LUISS Guido Carli University

Degree Program in Management and Computer Science

Bachelor Thesis

Spring 2025

Thesis Title

Creating a Simulation of Life and its Interaction with the Environment to Induce a Complete Cycle of Unsupervised Self-development

Candidate

Savva Krasnokutskii (281441)

Supervisor

Professor Alessio Martino

Abstract

This thesis aims to create a simulation of life and its interaction with the environment to induce a complete cycle of unsupervised self-development by the means of expanding upon the classical cellular automaton model by simulating fundamental evolutionary principles via addition of genome based sexual reproduction to each ‘cell’ of an organism. Each cell in the proposed simulation will have an arbitrary long genome which would define the way a ‘cell’ would interact with the outside environment, how it would react to its surroundings, compete for resources, interact with other cells and reproduce. The objective of this investigation is to implement the aforementioned simulation and attempt to analyze its underlying mechanisms, placing particular emphasis on the notion of developmental continuity.

Table of Contents

1. Introduction

The fundamental goal of machine learning, the Philosopher's stone of this discipline, lies in the creation of a self improving model, capable of single handedly, without human intervention constructing better versions of itself. To the best of my knowledge, so far, no such model exists (or at least it is not publicly available).

My hypothesis is that the creation of such a model could be made possible by shifting the approach we use, via changing the paradigm, where instead of teaching the model (be so directly or indirectly) we shift our focus from model oriented approaches, to the environment oriented ones.

Let us look at the existing example, humans. Humans are an intelligent species, capable of self-development. From the stone tools and silicon weapons, we have made it to space, to the Moon, our spacecraft has reached interstellar space. But let us look at how we appeared as we are. The appearance of the human species was a result of a long interaction. At first an interaction within the environment which gave birth to life. Later interaction of life with the same environment, the process of evolution.

In my study I will try to solve the problem of development of self learning and self improving models, by turning to cyber biology and cellular automation fields. My aim lies in the creation of a simulation of life and its interaction with the environment to induce the complete and sustaining cycle of self-development.

It is important to note that the key concept would be the interaction between creatures and environment, as specified in the title of the thesis. Here I am only stating the necessity of greater emphasis on working with the environment. It is imperative to have both parts of the simulation to be well developed and capable of working together, in order to even have an opportunity to go beyond the scope of classical cyber biological methodology and the tools that the discipline offers.

2. Theoretical Background

In this section our aim would be to formalize the notions brought up in the introductory sections and briefly document preliminary research conducted, as well as to document early hypotheses regarding the expected performance of the project's prototype.

2.1 Introduction to Cellular Automatons

For the sake of this thesis report let us use the Stanford definition of cellular automaton. Despite there exist many possible ways of defining it, the Stanford definition is the most definitive, elegant, as well as more convenient in regards to the topic of our report.

The definition provided by Stanford is as follows:

"Cellular Automatons are discrete abstract computational systems [...] typically spatially and temporarily discrete: composed of a finite or denumerable set of homogeneous, simple units, the atoms or cells. At each time unit, the cells instantiate one of a final set of states. They involve parallel at discrete time steps, following state update functions or dynamical transition rules: the update of a cell state obtains by taking into account the states of a cell in a local neighbourhood."

From the above definition, we can also conclude that cellular automatons can be represented by abstract systems: it is not necessary for such a simulation to represent something practical. It can be defined in mathematical terms, using matrixes or graphs from graph theory, later in more complex cases. This does not however mean that the physical structures can not implement them.

Thirdly and lastly, an important corollary of such a definition would be the possibility to define such rules for an automaton, that it would meet the requirements to be considered an emulator of a Turing-complete machine (therefore, by the Church-Turing thesis, being capable of computing everything that is computable). Again that would require a very special subset of all possible rules and it would not look like a classical Turning machine, but the mathematically defined system is still able to produce a sufficiently powerful mathematical

toolset to make such operations possible. Yet since this is not directly related to the topic of the thesis I will omit proof of this statement from the report, for more information on the topic the interested reader is referred to the bibliography section at the end of the report.
Now that we have a more formal understanding of what a cellular automaton is, we will be able to translate our goals into a more rigorous and formal language. First of all, let us set a strategic goal of defining what a self-developing system is in the context of a classical cellular automaton. That is, let us define a classical cellular automaton as a two dimensional array of numbers, representing *states*. The states are updated each *turn*, where a turn is defined as a function applied to the state matrix in order to get the next result. It is very important to note that when it comes to classical automatons, the update function is not dynamic. The entirety of the state matrix is being updated at the same time where the values of the surrounding states are calculated based solely on the previous state matrix.

2.1.1 Discrete vs. Continuous Automatons

In order to make the above points clearer, let me give a trivial example. Suppose that the state of the field is defined by the opposite of the value of the field to the right of it. Let us also assume that the rightmost field is connected to the leftmost field. Let us say that at turn one the system looked like this “001”, then it might seem logical to conclude that at the next state would look like “100”, but in fact this is wrong since we are doing the calculation instantly and therefore the field would look like “101”, since first element at the time of the update is still zero.

That is a fundamental distinction between dynamically updating automatons (100) and discretely updating ones (101). From now on let us define *discrete automatons* as the ones updated using previous state matrix as reference and *dynamic* ones as ones using current state matrix as reference. Note that in case of dynamic cellular automatons there can be multiple algorithms to determine the possible order of operations, whilst on discrete ones there exists only one. Let us also note that the classical automatons are almost always discrete.

2.1.2 Dimensionality of an Automaton

When we were defining cellular automatons we used two dimensional matrices to describe the state in which they operate. Yet since we know cellular automata as an abstract concept we have no difficulty in extrapolating the aforementioned definition to tensors of any rank. Speaking in Pythonic language we are not necessarily stuck working with 2D arrays, an automaton can be defined in 1D, 3D up to N-dimensions, where N is arbitrarily large.

An example of a 2D automaton could be the famous "Conway's game of life" (CGL), a classical member of the genre. It is a simple discrete automaton with rules concerning the amount of neighbours in the "alive" state. An example of a 3D cellular automaton can be found in the Appendix as a link to my Github repository. A classical automaton concept implies it being 2D though.

2.1.3 Update Principles

A cellular automaton rules can vary from case to case, or be a combination of a few principles we are about to talk about. The most basic rule would be to give a cell with coordinates (X, Y) to update depending on the number of cells in state 1. An example would be to take a cell above, below, to the left, to the right and all diagonal cells, count the number of cells in state 1. If this number is in range $(0, a)$ - kill cell, (a, b) - cell stays alive or appears if it was empty previously, $(b, 8)$ - cell dies.

Again, there are thousands of potential variations: you can choose any region in which to count the states. That region can be predefined, as well as dynamic, changing its shape every turn also depending on the states previously assigned to it, gerrymandering of some sort. This would create very complex simulations, but right now is not the point of our investigation, it is however worth noting that such operations are legal, possible and can/should be well defined.

Turning back to the rules, counting the states in the region is by far not the only way to make a decision, you can assign dynamically changing weight coefficients to the states and apply different arithmetic and logical operations to their states in order to get the final result. For example we might require the state on the upper right diagonal to be equal to the state on the

lower left diagonal and there to be at least 3 cells in state 0 in the direct proximity to the cell in order to reverse its state else we do nothing.

Classical automatons follow the "make everything as simple as possible" paradigm and therefore rarely practice the additions described above, since despite they seem to overcomplicate the model greatly, they might end up being reducible and therefore having practically little effect on the simulation.

2.1.4 Multiple States

A state does not have to be a boolean variable itself. It can be represented by a subset of integers, or more broadly speaking any set that is ordered and finite. Expanding a definition of the state would allow to make the rules more complicated and more complex. Yet all classical automatons are using only 2 states. Often labeled in the corresponding literature, as alive and dead.

2.1.5 On the Notion of Reducibility

When working with different systems it soon becomes obvious that a bigger formula does not necessarily mean more complex. For example, let us define a given function $f(x)$ as $f(x) = \frac{x^2+1}{x^2+1} + x! - x!$. This function seems to be very complex, yet is completely analogous to $f(x) = 1$. By drawing this analogy, I want to express the fact that the same thing can be applied to cellular automatons, the rules might end up reducing each other and in the end we might end up in a situation where the model would act like a simpler model, but due to us being overloaded with different factors we would have no capacity in any way shape of form to understand it.

Therefore to avoid creating "reducible" simulations it would be very important to adopt and follow, as a simple but not simpler motto. The sheet existence of a possibility to complexify the ruleset, is not sufficient reason for its implementation. Before implementing we would need to think about what the implications and consequences of such a decision would be and only proceed with the implementation further knowing that without it the desired ruleset is unexpressible (or very difficult to express).

2.1.6 On Loops and Graphs

As we previously saw, one fundamental problem of classical cellular automatons are borders. An angular tile only has 4 neighbours, whilst central have nine, that makes the edges behave differently throughout the simulation. That breaks the common convention of classical cellular automatons: the simulation behaves differently at different points on the graph, and whilst officially by the word of the definition that is not forbidden, it is considered a good style to have the simulation playout without borders.

From a programming standpoint simulating an infinite field is impossible and implausible due to the eventual memory over usage (2 gliders going in the opposite directions would eventually make the system run out of memory). Therefore a solution is either to put up borders, which assume state 0 by default and can not change, or to use a loop.

A loop is a simple concept, where pixel A is linked to pixel B despite not bordering each other, it is usually done vertically and horizontally. Connecting rightmost cells to leftmost cells and upmost cells to downmost cells. Such loops are quite convenient and simple, yet they make the system less clear for the outside observer.

Loops can be developed further however. For example we can use a graph to define the connecting relationship between cells. This would permit us to simulate not only more sophisticated loops, but also tilings. Now for example we would be able to cover a sphere with 2D tiles (more on that later) and have a simulation play out on it.

Note that besides squares we can also use hexes, pentagons and many other shapes in any combination. A graph would permit us to do that.

It is worth noting that no stochastic operators are permitted in cellular automation. There must exist a definitive ruleset, which given the same initial state does not change when experiment is repeated. This is a very important feature of classical automatons.

2.2 Automaton Convergence

In mathematical analysis and calculus one of the most important tools designed to study functions is the limit. We try to understand what value the function assumes at a point X or infinity: will it converge or diverge? Since we want to define a continuous self-developing system we would also need to define a limit and convergence for automatons. We should try to first heuristically study and later fundamentally understand the behaviour of such a system. What does it mean to develop? What does it mean to converge? Does a Conway's glider converge or not? To understand that let us define a concept of repeatability.

From now on assume, unless specified otherwise, that we are dealing with classical cellular automata.

Upon first visual inspections of cellular automatons with different rule sets I was able to derive 6 potential endings. The first one being total extinction, the entire field is empty, no cell left alive. Second one is complete filling - all spaces covered with cells, no cells die, nothing happens. The third one is some sort of static continuous oscillation where there are certain configurations of cells that alternate one another. The Fourth one is also oscillating, but moving.

For example, like a glider. When cells also alternate a series of configurations, but are not static with respect to (X, Y) coordinates. Fifth one (hypothetical) is when the repetitions happen all around the experimentation plane. And six one (also hypothetical) is when we encounter types of motion close to stochastic, without noticeable patterns. The last two were not noticed by me when experimenting with CGL and were only noticed by me in more complicated simulation.

In order to visually enhance my description, below I attach examples of states (three, four, five and six) on Diagrams 1, 2, 3, 4 respectively. Note that states 1 and 2 would not be shows, due to their trivial nature.

The observations are certainly interesting, but require a much more rigorous definition and particular simplification. Let us try to use a concept of repeatability for that. For our definition let us define some variables. There exists a classical cellular automaton C, it has a border, it follows a particular static ruleset, it has some starting configuration S0 (zero indexed) it was simulated for N rounds, where N is arbitrarily large.

Diagram 1 (Depicting state 3)

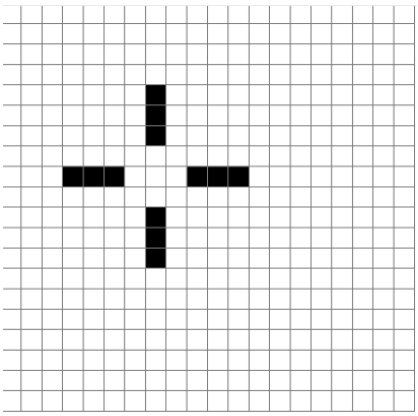


Diagram 2 (Depicting state 4)

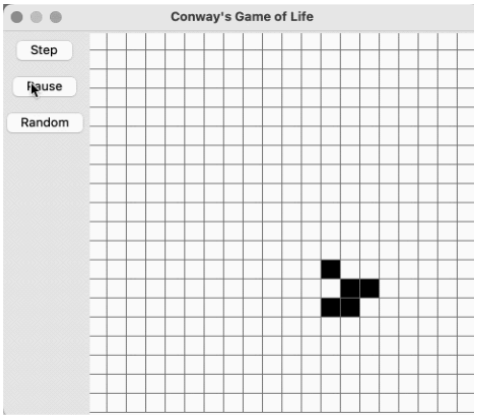


Diagram 1 (Depicting state 5)

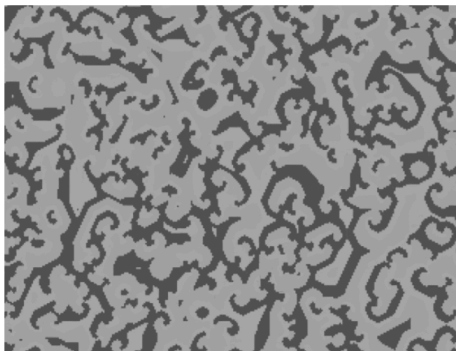
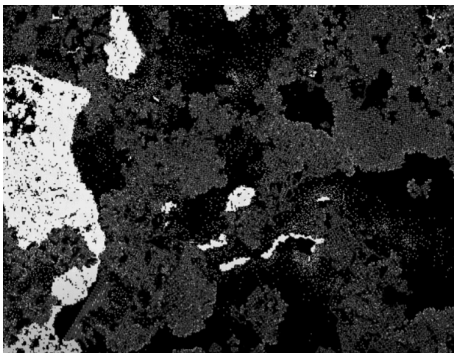


Diagram 2 (Depicting state 6)



Definition attempt 1

A cellular automaton is considered to be looping when there exist a state S_n such that, for every element of S_n, $S_n(x, y) = S_0(x, y)$

This definition seems solid, yet it is not rigorous enough. Indeed assume we have a case 3 looping cellular automaton, then add a configuration that leads to case 0 in 1 turn far away from it. Such configuration would disappear, but according to our definition the system would not loop, because $S_0 \neq any\ S_n$. Also periodic is a better word as compared to looping.

