile view, as it was designed to work alongside the render window, not on top of it.

7.3 Class Structure

From the very start of the project, it was questionable whether the 3D simulator would be included in the final version of the web-app. The main focus was proper implementation of the original Game of Life, with a high degree of interactivity and customizability. Because this function was, thanks to good time management and task division, completed rather on pace, and the lessons, due to the good foundation of the 2D simulator, were not too difficult to implement either, it was decided that the 3D Game of Life will indeed be part of the Tour of Life suite.
Because an existing foundation was already in place, which was being used in the lessons as well, it has been decided, that instead of reworking the 2D code-base to support an expansion into 3D, its functionality would simply be copied, with the necessary reworks to support the extra dimension added later. This is ultimately not a state-of-the-art approach, and should the application grow in the future, the class structure would inevitably need to be reworked to use interfaces and prevent duplicate code.

7.4 Achievements

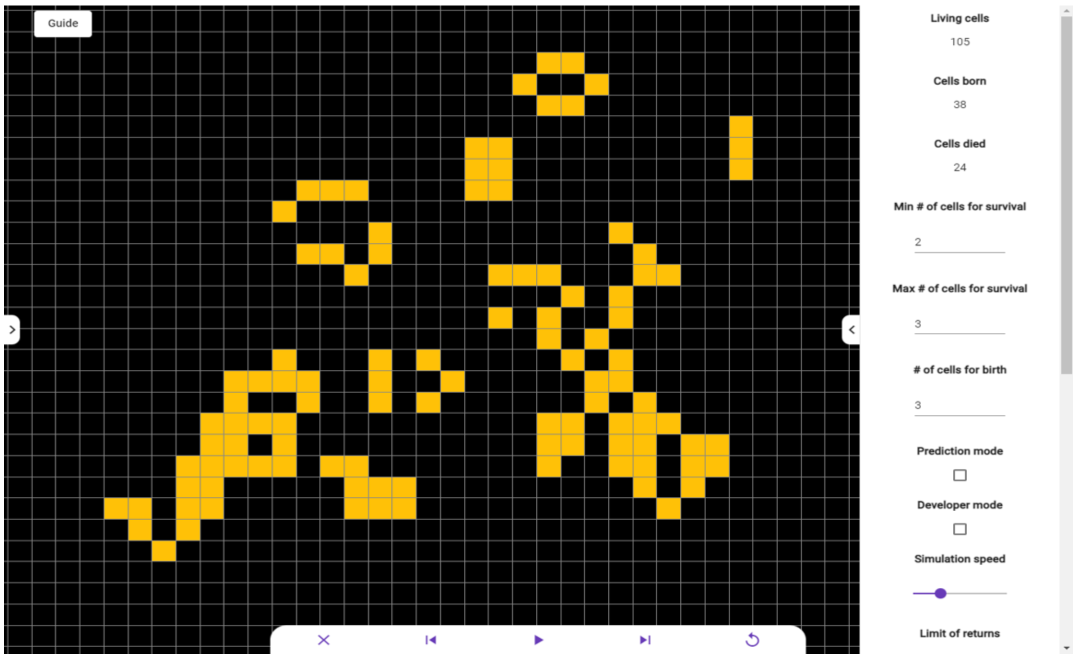
This final section of the reflection chapter serves as a way to determine, if the goals/criteria laid out for this project in Chapter 1, Section 2 have actually been reached, and if so, how. What follows is a repeat of the list of goals, as well as a description on how they have been integrated into the project:

- **Approachable to laypeople:** Thanks to the interactive learning section of the website, called the Tour of Life, users have the opportunity to get acquainted with the concept of a cellular automaton, the Game of Life, and later even interact with concepts of computer logic and Turing machines. This is done via lessons which start out very simple, but increase in complexity as the user works through them. Every new concept is thoroughly explained and the user has freedom to experiment with the presented lesson at will.
- **Interactivity:** Every lesson that contains a pattern also allows the user to either interact with the lesson directly, or copy the pattern in the general Game of Life sandbox section, where it can be properly experimented with. Additionally, exercises have been added, which task the user with completing a pattern based on a description, to keep them engaged.
- **Proper documentation:** In addition to the annotated playback controls, both sandboxes contain a guide, which explains the less intuitive features and options available to the users. The Tour of Life learning experience contains an explanation page, meant to prepare the users for the lesson format, as well as teaching them how to interact with the page. Some chapters contain hints and useful tips regarding the lessons. Every lesson also provides a link to an external resource specific to the lesson, where the user can find out more about the presented concept/pattern.
- **Self-containment:** The Tour of Life has been developed as a web application. This means, that it is accessible by anyone with an internet connection and a browser. It has not been properly adapted to a mobile view (yet), but any computer will be able to run the application without any third-party software or add-ons. The website is also publicly available, without any need for a private web server setup.
- **Customizability:** The look and feel of the sandboxes is highly customizable. Users can edit the underlying algorithm, speed of execution, and even the colors and presentation of the simulation. For pre-designed parts of the experience, accessibility was kept in mind by creating a base color palette easily usable even by people with different kinds of color-blindnesses.

- **Visualization:** An intuitive and simple way of interacting with the 3D rendition of the Game of Life has been implemented, which relies almost solely on the computer mouse. This helps with navigation in the 3D space, as well as with the creation/editing of patterns. The additional dimension has been kept in mind when designing the prediction mode, to make it as clear as possible.

The Tour of Life fulfills the goals set in the beginning of this thesis. Users can learn about the potential of the Game of Life, and what it means to be Turing complete, through the use of interactive elements and theoretical lessons, as well es experiment freely with the algorithm of Life inside multiple sandboxes. We propose that this makes it a worthy addition to the existing online resources on Conway's Game of Life.

What follows are screenshots from the actual Tour of Life website. These depict the 3 main sections of the website, namely the 2D Game of Life sandbox (Fig. 7.1), the Tour of Life learning experience (Fig. 7.2) and the 3D Game of Life sandbox (Fig. 7.3).



Living cells
105

Cells born
38

Cells died
24

Min # of cells for survival
2

Max # of cells for survival
3

# of cells for birth
3

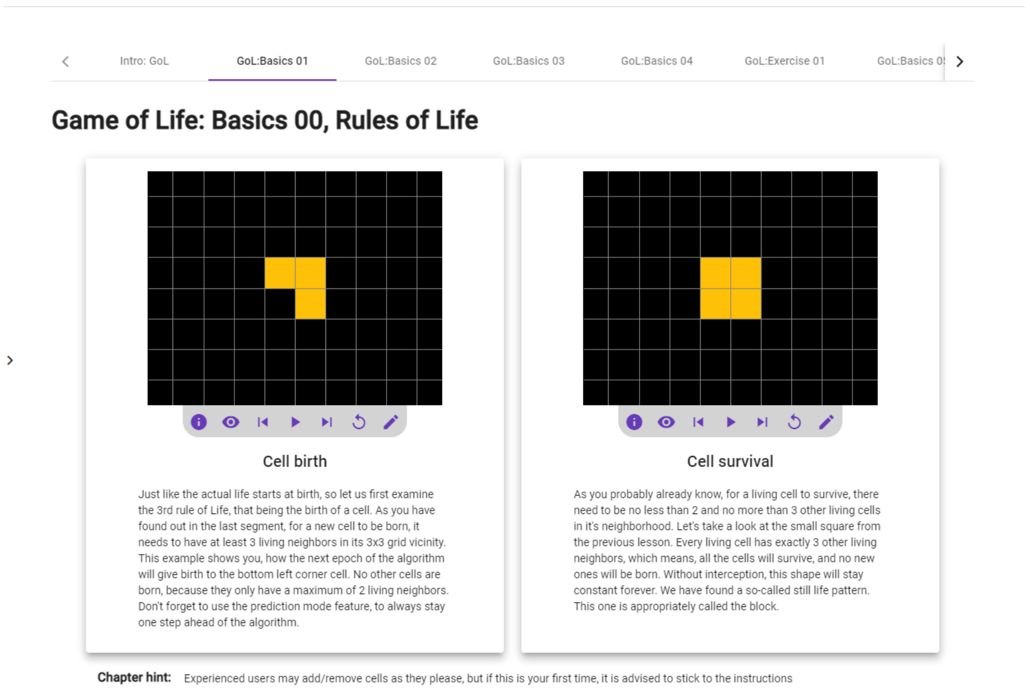
Prediction mode

Developer mode

Simulation speed

Limit of returns

Figure 7.1: A screenshot of the 2D Game of Life simulator with an expanded options menu on the right, playback controls at the bottom and a guide button in the top left corner of the screen.