Definition attempt 2

A cellular automaton is considered to be periodic when there exists such a pair of states S_N and S_K such that $S_N(x, y) = S_K(x, y)$ for all x and y, where $N \neq K$, with periodicity of $|N - K|$.

Corollary 1

All automatons can be considered to have a period of 0.

Corollary 2

An automaton with a maximum period of 0 is non periodic.

This definition is better, but according to it a glider is not periodic, since its states never repeat. Therefore in order for a definition to be truly rigorous and all encompassing the positions should not be considered relative to the coordinates but relative to the states. For that we need to define an operation of shifting, where we move the entire simulation n pixel up/down and k pixel left/right.

Auxiliary definition

Given a state matrix S a shifted matrix $S^{*\{n,k\}}$ is defined as $S\ *\ (x, y) := S(x + n,\ y + k)$, where n and k are constant for all values of x and y.

Definition attempt 3

A cellular automaton is considered to be periodic when there exists such a pair of states S_N and S_K such that $S_N(x, y) = S_K{}^{*\{n,k\}}(x, y)$ for all x and y, where $N < K$.

Note that N is different from n and K is different from k: N and K define turns, whilst n and k define shifts.

The following definition can be extrapolated to more than two dimensions and non-binary states.

Theorem 1

A cellular automaton for which $S_N(x, y) = S_K{}^{*\{n,k\}}(x, y)$ is true, given $K - N = P$, it continues to be periodic with period P forever. The proof is trivial, since classical cellular automatons do not have stochastic factors after applying a change of state function the result that is yielded is always the same, therefore once periodic means always periodic.

This allows me to simplify the previous 6 case definitions I came up with. Cases 1 and 2 are simply 0-periodic; 3, 4 and 5 have N-th periodicity; whilst 6 is aperiodic. With this we are now able to understand that if we are interested in creating a self-developing system we ought to avoid any sort of periodicity. Because, as Theorem 1 states, once a period is always a perpetual period. Therefore a periodic state, like a glider or an object moving in place is a state analogical to death and therefore should not be considered.

Fundamentally speaking we were able to discover the two stages of any simulation. Period 1 (from S_0 to S_N) when initially placed objects dynamically interact, and another period S_N to S_K and till infinity when the simulation starts looping. We are therefore more interested in the S_0 to S_N state, we need to find an aperiodic automaton, for which there do not exist states S_N and S_K at which all cells share an identical structure, irrespective of coordinates. Note that having a border is an extremely important prerequisite, because otherwise you would need another definition, since an infinitely expanding simulation, even with a trivial ruleset, is hard to prove to be repetitive. Example making all fields with at least one active neighbour active as well, on an infinite scale would never loop with accordance to our definitions therefore we assume borders. Plus by assuming them, we are not losing much, since as we said there exists a lot of fundamental problems concerning practical computation of such a simulation.

Therefore our goal is to create an aperiodic simulation on a finite space, but since it might not be possible we might introduce stochastic factors deviating from a classical cellular automaton approach (keeping everything else intact), which would not permit the looping of the simulation.

Note that a question whether it is possible to create a continuous aperiodic classical cellular automaton is a big and interesting question by itself, and might be worth investigating in the future. Also, since computationally it is impossible to simulate true randomness, I came up with an interesting theorem.

Theorem 2

Every bounded cellular automaton, with non stochastic ruleset and finite number of states is bound to be looping.

Proof

Assume we have $n \times k$ tiles and each cell can have m states. In total there are $m^{n \cdot k}$ possible arrangements of the state matrix S. If we run a simulation for $m^{n \cdot k} + 1$ turns there bound to be repetitions and by Theorem 1. A single repetition implies periodicity.

Imagine proving it also for unbounded cases, since cellular automaton are also Turing complete: that can have some super interesting implications. However, back to the topic. Now that we are able to define what we mean by continuous, let us talk about self-development.

2.3 Self-development in Context of Automatons

Notion of self-development is clear in the field of Artificial Intelligence (AI), where given various tests (e.g., guessing a credit score, writing a sentence), given a second player that would not be able to identify an AI is being evaluated, and given some performance metrics (e.g., F1-score), we conclude that development was successful if such performance metrics tend to improve. Sometimes the AI is allowed to play against itself or is being controlled and taught by an independent set of examples.

So far, however, there has not been an AI capable of achieving complete self-development, that is, a cycle where AI continuously improves at a relatively steady rate without human intervention. Many models exhaust themselves, or get overfitted. Even state of the art Generative AI models (e.g., transformers) are not safe from these types of problems: Chat-GPT stagnated in its development and it is only capable of simulating thought and guessing next words, but unlike real human beings it does not have consciousness.

My thoughts on it are as follows: so far we were most likely unable to simulate (at least openly) an AI which has a real consciousness and is capable of thinking logically without being hard coded on how to do so and it is still significantly behind humans in many areas. However a potential fix to that would be turning our attention to nature. We humans were able to evolve as a thinking species capable of rapid self-development. Life as a whole evolved extremely rapidly, their universe in the process of self organisations moved from gas fields, to stars and planets. Slowly all the elements and molecules became a primitive cell without a nucleus. Life started and soon humans appeared, the universe therefore is a self-developing

system. We can see that it develops very rapidly and exponentially: it took Billions of years for planets to form, millions of years for life to evolve.

Human civilization had moved from the late neolithic age to the Roman empire over the course of only a few millennia. And less than one thousand years separate crusades and the first spacecraft landing on Venus. This made me think that we should focus on the evolutionary approach and creature environment interaction in order to later apply the knowledge acquired to design an AI with the ability to self-develop and self improve continuously.

But it is nevertheless important to define a development as a process with respect to cellular automatons, because greater complexity over time as definition is lacking.
All of the processes need a metric, a subjective concept such as development can be defined with a help of one. Because development is subjective the same process might seem like a development from one point of view, but like a stagnation or regress from another. When it comes to country development metrics like GDP or GDP per capita can be used, or even HDI. Yet to measure a development of a cellular automaton such metrics would be unusable.

Let us divide a problem into smaller steps, first let us bring up the notion of complexity. You can theoretically measure development of a simulation, by how complex it is, for example how many interactions happen and by some other metrics which potentially we would be able to define and analyze in a clearer way.

However complexity in fact does not always mean development, for example if we want a good shovel we want it to have a somewhat simple shape, making the shape more complex, filling the blade with composite holes and adding spikes would not make the shovel better.

This means that the development is primarily subjective with respect to the environment, since the point of life is to survive and reproduce. Obviously development and survivability are not the same concepts, for example some bacterias can survive and form colonies in the environments human will perish in, while being significantly less developed live form than humans, meaning that in the simulation we are making an evolutionary adaptability is an important and mandatory, but not the only criterion of success.

Therefore five development criterions I was able to derive are as follows:

1. Survivability: ability of a creature to survive in an environment
2. Adaptability: ability of a creature to adapt to the changing environment
3. Complexness: how complex the creature ruleset is
4. Complexness evolutionary dynamics: how does complexness change as creature adapts
5. Impactfulness: how strong is the creature's influence on the environment.

These rules are still broad, however they can apply better since objective metrics can be developed to measure them. Some concepts like “creatures” and “genome” might still be unclear, as when we were talking mainly about the classical cellular automatons so far. We will soon define them better. For now it is worth noting that the genome corresponds to the state and creature to the cell. Also important to note, that these five characteristics are present and strong for humans, but not for rocks, tentatively pre-affirming this theory.

The particular metrics used, as well as practical implementations of their calculations would be available in the third chapter, concerning the study design.

2.4 Important Concepts

Before we would be able to proceed to the study design section I would like to make a few definitions and to outline some concepts which I will use later. Since we are making a transition from a classical cellular automaton, to a complicated cellular automaton. Fundamentally not much is changing, but I would like to avoid confusion and get the terminology sorted out:

- *Cellular automata* or *cellular automaton*: in this thesis, these two terms are used interchangeably;
- *Genome*: a string from which simulation would be able to parse the information about the creature, its body parameters and certain behavioral biases;
- *Creature*: a “unit” in the simulation, with ability to move;
- *Mega creature*: agglomeration of creatures, typically with each element having unique specialisation or a specific role;

- *Hex*: a cell of which the environment is composed, as opposed to classical cellular automaton simulation, those are hexes, not squares, each has 6 neighbours (despite there being 12 pentagons) but individual cells would still be called hex;
- *Energy*: a characteristic of each hex in the environment. Is defined by a dynamic function with stochastic cyclical elements. Is used to simulate changes in the environment and create evolutionary challenges for the population, as well as to bolster creature environment interaction. Creatures would be able to store/release energy, whilst the environment can accommodate it;
- SUI (*Simulation User Interface*): a visualisator program that would be showing us the environment visually, instead of numerically.
- RC (*Render Core*): a Java application which will do the rendering of the final project. Other analysis would be done by working with save files (history of states of the simulation).

3. Study Design

3.1 Introduction to Project Design

3.1.1 General Description

The project aims to expand upon the classical cellular automaton model by simulating classical evolutionary principles via addition of genome based sexual reproduction to each ‘cell’ of an organism. Each cell in the proposed simulation will have an arbitrary long genome which would define the way a ‘cell’ would interact with the outside environment, that is, how it would react to its surroundings; compete for resources, interact with other cells and reproduce.

3.1.2 Implementation

3.1.2.1 Environment

Let our environment be represented as a 2-dimensional plane, subdivided into pixels of equal size. The plane size is planned to be made parameterizable (i.e., it would be possible to select a size before starting a simulation). Another important feature of such a plane would be the possibility to make it loop on the x or/and y axis (Illustration 2). The point of this feature is to avoid system’s impairment around the border region (which is a very common problem for many similar situations).

Each pixel of the environment would be able to host a cell, just like in vanilla CGL. But unlike CGL, it would also have the ability to store energy (see illustration 1).

0,c	0	1	1	4
0	0	0	3,c	c
1	8	2	1	c

4	1	2,c	2	c

Illustration 1

Example field. Structure: energy, cell. Each cell of the above table would be a cell in the simulation environment.

1	2	3	4	5

Illustration 2

The environment is wrapped around, so neighbours of 3 are 2 and 4. While neighbours of 5 are 4 and 1.

3.1.2.2 Energy

Since the simulation setting is a ‘game-of-life’-like simulation, I will also need to revoke the live/die/reproduce criterion to make it work adequately with improved ‘cell’ mechanics. To this end, let us define the notion of *energy*. In our simulation setup, every cell would need energy to live/reproduce, or take any other actions. If a cell runs out of energy, it dies. The energy costs would be subdivided into two main categories: *maintenance fee*, an energetical cost a cell would pay just to exist - it would be based on genome and organism characteristics (stronger ones might need to consume more), and *action fees* - how much an organism would pay per action.

In the simulation (at least during early stages of development) the energy distribution would be similar to the one of digital tree evolution projects. In other words, there would exist an energy source, (let us call it “sun”) which at the start of a new round would add an amount x of energy to every pixel on the plane of our environment. The x would not be constant, contrary to that it would be possible to express x as some periodical function with trigonometric elements, to simulate day/night, or seasonal cycle. By controlling the value of x it would be possible to experiment and see how life adapts to rapid changes to the environment.

Any cell would also have some energy stored to consume when needed. When a cell dies, it leaves some energy behind on the tile it died at. That would also allow to implement predatory behaviour into the simulation, together with parasitism and would allow for more complex interactions and more dynamic behaviours of cells in general.

3.1.2.3 Reproduction

Sexual vs. Asexual: reasonings

My simulation would permit both types of reproductions, because both are present in the real world as well. The type of reproduction an organism would be practising would be specified in its genome.

From my research/previous experimentation, I was able to come to the conclusion that sexual reproduction is in general significantly better - since it allows populations to evolve faster. Imagine the situation, an optimal genome is 12345 (highest adaptability). And in the asexually reproducing population there appeared (via mutation) two organisms 02000 and 10000, both of them have element of correct sequence, but only one of them would win in the population, since in the asexual reproduction there would be no way to combine them together quickly.

Asexual	Step x (base step)	Step x+1 (mutation)	Step x+n (all others died)
Sample genome 1	00000	02000	02000
Sample genome 2	00000	00000	02000
Sample genome 3	00000	10000	02000

Sexual	Step x (base step)	Step x+1 (mutation)	Step x+n (descendants of

			02000 and 10000 dominate)
Sample genome 1	00000	02000	12000
Sample genome 2	00000	00000	12000
Sample genome 3	00000	10000	12000

One large population sexual reproduction fixes this problem by having the most adaptable cells split their genome evenly to send to the next generation preserving successes of all members of the system, instead of just one most successful. Of course there are problems and disadvantages, like per say the fact that it might happen that “good part of” the genome is simply not selected on the next stage of offspring, but these problems are simply covered by the large population size, where by law of big numbers the population would be able to evolve correctly.

Asexual reproduction details

In the asexual reproduction a ‘cell’ would just create an offspring on one of the sides (where exactly specified by genome). The genome would be taken from its parent, with some random small altering (mutation).

Step 1

Empty cell	Empty cell	Empty cell
Empty cell	Cell A	Empty cell
Empty cell	Empty cell	Empty cell

Step 2

Empty cell	Empty cell	Empty cell
Cell A offspring	Cell A	Empty cell

Empty cell	Empty cell	Empty cell

Sexual reproduction details

In sexual reproduction a cell would find a partner capable of reproducing. They will split the genome 50/50 and create an offspring in any cell adjacent to parents.

Step 1

Empty cell	Empty cell	Empty cell
Cell P1	Cell P2	Empty cell
Empty cell	Empty cell	Empty cell

Step 2

Cell C1	Empty cell	Empty cell
Cell P1	Cell P2	Empty cell
Empty cell	Empty cell	Empty cell

Lifespan

A cell would have a life expectancy, the average would be specified in the genome and it would follow a Gaussian distribution. Hence, each individual life length would be somewhat unpredictable. For example, if the life expectancy of an organism is 20 turns (and standard deviation of 1), it could live for 18, or 21 years, not necessarily 20.

3.1.2.4 Simulation Order

The naive solution to the problem would be to make a cell by cell implementation, but it would give cells that start an unfair competitive advantage. Also the newborn cells, they would either be over privileged or placed in an unfortunate position. So classical list stack implementation would be flawed.

Approach 1

1→2→3→4→5→…→1→2… If a new element appears it would be in trouble because all other cells would be able to take ‘action’, whilst it would not be able to do anything.

Approach 2

2→3→4→1→5. An approach would be to select cells randomly, sampling without replacement. But that would be very chaotic on a smaller simulation, and would make the project much more difficult to debug. Plus there is an extremely high risk that the simulation would go out of sync, killing complicated organisms like colonies and composite nomads.

Approach 3

Making a linked list, where every cell would have a next/previous attribute. Each time a new offspring would be born it would be inserted after its parent. Each time a cell dies it is simply removed and the chain is restored. For example:

Cell 1 (Prev 4 Next 2)
Cell 2 (Prev 1 Next 3)
Cell 3 (Prev 2 Next 4)
Cell 4 (Prev 3 Next 1)

Assuming Cell 2 died, we simply restore connection, since due to the chosen data structure we can restore the flow without damaging integrity. Reconnecting cells 1 and 3:

Cell 1 (Prev 4 Next 3)
Cell 3 (Prev 3 Next 4)
Cell 4 (Prev 3 Next 1)

3.1.2.5 Genome and Mutation

Genomes are fundamental for our simulation. In the genome all the characteristics of a cell

would be stored (such as reproduction, life expectancy, behavioural preferences). An example of data structure hosting the genome is given below:

Genes	Val 1	Val 2	Val 3	Val 3	Val 4	…	Val x
Gene 1	002	232	212	100	011	..	071
Gene 2	093	081	122	062	100	…	164
Gene 3	051	023	205	111	034	…	078
Gene 4	044	082	142	001	044	…	195
…	…	…	….	…..	…..	…	…
Gene n	172	010	029	091	156	…	255

Genomes would be expressed as a sequence of genes (likely 16, 32 or 64 genes per genome). Note that by genome I do not mean the classical scientific definition of a “building unit”, but a complete set of characteristics/instructions for the cell. During the simulation it would be possible for the cell to switch between different behaviours.

The cell will have some physical characteristics, such as:

- Health points
- Energy storage capacity
- Livespan

and many behavioural characteristics, such as:

- Movement
- Aggressiveness
- Reproduction
- Energy management

Each “Val” from the table above would represent one of these characteristics. During each new turn the execution might go as follows. Suppose the gene (tiny part of it) is structured like this:

121-001

We first take a random number from 0 to 256. If it is above 121 the specified action would be taken. The genome would be big enough to account for all potentially takeble actions.

Active genes

Is an idea I saw on Simullife Youtube channel, which also does cyber biology. There to allow for many different behaviours the gene is split into a table of many smaller genes responsible for fewer interactions and then the cell picks the active genome that suits it the best depending on the environment.

Energy Management

To allow for more complicated life forms the cells would have the ability to transfer energy to each other to allow specialisation. Because the simulation would be made in a way in which if a cell is good at extracting resources from the outside world it would have debuffs on a physical level, promoting diversification and formation of more complicated systems.

For example the cell which is specialised at extracting resources would not be able to attack others. But would have an opportunity to give energy to offspring, which in turn might switch to predatory or parasitic behaviour.

The genome would be subdivided into 2 different parts. The first part would concern the physical characteristics of an organism. How fast is it? Can it eat other cells? What size will it have? From there we would derive an energy cost for an organism to last for the next round.

The second part of the genome would be concerning the behavior of an organism. Different biases (Granivorous or Carnivorous) propensity to stay still, propensity to nomad lifestyle, propensity to symbiosis, etc.

From a practical standpoint I have further developed my concept: each organism would be similar to a small artificial neural network (1 hidden layer) where the input would be environment characteristics, whilst the output would be one amongst all possible actions

which a being can take during a turn.

This implementation's big advantage is the fact that the creature would have the starting characteristics and biases in its genome (starting weights and balances), whilst would have an opportunity to adapt and change its weights and balances during the simulation[1].

3.1.2.6 Observations on Simpler Systems

Looking at most basic simulations I have discovered that there are only three scenarios that can happen when dealing with trivial simulations:

- Scenario 1: perma-statis. The simulation reached stability (or loop with step 1), where either all cells died, or all cells stuck at their static places.
- Scenario 2: loop. CGL gliders are a prime example of a loop. Cells engage in repetitive behaviour, each n steps they return to the same configuration (might move as a whole).
- Scenario 3: bifurcation, chaos. The system never stabilises and just acts stochastically.

My final product is expected to end up in one of the following scenarios. But before I expect some time of 'interesting development' when different life forms appear and compete for resources. Since the simulation is complex, the time before an appearance of an ultimate gene is expected to be somewhat long.

The calculations are going to be executed using matrices/tensors, whilst there would exist a separate visualization program, which would make us able to observe the situation as it proceeds.

[1] Have not decided on how to do this exactly, probably would be some process similar to mutation, but instead of being random it would be guided by an immediate reward system.

3.2 Simulation Architecture

3.2.1 Creature Architecture

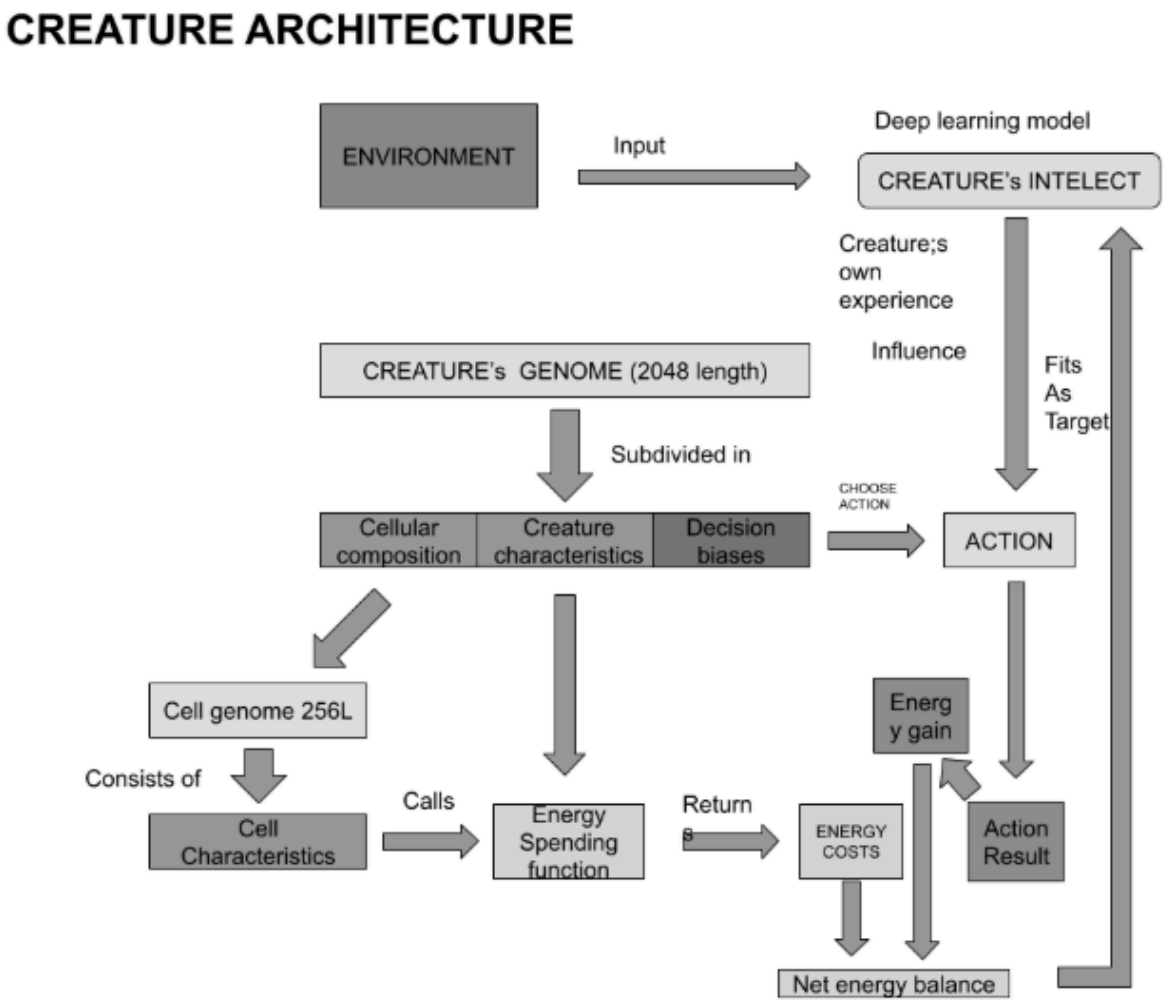


Illustration (A1)

In this illustration we see how an individual creature would be structured. The creature's key component is a genome with length less or equal to 2048. As said previously it is subdivided into cellular composition, creature characteristics and decision biases, where individual cells take up only a small size of a genome. Most important factor would be creature characteristics and decision biases, since most of the creatures would simply have one cell to begin with. Later on the creature would interact with the environment via handling energy, costs and gains some would be stored affecting the environment, which interacts with a creature by providing energy and being interacted by a creature via creature receiving energy as a deposit or releasing it.

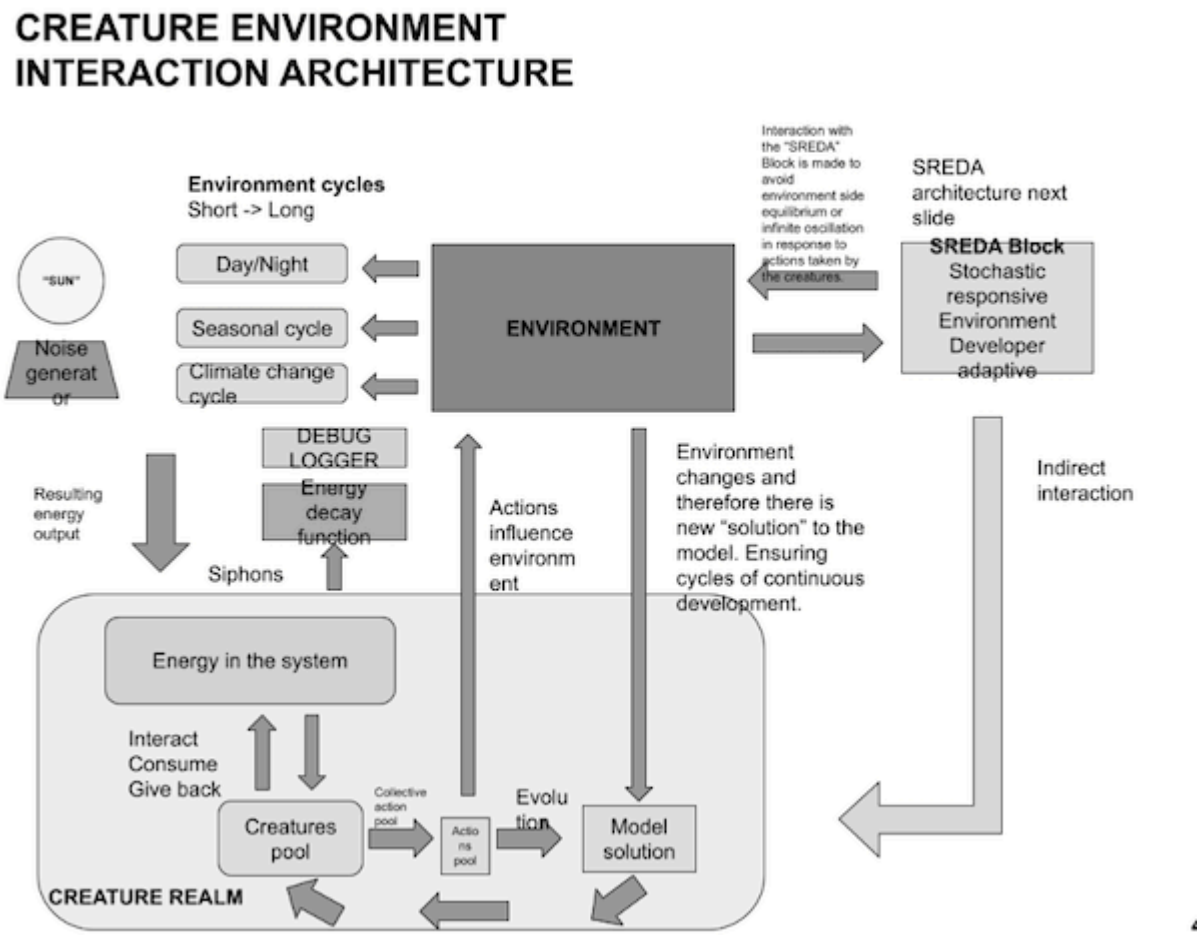


Illustration (A2)

In this illustration there is a bit more insight on the inner workings of energy function. It would be a noised triple nested sinusoidal function. With smallest cycles being called day/night more global to be called seasonal and most global to be called climatic. Obviously it is just a tag assignment, to make a mathematical function have some real life analogies. The SREDA block might seem unclear for now, it would be more clearly defined in the next illustration.