Game of Life: Basics 00, Rules of Life

Cell birth

Just like the actual life starts at birth, so let us first examine the 3rd rule of Life, that being the birth of a cell. As you have found out in the last segment, for a new cell to be born, it needs to have at least 3 living neighbors in its 3x3 grid vicinity. This example shows you, how the next epoch of the algorithm will give birth to the bottom left corner cell. No other cells are born, because they only have a maximum of 2 living neighbors. Don't forget to use the prediction mode feature, to always stay one step ahead of the algorithm.

Cell survival

As you probably already know, for a living cell to survive, there need to be no less than 2 and no more than 3 other living cells in it's neighborhood. Let's take a look at the small square from the previous lesson. Every living cell has exactly 3 other living neighbors, which means, all the cells will survive, and no new ones will be born. Without interception, this shape will stay constant forever. We have found a so-called still life pattern. This one is appropriately called the block.

Chapter hint: Experienced users may add/remove cells as they please, but if this is your first time, it is advised to stick to the instructions

Figure 7.2: A screenshot of the Tour of Life chapter layout. It shows the first chapter after the introduction, that's supposed to teach the user the basics of the Game of Life algorithm.

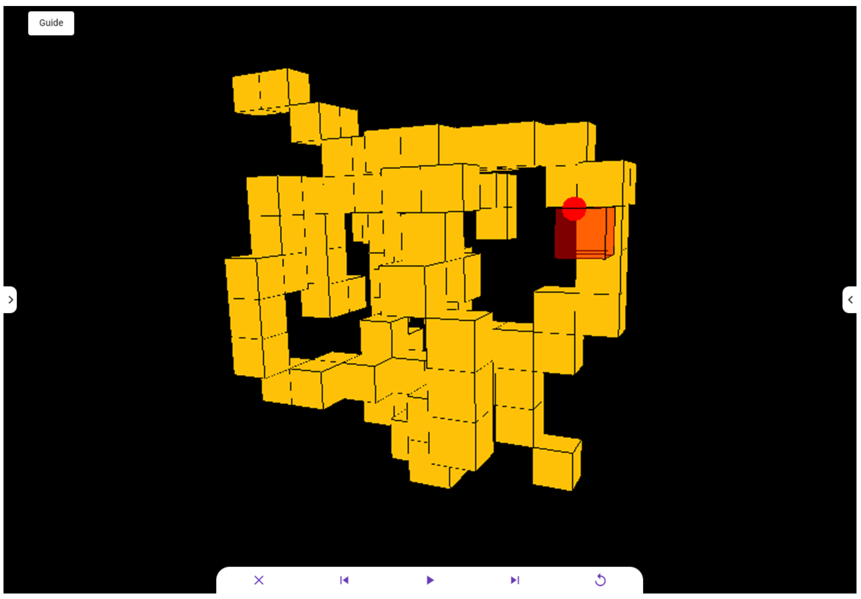


Figure 7.3: A screenshot of the 3D Game of Life simulator, including painted cells (yellow) and the 3D cursor (red).

CHAPTER 8

Conclusion

This last chapter serves to conclude the thesis with a brief summary of the services the platform provides, and also to present potential future developments of the project, based on experiences collected during development, and feedback from actual users.

8.1 Original Vision

Originally, this bachelors thesis was supposed to be a mostly theoretical study on the Game of Life cellular automaton, the idea of how simple rules can give rise to very complex results, and the algorithms unintentional Turing-completeness. Before even entering the research phase of the project, when brainstorming initial ideas for this thesis, the original vision of this project was to explore the potential of the Turing-completeness of Life. Experiment with cell configurations for making simple calculations and even exploring how the actual computer as we know it, would work inside of Life. During the initial research, however, no convenient and reliable resources were found online, which would understandably and interactively teach the very basic concepts behind the Game of Life. This was seen as an opportunity to contribute, and potentially even help others with their own research in the future. The concepts and ideas listed here have in some capacity still found their way into the lesson section of the web-page, as well as the thesis itself; yet this content is less detailed, as the focus of this project has shifted slightly, towards the broad basics of the Game of Life.
This type of research focused thesis as originally envisioned would be a better fit for a separate, maybe even a follow-up paper, including an add-on to the Tour of Life, focusing more on the Turing-completeness of Life, instead of the very basics as is currently the case.