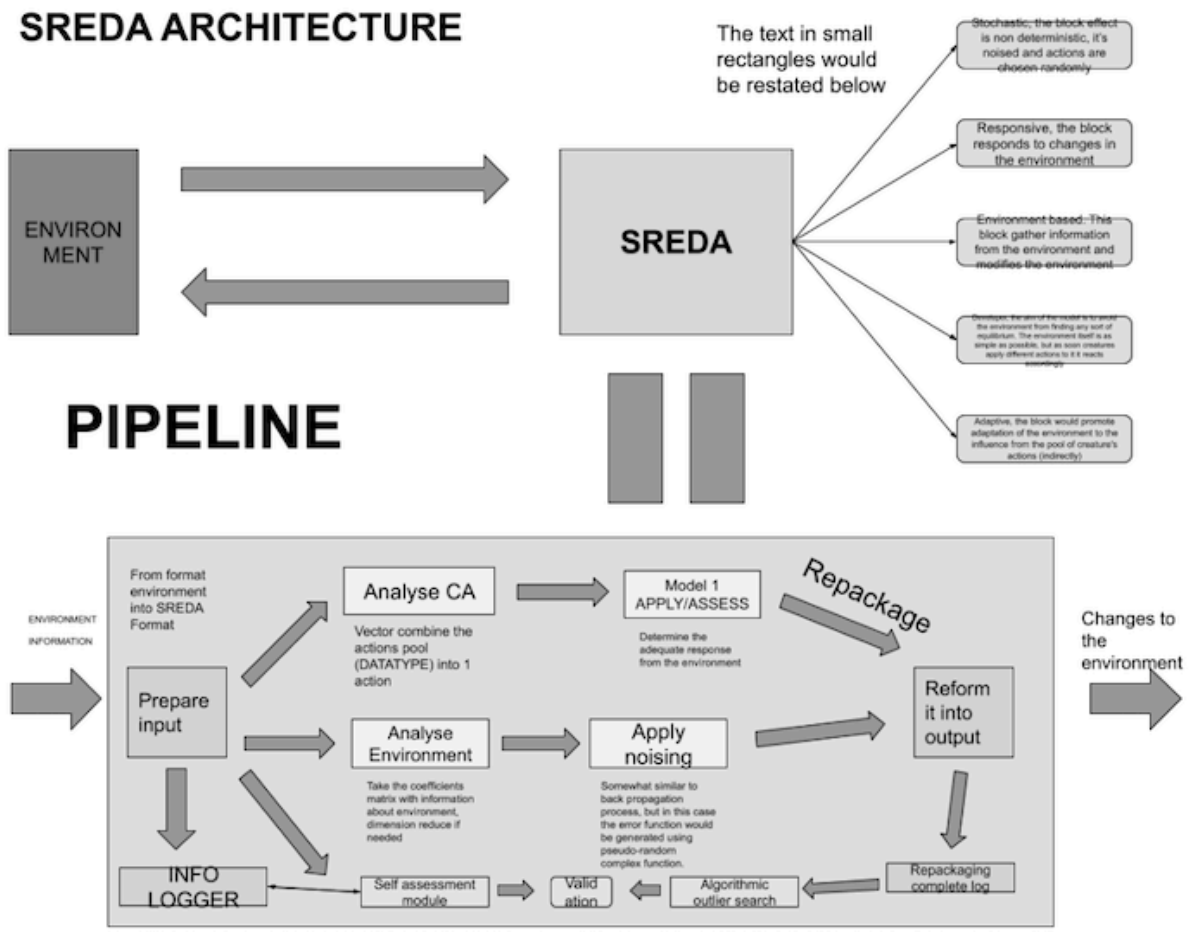


Illustration (3)

Clarifications: the aim of the block is to avoid the environment/creature interaction from getting into any sort of constant equilibrium. The environment itself is as simple as possible (at the start), but as soon as creatures apply different actions to it it reacts accordingly and develops together with them.

Another purpose of such an environment auxiliary would be to implement a logger function there, which would take the simulation data and create a json formatted save file, that would be consequently used during visualisation stages as well as parsed for analysis.

It is also important to know that there exists two separate parts of the project: the Python-based simulation code and Java-based render code. The reasons behind choosing Java for visualising the execution lies in the existence of more potent libraries for 3D visualisation, whilst Python was used in the simulation due to its speed.

3.3 Data Visualisation Environment

Visualising a complicated cellular automata can prove to be a big challenge. The first big problem which we needed to deal with was the borders, as I told previously infinite automatons are incomputable in the long run, but bounded are difficult to preview. I knew that I needed to have a loop to visualise the final product. At first I sought to simply have a 2D interface neglecting the visual imperfections. Yet later I came to the conclusion that it would not match with the overall spirit of the project and decided to bound the simulation by putting it on the sphere. Also instead of a sphere I wanted to use hexes, because unlike squares the diagonal and directly adjacent hexes are equidistant to each other.

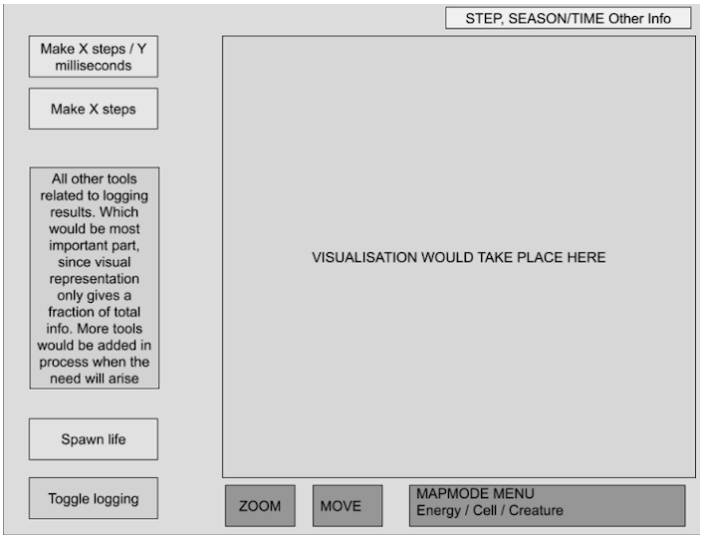


Illustration (B1)
An early prototype of the interface.

This caused some fundamental topological problems: the problem of tiling the sphere with hexes has been known for a long time and has been proven to be impossible and unsolvable. During my research I have found two potential alternatives: first one would be to use Lobachevsky geometry in 3-dimensions and generate a slightly corrupted space in which such action is possible, however due to the nature of a sphere such modifications would be visible and would create an visual unpleasantries as well as dramatically increase the computation power. The second solution was slightly different. By adding 12 pentagons to the tiling to be used as reper points, it actually was possible to generate a desired tiling to an arbitrary large size.

The process I used for it was as follows. To create a Goldberg (also known as geodesic) polyhedron I started by taking an icosahedron. Later I applied a truncate operation on it, cutting away its edges (with power = 1/2). Later on I dualed it, interchanging edges and vertices. I have found this algorithm on the internet and a particular website on which I found it is cited in the bibliography. But the Python algorithm and data types to represent vases and vertices as well as points was developed by me from scratch, the same can be said about dual and truncate methods, not libraries were used for it.

This code allowed me to make a visualization program with a predefined geodesic polyhedron.

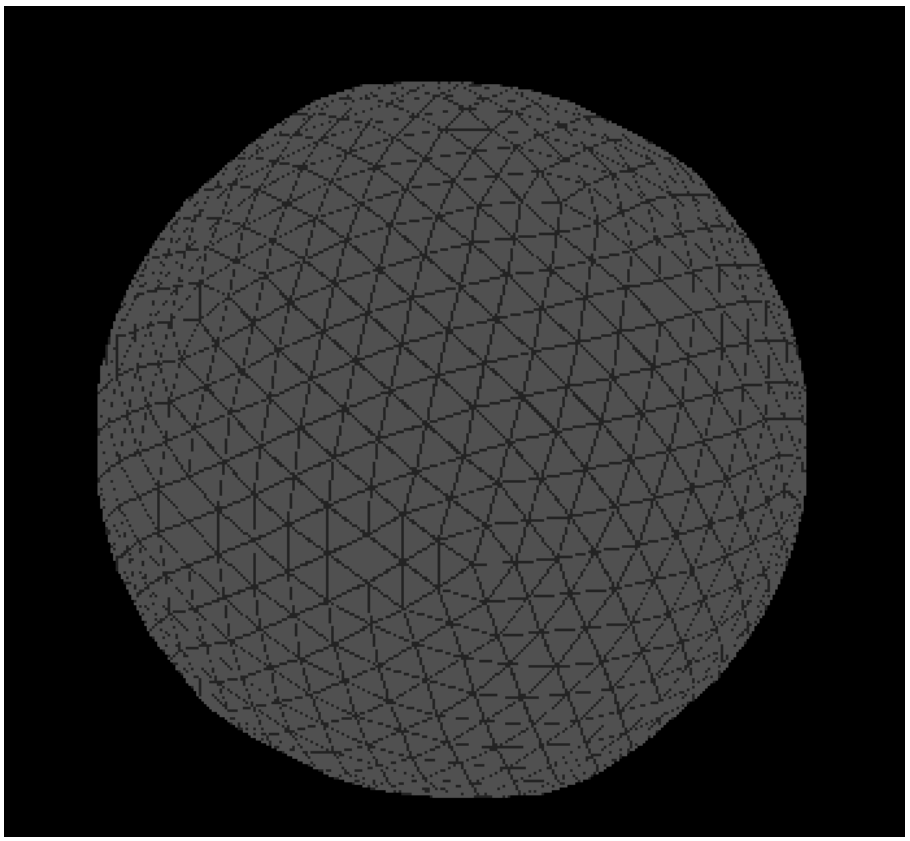


Illustration (B2)

Here an empty field is shown. There are no creatures in this image. There are hexes and pentagons here, the triangles you see subdivide hexes for better visualisation purposes, as well as for the individual cell map mode (TBD).

A widely accepted definition of truncation would be: an operation in any dimension (our case 3) that cuts polytype vertices creating new facets in the place of each vertex. In our case the operations were conducted on points, faces and edges files that helped to identify where “cuts

should be made". Vertices were cut half distance between the center and the said vertice, from both sides, keeping a half of previous edge lengths.

As for duelling fust like I said, vertices and faces switch places. Pairs of edges and pairs of vertices behave accordingly.

On Illustration B1 one was most likely able to see the *mapmodes* label: it was introduced but not defined, so let us fix this misunderstanding. Because our simulation is going to be extremely complex and complicated we need to be able to see all the nuances of cellular composition as well as to visualize some additional info about creatures' state of being, in simple cellular automatons.

Like CGL per se there exists only 2 states, which are easy to depict, in some simulation there exists 3, 4 or even 5 states which are also RGB depictable, but in our case we have much more combinations (genome is massive) and so the need arises to somehow visualise all that plentitude and diversity.

In order to do that I have decided to implement the mapmodes feature commonly used in modern strategy games, which are faced with similar problems. We will have a few mapmodes depicting different information about the cells, for example their life expectancy or aggressiveness with color.

This would allow us to represent more information that we otherwise would have been able to represent. This does now imply however that everything would be visualisable, we would need to make some sacrifices or dimensionally reduce and aggregate the genome information. To reflect on the visualizer imperfection however we would use save files for a more rigorous analysis with better information extraction possibilities.

3.4 Save File Handling

As discussed above, the point of save files is dual, firstly and primarily they were planned to be used simply to be plugged into the visualisation tool and be parsed. Yet it turned out that visualizing the entire information would prove to be an almost impossible task, so save files

secondary usage lies in the fact that they store the entire information about the simulation in an easily accessible format.

We are able to separately open and parse them in a Jupyter notebook and use all available Python libraries for data analytics (e.g., Pandas, NumPy and Matplotlib), to handle and create graphs.

The structure of the save file is simple. It is essentially an array of Python dictionaries casted to string. The dictionaries represent a graph, for each ID of each simulation cell the first element is an array of neighbours, the tiles the said tile is neighbouring (5, 6). Thankfully, unlike Java, there are no predefined sizes for an array and handling of this issue is automatic, otherwise a shadow tile would need to be created like -1 to add six neighbours to the pentagons. Afterwards there is creature data storage: every cell and its genome from which we are able to parse the data that interests us during the analysis.

One potential limitation of these .json files, is that they weigh a lot and due to us requiring a save file for every turn of every run (of course we can discard them, but we might need them for historical analysis) it would be quite inefficient memory wise. A proposed solution to that issue would be implementing a zipping algorithm, or using an existing one to zip save files in the storage, that would converse the memory, but we will pay for this with growing computational costs and greater code complexity (deemed in the end not worth it, since memory costs are big, but bearable on commodity hardware).