8.2 Summary

Even though the project has slightly changed direction in its initial planning stage, its evolution is arguably even more interesting and useful to the average user. It will ideally help with further research and studies of the Game of Life algorithm and its potential since the Tour of Life enables even the more inexperienced users to get into the complexities of the Game of Life more easily.

As a first step, and to properly understand the underlying topic of this project, the original Game of Life has been reproduced. To fit with the theme of customizability, multiple customization and interactivity options have been added.
The core idea of this project has been realized in the form of a learning section, called the Tour of Life. Using a number of interactive lessons and exercises, this part of the experience is supposed to help the user understand the very basics of the Game of Life, as well as the more advanced concepts of computer logic, Turing machines, and recursivity. Lastly, a 3D version of the Game of Life has been developed. This section required an in-depth analysis of the concepts of visualization in 3D space, as well as intuitive interaction with objects inside it.

Hopefully, after presenting this project to a wider audience after its submission, it can help people willing to learn more about John H. Conway and his Game of Life. There are still undiscovered patterns and potential usages for the automaton, and if this project helps introduce more people to the exciting world of Life, then it has served its purpose.

My prototype, described throughout this thesis can be accessed via the following link: `https://11811341.github.io/tour-of-life/` [21].

8.3 Future Development

All planned features have been successfully implemented. The most important next step in the development of the website is collecting feedback from users. This would be done in the form of a simple user study, where users would be given a questionnaire, and would consequently be asked to rate their experience with the website. The questions would range from usability and understandability of the presented content to the website design. A section for suggestions would also be included. Based on this feedback, features can be added, and changes made to the application, to make it even more user-friendly and turn it into a more complete experience.
The two aspects of the application that arguably need further development are the performance of the simulators, and the class structure, as has already been outlined in the reflection Chapter 7.

An idea to speed-up the calculation of cell/pattern evolution is to recursively examine only the most recently added cells and their potential neighbors, instead of looking at every living cell at every step of the simulation. The existing approach is more than usable at the scope at which the lessons operate, or a user would logically work at, where not many cells are alive at any given time. However, especially since the algorithm can be altered, patterns can grow unpredictably large, and the website needs to be able to support this growth, to ensure proper usability.

The solution to the class structure has already been hinted at. It would involve refactoring the existing classes, and creating an interface that could be implemented for both the 2D and 3D versions of the simulator, where the 3D version would expand upon the functionality of the 2D version. The way classes exist now, a lot of duplicate code is present, which is not ideal for a publicly available service.

An interesting addition might be the option for users to create their own lessons for the Tour of Life. This would include a simple input form, with a small simulator window, where a user could input a custom pattern, and a description of the lesson. This would then be automatically added to the users locally stored version of the lesson lexicon. Additionally, instead of uploading the custom lesson to a shared database of user-created lessons, a special string code could be generated, which when shared with others, would reproduce the lesson on a different instance of the website and add it to any other users lesson lexicon.

Hopefully, this project will serve as a step towards a better understanding of the Game of Life, and help more people get properly acquainted with this topic. At the same time, it can be seen as a push towards more interactive learning experiences, that use the web medium to educate and to clarify complex topics using modern teaching methods.

Bibliography

[1] Angular in 100 seconds. https://www.youtube.com/watch?v=Ata9cSC2WpM&ab_channel=Fireship. Last accessed: 11.01.2022.

[2] Angular material ui component library. https://material.angular.io/. Last accessed: 11.01.2022.

[3] Angular website. https://angular.io/. Last accessed: 11.01.2022.

[4] Blinker pattern. https://conwaylife.com/wiki/Blinker. Last accessed: 06.02.2022.

[5] Chaos and fractals, conway's game of life. http://pi.math.cornell.edu/~lipa/mec/lesson6.html. Last accessed: 06.02.2022.

[6] Creetah game of life 3d app. http://www.creetah.com/game-of-life/. Last accessed: 06.02.2022.

[7] cuug game of life. http://www.cuug.ab.ca/dewara/life/life.html. Last accessed: 06.02.2022.