Lastly regarding save files I would like to mention that there are sometimes bugs related to parsing which arise with the inability of Python code to always recognize special characters within a genome like { } therefore in a final version of the project genome is only represented alphanumerically.

4. Methodology

We have reached an important milestone. Beforewards I mostly talked about various theoretical approaches to the topic, outlining theory and its concepts as well as talking about high level ideas, without going much into the results. Despite I have outlined the practical implementation of the graphical part of the project as well as saves I still have not talked about what results were yielded, what data was collected and how data was handled. When it comes to the practical implementation all previous progress logs as well as all previously written code related to the project would be in the Appendix, so I would not talk more about it here. The goal of this section is to actually describe the experimentation conducted and answer some of the most important aforementioned questions. In this section we would not talk about the findings and analyze detected trends yet, we would only outline the toolset that I am planning to use during the analysis. I consider it important because there are many potential approaches to collect and interpret data, and whilst using all of them and picking the best one is certainly a preferable action, it is regretfully by no means feasible. We are always stuck choosing from a broad set of methods and only using the seemingly most adequate ones - for example if we wanted to use prescriptive analysis the resulting output file would need to be formatted in a way accepted by the majority of machine learning models (as a first step), say formatted with integers and floats. But I am only planning using descriptive, diagnostic and to an extent predictive analysis, since it is difficult and implausible to prescribe something for a self-developing model. So far our goal is to only measure the development, by the first five criterions outlined previously in Chapter 2 as well as to objectify them as metrics and ensure that they are growing. This would mean success for us, and yes of course further researching the model to notice deeper patterns and relationships is definitely preferred, but it does way beyond the scope of this report. To use this model for future endeavors a basis is to be constructed first. Of those five rules, the first three and fifth are extremely straightforward since they can be measured just by looking at the simulation: survivability is whether the creature has enough energy, complexity is about genome and impactfulness is how much energy the creature released into the environment. The only rule metric that causes problems is the complex trend of dynamic evolution, for it we would need to work with savefiles over time and plot complexity as a function of time for many different creatures, to avoid mistaking a rare anomaly for a trend.

4.1 Data Collection

The principles which we abide by when collecting data are relatively simple. When starting a simulation each turn we create a save, in which all the data about the current turn of the simulation is saved, permitting us to get turns from S_0 to S_N to S_F, where S_N stands for final state. These saves in turn are saved in separate folders, one folder per run, not to confuse the same S_F's of different simulation runs.

The data collected is the genome of the creature for every single hex on a sphere. As well as information about connections of geodesic polyhedron faces with one another. For computational cost efficiency we have decided to also separately save most important parts of the genome, to avoid parsing it each time for mapmodes, as well as during the analysis.

As per the individual metrics the situation is as follows:

1. *Survivability*

Let us define a creature age as A, in our case $A = S_D - S_B$ (where B is the turn a creature has appeared on the map and D the turn in which the creature perished). It is worth looking at the mean value of creatures life expectancy, at any particular moment as well as overtime, to see for the adaptability. But the survivability of an individual creature is not simply its age. Some creatures are short lived, by their genome so the creature life expectancy should also be taken into account. Let us define the creature's life ratio (LR) as $LR = A/EA$, where EA is the expected age of the creature according to its genome. When the ratio is 1 or above we can conclude that the creature performs relatively well, else we can conclude that the creature lives less than expected or is being killed. The cause of death is also important, because if a creature is dying of energy deficit it can imply evolutionary failure, whilst being eaten is more chance dependent and maybe not as affirmative of a sign. In general that can be incorporated into a formula simply as a factor. Hence we define survivability as $S = A/EA \cdot CD$, with CD being the cause of death. This could have been handled by SREDA block, but since it is done outside of the simulation and the metrics are to be objectified from my point of view I will just give cause of death some arbitrary values, which would reflect how successful (unsuccessful) each outcome is, with death of old age being 1, death by killing being 0.75 and

death by lack of energy being 0.5. These values can be changed depending on the researcher's values, since survivability can be seen as a subjective metric.

2. *Adaptability*

Adaptability is in our case essentially the performance of survivability over time. How does a creature perform over the course of its life? Since we are not interested in mixing local and global changes, we at first would look at an individual creature with the same genome and see how much food (energy) the creature is getting over time, in times of moderate changes of the environment.

Later on globally we would look at how the mean and median survivability changes over time as well as global energy consumption and depository by all the creatures. This data would be stored separately from the visualizer, since it is only capable of opening, handling and dealing with only one save at a time.

3. *Complexness*

Complexity, unlike the two previous characteristics, is going to be measured solely by looking at the genome. It would be a maximum depth akin to the one of a "decision tree" of the genome plus the average decision tree depth. That is needed to evaluate how complex the creature is as a whole, and what is its most complex possible decision. Because the genome segment might be as follows "if the cell above aggressiveness is above 0.5, move left". But it could also be "if the cell above aggressiveness is above 0.5 and my power below creatures power, move to the side where there are least creatures.". The tree depth is defined as a number of conditions in each particular segment linked to the behavior of a creature.

4. *Complexness dynamics*

Nothing new, very similar to the previous point put over time. We would need to look at save history and construct a plot of the measure defined above. It would allow us to see if globally creatures as a whole and therefore a system have a trend towards self-development, or are they limited by the environment and forced into degradation or stagnation. For example, there exists some simple solution to continuously do good and adapt well and even at different

energy levels this “META” works discouraging the evolutions and breaks the system by bringing it into almost sure equilibrium[2].

5. *Impactfulness*

Simply as a measure of how much energy a creature is releasing into its environment. Inside the SREDA block this release energy is handled and is sent to affect the energy generation function on top of the noised sinusoid we talked about earlier.

These five metrics are later summed, as greater means better for all of them. Therefore a continuous growth, or at least like perpetual periodic motion (cycles) means (would mean) that we have completed our task of developing such a system successfully. We can tolerate some temporary falls, since when dealing with stochastic motion and a lot of randomness and complexity that often happens, we only care about overall positive trend, especially among multiple attempts, failure of several runs would not be considered a project failure, such a demonstration of such a development by even a fraction of the tested worlds (even just one, but better more for replicability) would prove the possibility of such a simulation’s existence and would definitively prove out point.

4.2 Handling Replicability

Replicability in the context of our simulations does not mean the ability of the simulation to demonstrate the same end result or behaviour continuously, multiple times in a row for all the possible cases. Indeed this is not required, we are only interested in a small fraction of the total cases to be successful, because we only need to prove a possibility of a certain phenomena existing and in order to do that we are required to have a strong understanding of what are we measuring, which we already have and what would be considered a successful set of trials, whilst what would not.

A successful trial is, as specified before, a continuous long term growth of the five metric agglomeration. Whilst a set of N trials are going to be considered successful when among X

[2] When dealing with random variable (and in our case stochastic factors during sexual reproduction stages) we can not talk about limit, instead we use concept of “almost sure limit”, extrapolated by me into the concept of “almost sure equilibrium” where equilibrium is reached, unless something super improbable starts continuously happening.

trials at least 1 demonstrate the desired behavior, of course the higher the ratio of successful trials compared to the failed ones the better, but one is sufficient. The logic here is somewhat similar to the reasoning behind Collatz conjecture: if we are able to find at least one example of a number that never reaches 1, we would be able to proclaim it to be solved.

4.2.1 On the notion of gambler's ruin in the context of cyber biology

In statistics a “Gambler’s ruin” is defined as a theorem that a gambler that plays a game with a negative expected value will eventually go bankrupt regardless of their betting system. Applying this concept to cyber biology we would be able to rephrase it in this manner:

We can hypothesise that certain rulesets of cellular automatons might create a system, where the expected value of a development function on average is decreasing over time. That would mean that no matter how well the simulation is performing from turn 0 to turn X, it is bound to fail. Since we only can project the simulation for the finite amount of turns we would never be able to identify a “Gambler’s ruin scenario” neither by experimentation nor theoretically.

So we can only make an assumption, that if the simulation was running for long enough and the average development has continuously and steadily grew, means that the ruleset does permit a continuous growth, yet due to the concerns outlined and described above the proof might never be completely rigorous, requiring greater replicability and more retesting for a more prolonged time periods to increase a probability and strength of our hypothesis.

Earlier in our thesis we have managed to prove that a determined ruleset on a finite space with a finite number of spaces is bound to be periodic with periodicity greater than 0 (Theorem 2). This however does not apply to our simulation, due to the usage of stochastic factors. When it comes, however, to almost sure convergence, we do not have a way to prevent it from happening, except for the hope that SREDA and dynamic environment would make such an equilibrium impossible.

There are concerns however that all possible systems of control which were used to break an equilibrium would eventually enter an equilibrium with the system itself. The last counter factor is the ability of a creature itself to influence the environment, because according to the

main hypothesis of my thesis this is meant to induce a complete cycle of a non-stop self-development.

4.3 Developer Tools and Methods for Data Analysis

Jupyter notebook

Is a very important base, supports interactive computing on multiple languages, in our case we are interested in python and its libraries. I decided to use it due to its convenient format and broad flexibility, as well as availability of it in the browser, together with the export to pdf option. Most of the progress logs were completed in it.

Matplotlib and Seaborn

Very important python libraries, which I am using to construct graphs. Very versatile and quick. Are able to work with databases and can show much more compared to the SUI application.

Numpy

C-based library, which I am going to be using for quick operations on large data. Can deal effectively with arrays and matrices, is quite useful despite vanilla Python offering quite a lot of tools to perform calculations by itself.

Scikit learn

For any sort of predictive modeling. It is a little bit out of scope for our project, but it is so potent and versatile, so I decided to occasionally use it anyway, to develop the prediction of how the simulations are going to perform in the future.

Version control

Since most of these libraries would be used in Jupyter notebooks, data from which would be later partially reused here, partially uploaded to Github repositories version control would not play as crucial of a role. On Github you would still be able to find the conventional requirements.txt, but they are not supposed to cause difficulties. As per simulation itself I am mostly avoiding any non vanilla libraries, making the code more safe to run. It still requires Python 3, preferable v3.12 to run well, however using the latest version is expected to work.

Lastly it is import to note that since the source code was modified and altered over long periods of time the most up to date code documentation would be stored right within in as # or “””” comment sections. The naming conventions I have decided to use would still be in accordance with the architecture designs and pipelines shown in the second section, in order to make understanding the code simpler to the user. In all other unspecified cases classical pythonic naming conventions are used in Python code segments, whilst in Java I decided to stick with the camel casing approach.

5. Findings and Results

5.1 Preliminary Analysis

In this first section, we will not focus on operating the final prototype, but preliminary analysis of picking correct types of environment and verifying architectural and project design decisions (mainly concerning sexual vs. asexual reproduction types).

When moving between the theoretical concepts provided in the first two chapters towards the practical implementation I have encountered a new set of problems and challenges, but took significantly more time to solve than anticipated. In this section I am going to show off the results of building a practical implementation of a genome which would be used in our model.

At first I constructed a simulation of a hypothetical genome for sexual and asexual reproduction. Where in the first case the genome of an offspring is represented by a mixed genome of parents and in the later the genome of an offspring is just a genome of a parent. Later on, mutation would be implemented for both of these systems and we will see how its rate is affecting the population.

Before introducing the environment I was viewing a genome just as a n-length string of alphanumeric characters. For example “abcdefgh” where the adaptability would be judged by the genome proximity to the “ideal” genome (an arbitrary defined string, before introduction of the environment).

We would look at the different trends present in the simulation as we play around with the various parameters and look at the different trends present in the simulation. Our particular focus would be the adaptability of the simulation.
During the initial phases of testing the parameter n, notably the length of the genome, was set to 50. It was long enough to offset potential stochastic fluctuations, but not too big to slow the simulation down in the long run. Later on the genome length was expanded in order to match the growing requirements faced.

Below are some graphics we constructed after running the simulation for 10,000 generations. Mutation rate type A is when the total change in genome has a 5% chance of happening and type B, when such chance is for an individual string.

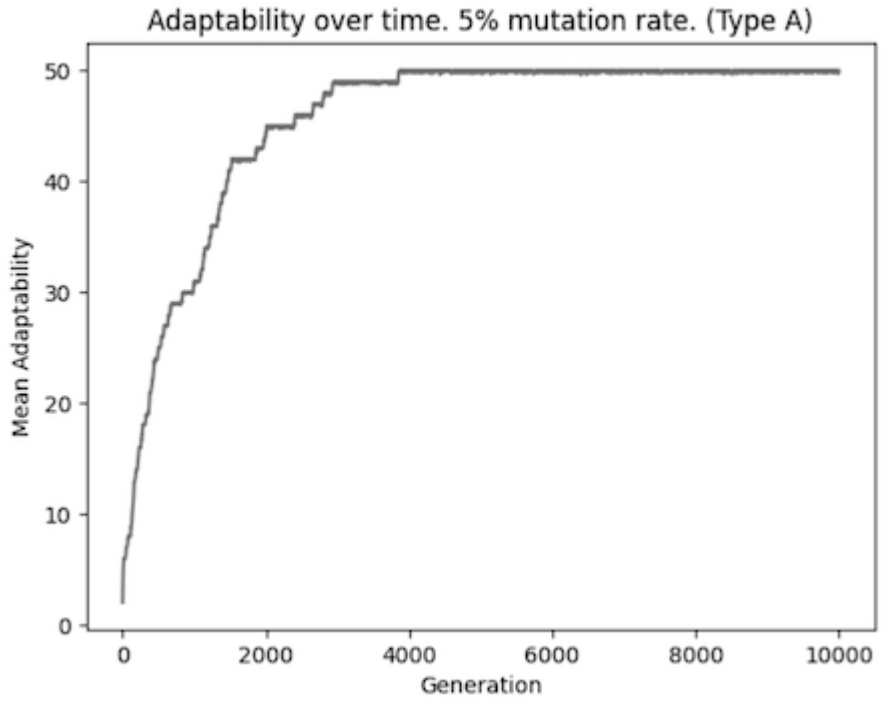


Diagram (C1)

We can see that at a 5% chance to have a single change in a string it takes approximately around 4000 generations to reach max adaptability. Afterwards adaptability does not noticeably decline.

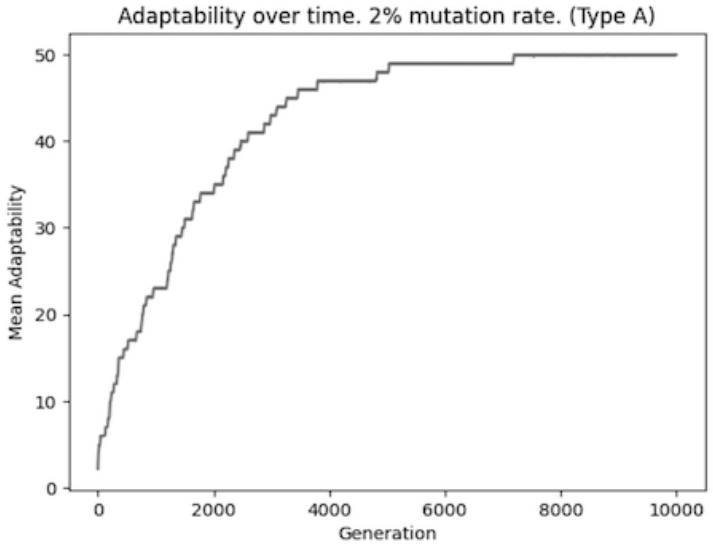


Diagram (C2)

We can see that when the mutation chances are reduced it takes longer for the population to adapt to the current static environment. Note that the stability in the end is only due to the environment being static, later on it would be subject to change.

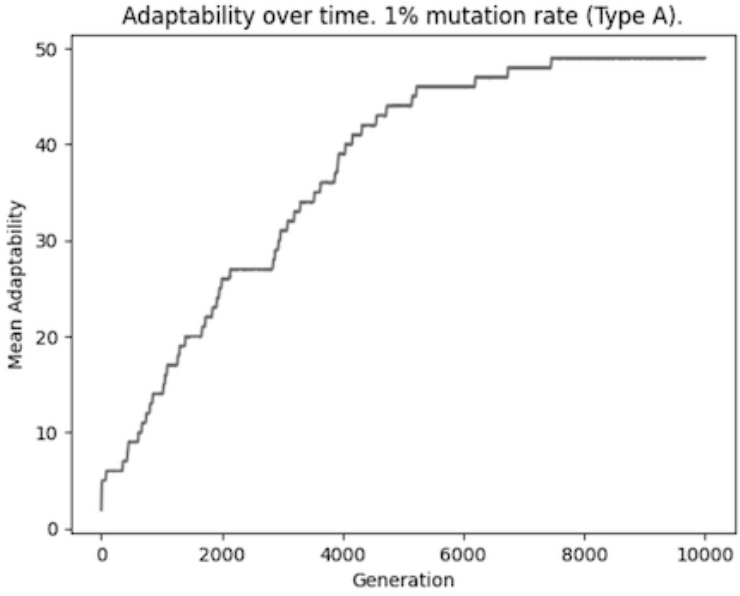


Diagram (C3)

On this diagram we can see the continuation of trends from the last one. The adaptation to the static environment is even slower, but still generally monotonously increasing.

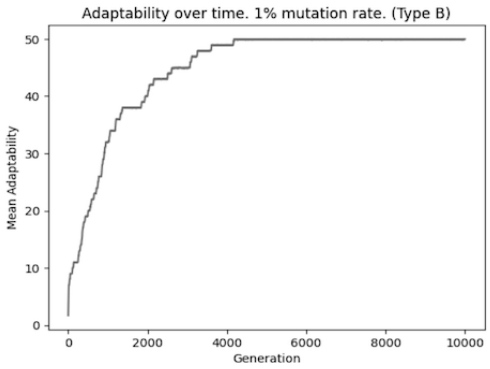


Diagram (C4)

As expected, type B mutation takes effect much quicker, and has ultimately a higher rate even at 1%, because of the expectation of a change in type A terminology would be $0.01 \cdot 50 = 0.5$, hence way higher rate, and it would be even greater with a higher genome.

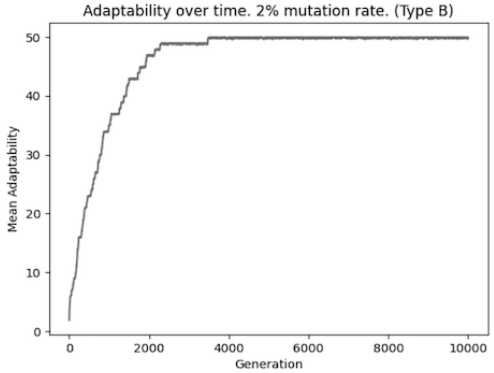


Diagram (C5)

The adaptability is showing signs of continuous growth as mutation rate is growing.

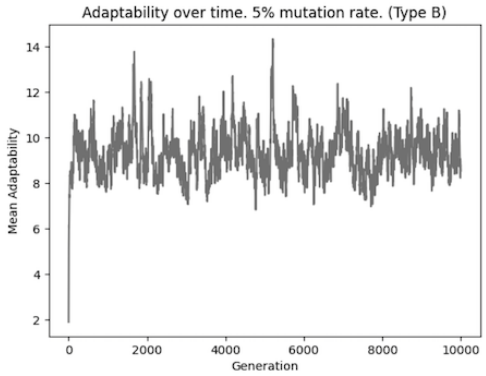


Diagram (C6)

Here we can see that we overshot with parameterizing for mutation rate, it is so quick that successful mutations simply do not last for a reasonably prolonged time period.

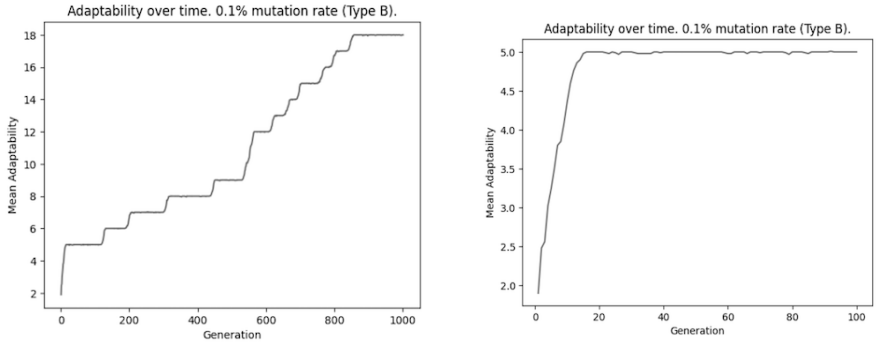
Now the results have changed significantly, but adding a degree of realism since mutations in our simulations are now closer to the real ones. They are more stochastic and good distributions are maintained, avoiding mutations of the same small size which would be insufficient for our simulation.

At a very small probability of mutation we reach 50 adaptations pretty quickly as well, which is understandable, since nothing about the simulation changed fundamentally. But at a high mutation rate we can see that the population does not reach the ideal genome, since they mutate too quickly. The problem from the last part when negative mutation possibilities outweighed positive mutations possibilities is hitting stronger at a higher mutation rate.

Note that from now on we will be using our Type B mutation (that is, character based mutation) because Type B allows for mutations like "aab" → "ddb", while also maintaining a probabilistic distribution (that is, "aaa" → "aab" is more likely than "aaa" → "abb"). Conversely, type A mutation is incapable of that.

Now let us look at and analyze earlier stages of type B mutation to see the evolution of mean adaptability over the first 1000 and 100 generations respectively.

Diagram (C7) & Diagram (C8)

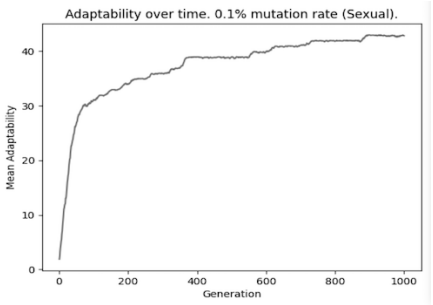


Here we are able to clearly see the striking revelation, what seemed to be the start of constant stability was only the beginning, as we can see in longer run, it was due to the fact that evolutionary changes are happening quite rapidly over the entire population: when a better genome is discovered the majority of the population switches to it. Some small oscillations can be and are explained by the constant continued mutation, the new mutated (unsuccessful)

creatures are born, but with a failed genome they die out affecting the mean adaptability only temporarily.

On Diagram C7, as instead, we notice an interesting step-based trend. This is most likely because of the fact that there are periods of stasis (when no new good mutations appear) followed by periods of rapid development (where a more advanced species quickly - matter of a few generations - overtakes the population). As we can see the asexual reproduction model successfully shows itself to be capable of evolving and developing given a static environment. Yet with more creatures their model begins experiencing problems. If we have two creatures who both had different successful mutations parallely only one would win.

Note in case of sexual reproduction the mates are paired randomly among the members of the surviving population. We can see the results of a sexual reproduction on Diagrams 9 and 10.



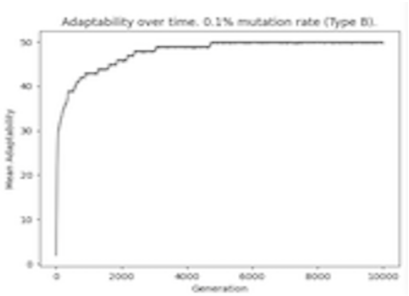
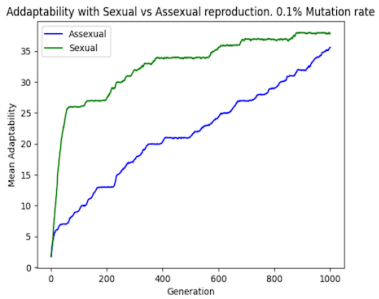
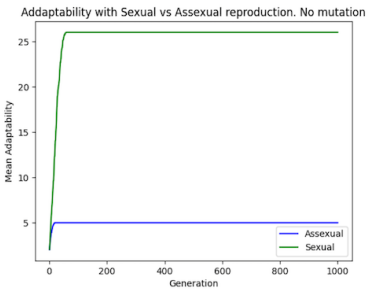


Diagram (9C) Diagram (10C)

In the short run the results are amazing for sexual reproduction. By generation 1000, asexual adaptability was only around 18/50 whilst sexual reproduction already made it to 40/50 adaptability. Also the rise is now more gradual.

In the long term however sexual reproduction has slowed down. This is not as bad since in real life the environment is dynamic and it is important to catch up to the change quickly rather than idealizing one particular static setup.





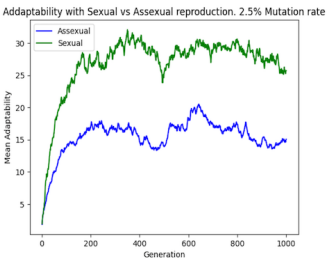


Diagram (C11) Diagram (C12) Diagram (C13)

Adaptability with sexual vs asexual reproduction at 0.1, 0 and 2.5 mutation rate respectively.

Here we can see a curious phenomenon. Both creatures reproducing sexually and creatures reproducing asexually start at the same footing, but whilst the asexuals only managed to create their dominant gene as direct and unchanged successor of one of the initial generated creatures, the sexually reproducing creatures managed to combine all the material they were starting with developing a significantly more adaptive genome. Yet with no mutations the evolution came to a halt. In Diagram 14, we can see the impact on the mutation rate on the adaptability for sexual reproduction.

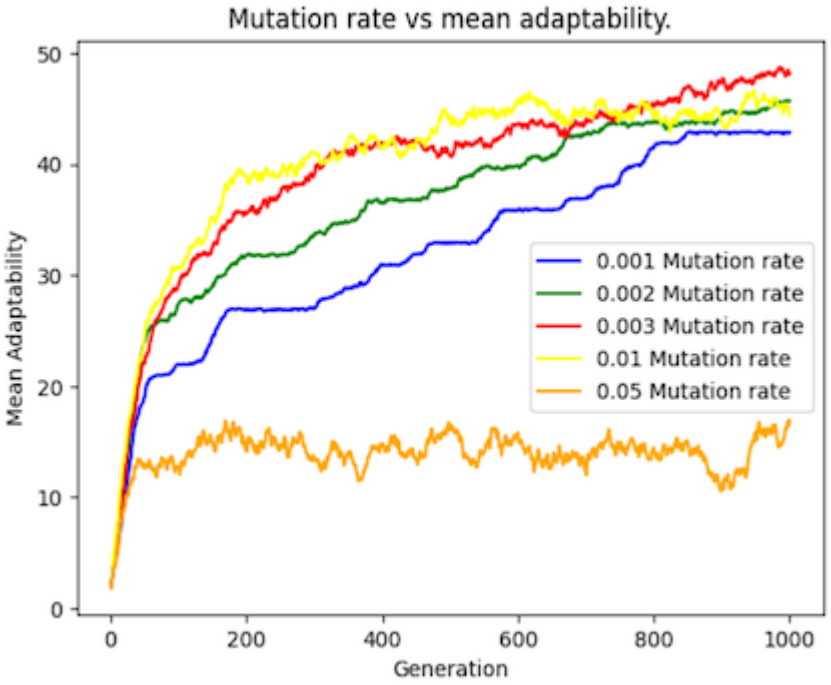


Diagram (C14) - comprehensive summary of varying mutation rates for sexual reproduction.