[8] decode game of life. https://www.dcode.fr/game-of-life. Last accessed: 06.02.2022.

[9] Github pages website. https://pages.github.com/. Last accessed: 12.01.2022.

[10] Github website. https://github.com/. Last accessed: 11.01.2022.

[11] Gosper glider gun. https://www.conwaylife.com/wiki/Gosper_glider_gun. Last accessed: 17.01.2022.

[12] Importance of web interactivity. https://www.nexuswebsites.co.uk/importance-of-web-interactivity/. Last accessed: 18.01.2022.

[13] Intellij idea website. https://www.jetbrains.com/idea/. Last accessed: 17.02.2022.

[14] Interactive teaching styles used in the classroom. https://resilienteducator.com/classroom-resources/5-interactive-teaching-styles-2/. Last accessed: 06.02.2022.

[15] Magic: The gathering website. https://magic.wizards.com/en. Last accessed: 06.02.2022.

[16] Minecraft website. https://www.minecraft.net/de-de. Last accessed: 06.02.2022.

[17] Node.js website. https://nodejs.org/en. Last accessed: 11.01.2022.

[18] Npm website. https://www.npmjs.com/. Last accessed: 11.01.2022.

[19] Rithmschool game of life 3d. https://rithmschool.github.io/game-of-life-3d/. Last accessed: 06.02.2022.

[20] Three.js website. https://threejs.org/. Last accessed: 11.01.2022.

[21] Tour of life website. https://11811341.github.io/tour-of-life/. Last accessed: 17.02.2022.

[22] Turing completeness, cs390, spring 2022. https://www.cs.odu.edu/~zeil/cs390/latest/Public/turing-complete/index.html. Last accessed: 17.01.2022.

[23] Typescript in 100 seconds. https://www.youtube.com/watch?v=zQnBQ4tB3ZA&ab_channel=Fireship. Last accessed: 11.01.2022.

[24] Typescript website. https://www.typescriptlang.org/. Last accessed: 11.01.2022.

[25] What is node.js? https://www.youtube.com/watch?v=uVwtVBpw7RQ&ab_channel=ProgrammingwithMosh. Last accessed: 11.01.2022.

[26] What is npm, and why do we need it? | tutorial for beginners. https://www.youtube.com/watch?v=P3aKRdUyr0s&t=137s&ab_channel=CoderCoder. Last accessed: 11.01.2022.

[27] Wikipedia, conway's game of life. https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life. Last accessed: 06.02.2022.

[28] B. Abykanova, D. Sadirbekova, Z. Sardarova, A. K. Khairzhanova, G. S. Mustagaliyeva, O. D. Tabyldieva, E. D. Abdol, and T. Bainazarova. Interactive teaching methods as pedagogical innovation. Technical report, BBRC, 2021.

[29] C. Bays. Candidates for the game of life in three dimensions. Technical report, Department of Computer Science, University of South Carolina, 1987.

[30] R. Beaulieu and E. Coy. 3d game of life. `https://rbeaulieu.github.io/3DGameOfLife`. Last accessed: 11.01.2022.

[31] J. Etchemendy and J. Barwise. *Turing's World*. CSLI Publications, 1993.

[32] R. W. Gosper. Exploiting regularities in large cellular spaces. Technical report, Symbolics Inc., 1984.

[33] N. Johnston. Lifewiki. `https://www.conwaylife.com/wiki`. Last accessed: 11.01.2022.

[34] N. Loizeau. Building a computer in conway's game of life. `https://www.nicolasloizeau.com/gol-computer`. Last accessed: 11.01.2022.

[35] E. Martin. Game of life. `https://playgameoflife.com/`. Last accessed: 11.01.2022.

[36] D. Nichols. Coloring for colorblindness. `https://davidmathlogic.com/colorblind/#%23D81B60-%231E88E5-%23FFC107-%23004D40`. Last accessed: 11.01.2022.

[37] W. Poundstone. *The Recursive Universe: Cosmic Complexity and the Limits of Scientific Knowledge*. Contemporary Books, 1985.

[38] S. Senthamarai. Interactive teaching strategies. Technical report, Department of Education, CK College of Education, 2018.

[39] A. Trevorrow and T. Rokicki. Golly. `http://golly.sourceforge.net/`. Last accessed: 11.01.2022.