5.1.1 On the Notion of Genders

As we can see in our simulation there is no genders everyone can mate with[3]. This makes sense and begs a question why there are genders in real life for most of the animals, except for some hermaphroditic species (e.g., Lumbricus earthworms).

The reason lies in the fact that it is expensive for creatures to hold a complete set of reproductive organs. So specialisation appears, which later becomes gender. It allows for an even quicker evolution since now mating selection is more quick and does not rely solely on the environment, also it takes less energy and allows specialisation of genders.

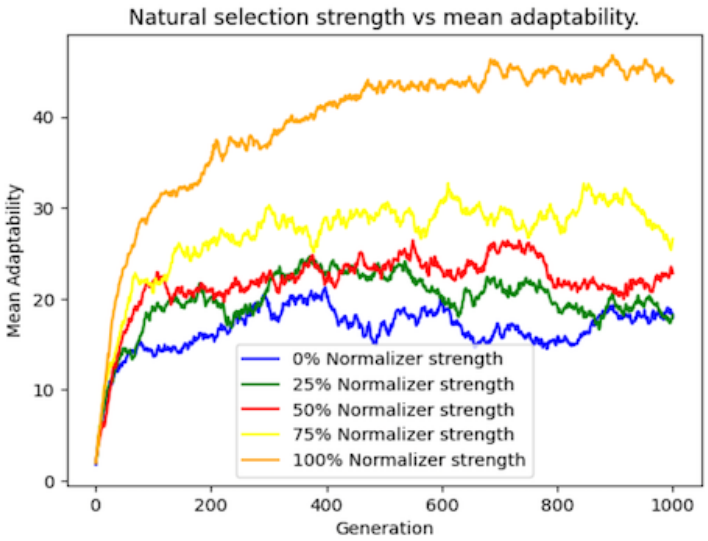
5.1.2 Strength of Natural Selection

In my project at the start there will be no genders, but my hope is that by the interaction with the environment creatures would naturally evolve into having them. There would be many "Construction pieces" in the final project which would result in many potentially interesting possibilities.

During the early stages of the project we were normalizing adaptability, yet it might be interesting to see what would happen if natural selection was more weakly/strongly inclined towards survival of the fittest. Let us add yet another parameter to our model to check it out.

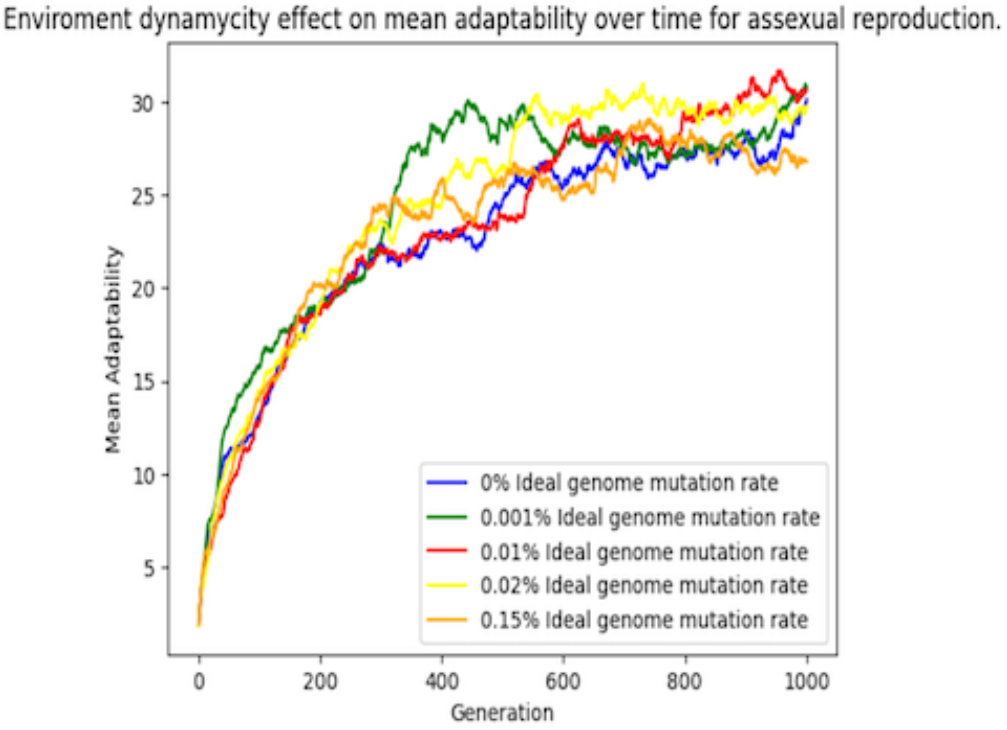
Diagram C15 : effect of normalisation on mean change of adaptability over time.

[3] Like in the novel *Brave New World* by Aldous Huxley.



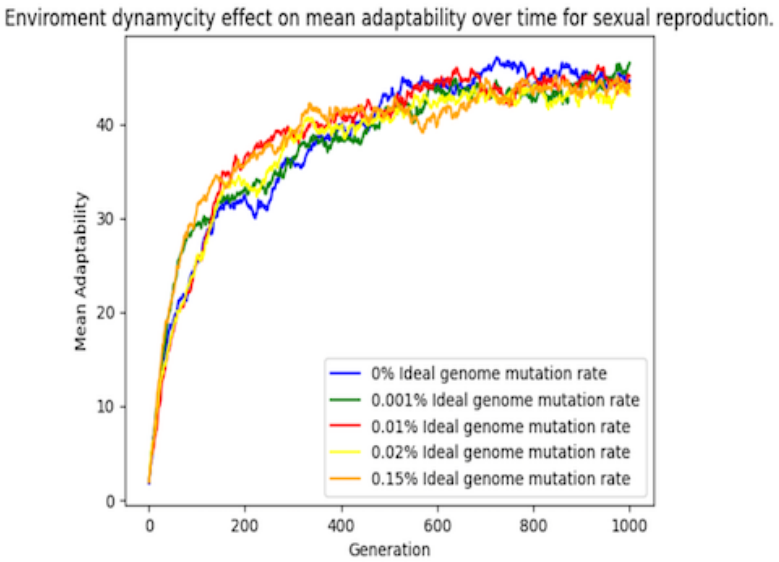
Without normalisation more fit creatures do not perform as well (e.g., values 21, 22, 23 and 24 are almost the same, whilst 1, 2, 3 and 4 are not: from a ratio point of view 1/4 = 0.25, whereas 21/24 = 0.875).

Diagram C16: Introducing environmental dynamicity.



Now there is yet one more experiment I would like to do. In the underlying philosophy of my project lies the concept of a dynamic environment. Let us see how creatures would behave if suddenly the ideal environment would also change.

Diagram C17: Sexual reproduction in the dynamic environment



5.1.3 Closing Remarks

During this section I have started working on practical implementation of the genome for my simulation. I have created most of the methods that I will use in the future. Also I have conducted some research with basic reproduction simulation, which allowed me to gain a practical understanding of the rules of evolution.

In the next section I will actually determine and demonstrate how the genome would be structured in its final form. Plus answer the questions of what and how it would operate in the final project and show the results of running it.

5.2 Complex Analysis

When the simulation was complete I decided to launch a 20 trials 10,000 turns long, sadly due to computation complexity I later had to reduce the simulation time to 2,500 turns, yet there

are no limitations or errors besides time and CPU power that prevent us from running the project for greater amounts of time. Please note that from now on I will be talking about the current project in its final form. In the previous section I dedicated some time and space to show off the result of preliminary research and early prototypes, but as I said from now on all the statements and information would regard the latest versions of the project in its entirety, whose latest version is accessible on Github[4].

First of all it is important to note that many simulations ended up dying out before turn 100. An estimated fraction over 100 trials is 37/100 or 37% these appear to be a large figures, yet it is important to note that over the course of this trials we only started with two organisms, who upon closer inspections turned out to have both failed genome, causing the simulation to fail, by roughly around turn 60. However if the simulation did not fail over the course of the first 100 turns it appears unlikely to fail in the future, over my 10 long term (2500) runs no signs of failure were detected. The complexity and development of the simulation did not show or exhibit any traits of strong degradation, yet due to the factors we will discuss in this section later we can not rule out a possibility of such a thing happening.

Over the course of this part of the report we will be focusing on two trials, both on Github: one was medium-term (350 turns), another was long-term (2500 turns). In the future it might be interesting to attempt to run 2500, but sadly due to the deadlines there was no time to run this process a sufficient amount of times to be able to derive conclusions and have the ability to generalize trends at such a long period. I will be referring to the 350 turn experiment as trial 1 and 2500 turn one as trial 2. Every graph would have a caption specifying the experiment, trial and rounds it belonged to. Assume linear scale is used, unless specified otherwise.

[4] Note that due to repository size and file size limitations instituted by Github I could not upload many saves as not to go beyond the 1 GB total threshold. Some save files whose zips were heavier than 100mb were either not included, or split into two parts. I have attached a comprehensive guide on how to remerge them on Github in the ReadMe file. I would also like to say that some screenshots which I was not able to attach, due to the thesis having a mandatory < 15 Mb limit are also located on Github in the “resized screenshots” folder.

5.2.1 Visual Analysis

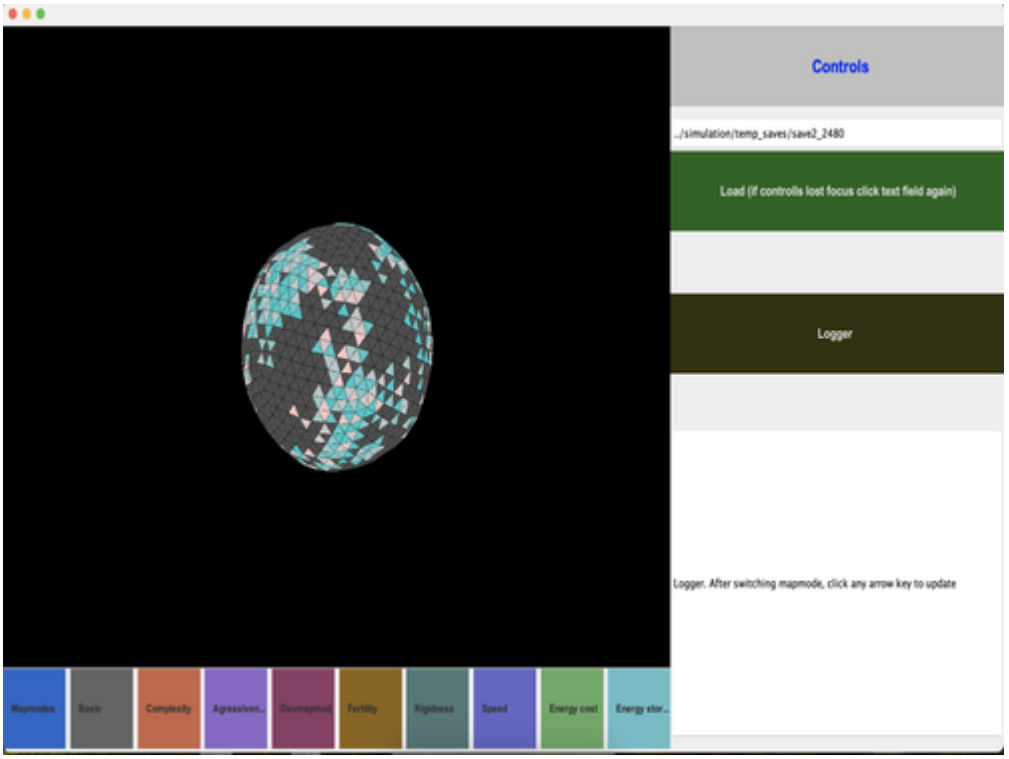


Diagram 5.1A

Here we can see the process of visual evaluation of the simulation. A second trial is open on the turn 2400. The simulation is currently in development mapmode. The colors are normalised from greyish to blue with blue being less developed cells and whitish more developed. This screenshot is attached as a demonstration to show how the project is working. Due to size constraints it is descaled and lost proportions during de-scalement, also it is a bit unclear, the original program has a far better resolution. This screenshot is attached only as a demonstration and as a proof of concept.

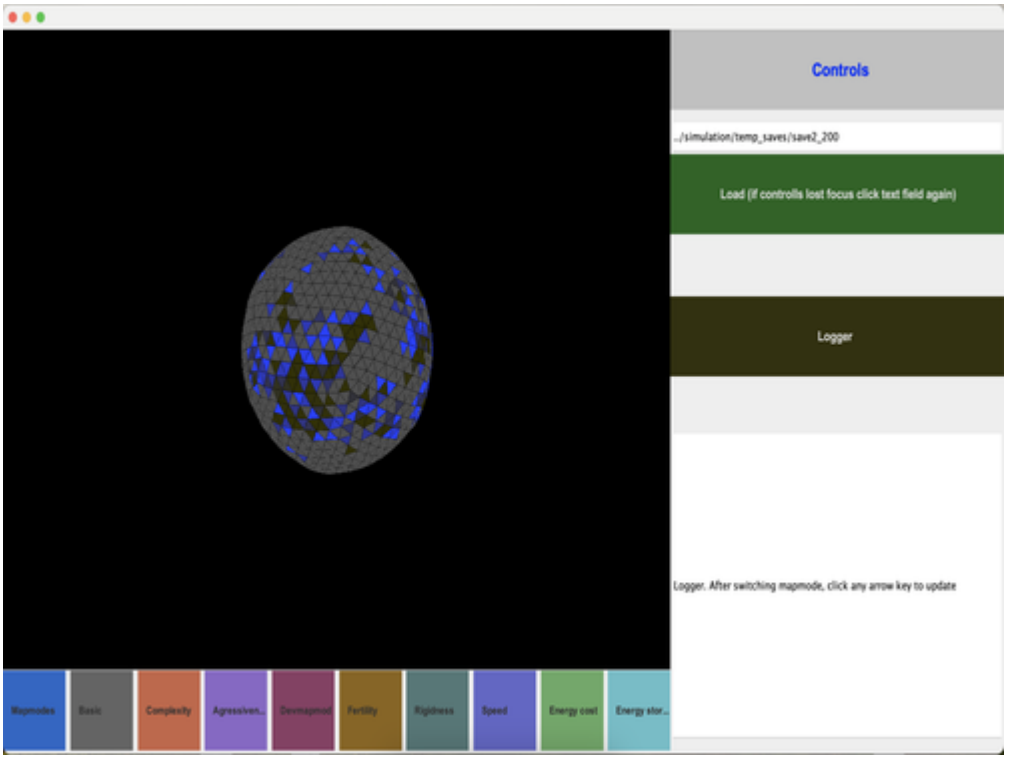


Diagram 5.1B

Now you can observe the energy distribution among the cells on the 200th of the second trial, the brighter the cell is the stronger energy it has stored currently. Please note again that it is min max normalised.

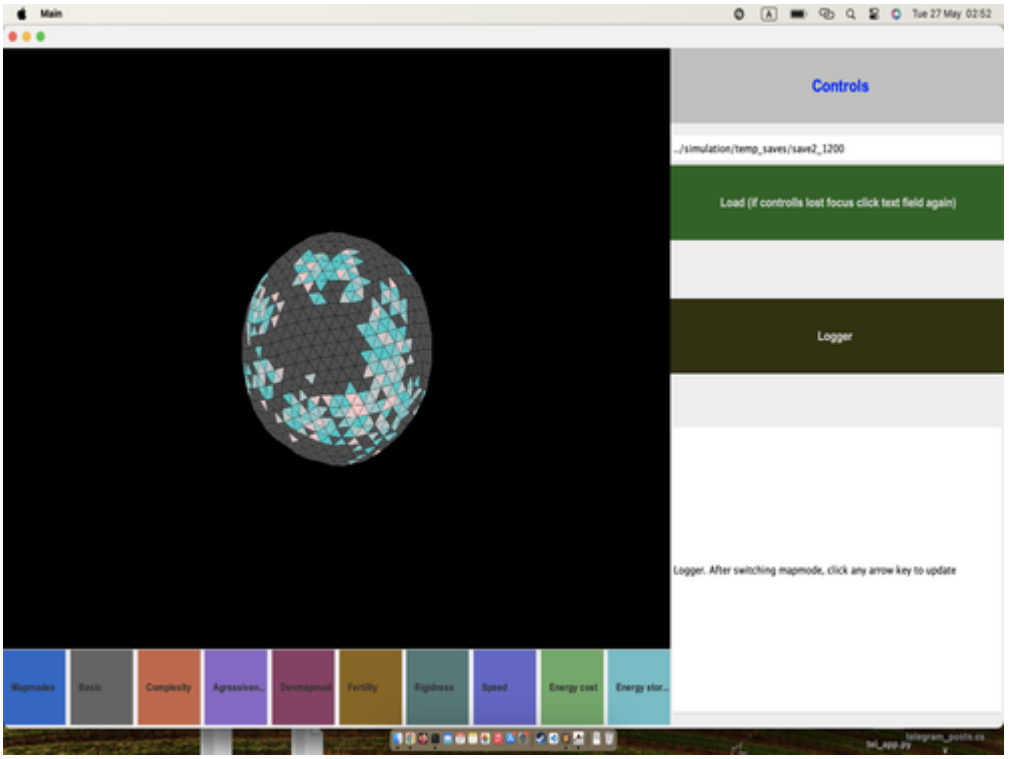


Diagram 5.1C

Here we look at the 1200th turn of the second trial of the simulation. Files are present and can be viewed from Github in the exact same way. Uploading the file to demonstrate the fact that the cells sometimes tend to form large colonial clusters and leave huge eras empty, this is especially evident when we observe the planet from a high polar angle.

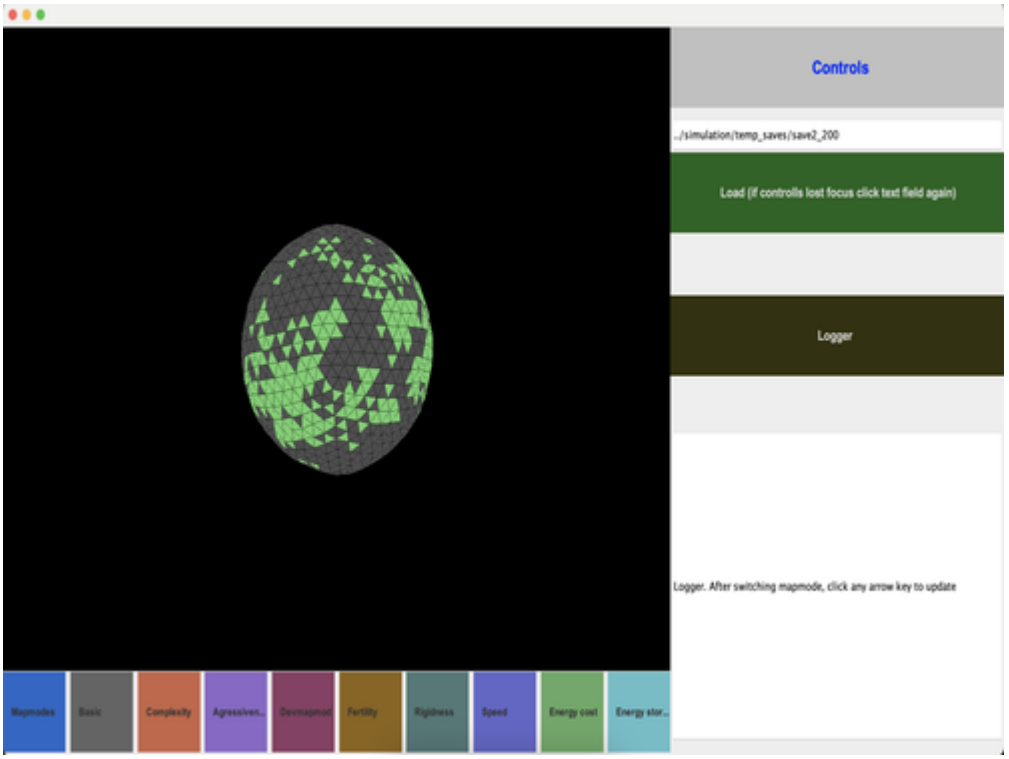


Diagram 5.1D

Last image showing very basic mapmode 1, switch to gamemode 0, if you are interested in only observing an empty icosahedron (triangular tiles 6 per hex and 5 per pentagon).

5.2.2 Visual Inspection Conclusions

We are able to see that over the development period in turns of clustering there exist few eras. First era is the initial era when the creatures spread around the world and populated it. During the first stages there exists a creature “pangea” , a giant cluster of creatures with no gaps from which they spread. The pangea phase lasts up until roughly ⅓ of the planet is covered and usually ends by turn 250. Then a second era comes. Creatures start to die out due to starvation and old age. Holes appear in the pangea, and it collapses into few large clusters. The colonisation of the remaining space continues during the rest of an era. By era 3 the entire

planet is fully colonised, there are major clusters sometimes diagonally interconnected to each other. The structure does not visibly changed for a long time, yet occasionally there are major strikes in stochasticity which occur in correlation to the transformation of developmental quantity into a global qualitative overall change, when a significantly better genome appears and overtakes the population, during that time there exists a long period of anarchy and after the period of anarchy is over continental structure is restored but this time with new creatures. It is similar to development spikes discovered during preliminary analysis and matches my "steps" development theory I talked about earlier. Yet visual inspection of singular is not enough to determine the global processes which happen under the hood. We are confident that continuous development never stops, but without definitive data for all the saves it would be simply impossible to tell.

Therefore I have made a separate Python script to concatenate all the save files into a single python dataframe and to be able to analyze and study long term trends over time. I will show the results below, but beforehand I would like to mention that what would be shown below is only a small fraction of potential analysis that could be done on the data. For future it would be possible to come back to it and study means and standard deviations of these statistics over time as well as concatenate a 3D tensors of 250 saves into 4D tensors of average simulation performance (separately or together with those who lived or died).

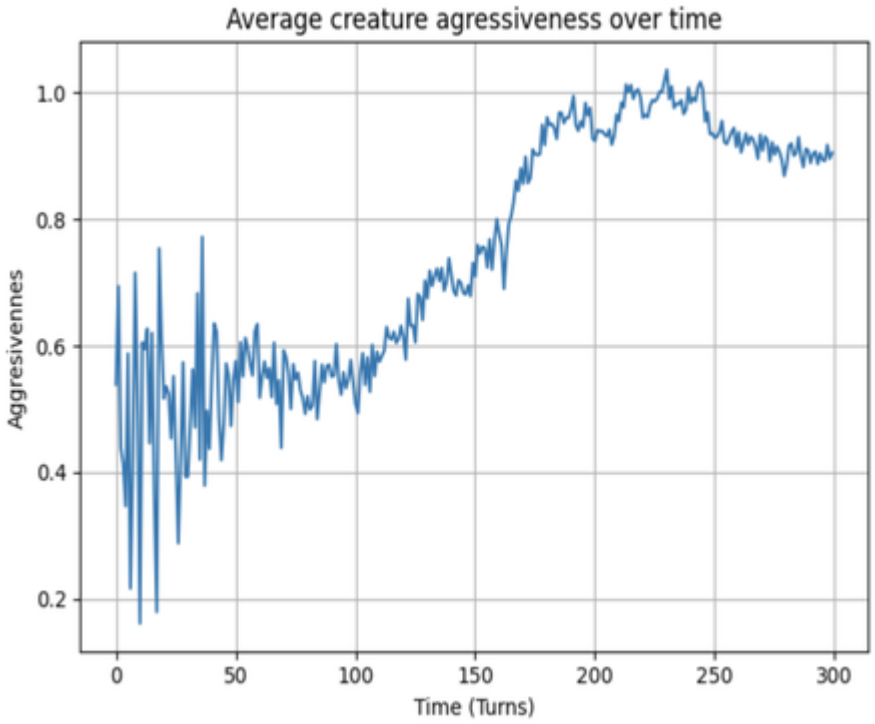


Diagram 5.1B

Average aggressiveness of the trial 1 simulation over the course of 300 turns. As we can see during initial stages (1 era) the values greatly fluctuated due to many potential genome global configurations competing for dominance. Later as the situation settled in place it started growing significantly almost doubling, compared to start. Later it went on a slow decline. This shows us that at first greater aggression is rewarded, but as creatures adapt and learn how to protect themselves the aggressiveness is facing slow but persistent reduction. The aggressiveness metric is the combination of specified DNA predisposition for aggressive action as well as brain decision biases for aggression (developed over the course of the simulation). Aggressiveness ranges from 0 to 1.2 in this simulation but theoretically can reach up to 10.

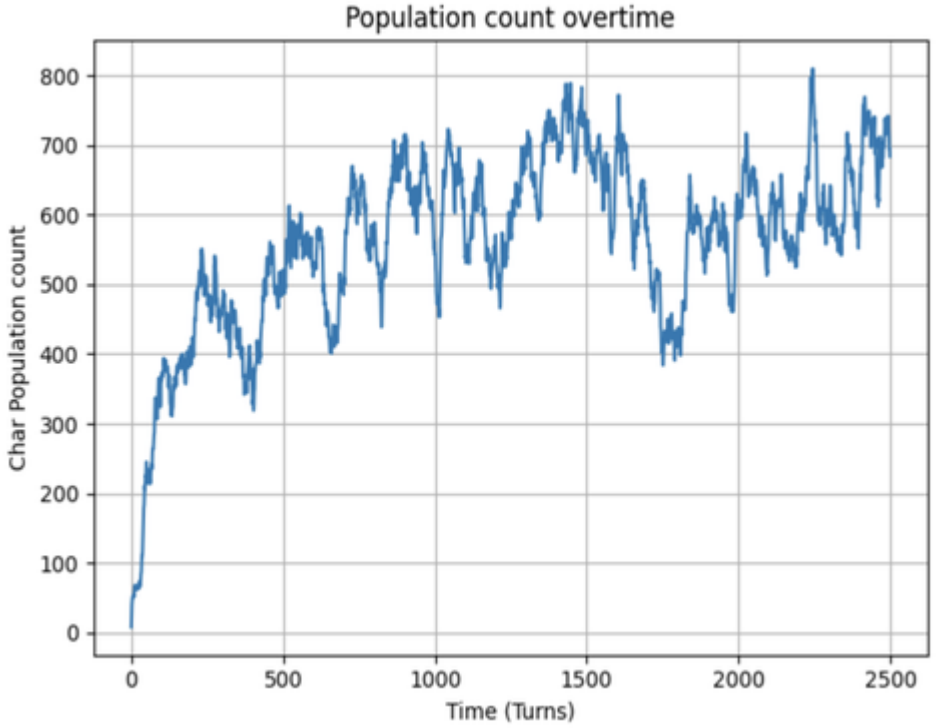


Diagram 5.2B

Here we can see the self explanatory population count over time, for trial 2 with 2500 turns. We can clearly see rapid exponential population growth in the first era, that is later reduced and becomes more tending to oscillation, yet still overall present and positive. It is interesting to note that the population dynamics are always very quick and sharp, there are often rapid spikes of population changes. In a way it resembles a Weierstrass function, with small trends going quickly, as more slow paced general population sinusoidal cycles exist and the super macro trends concerning the entire simulation and being roughly logarithmic. The maximum population is 1620, minimum 0. Such trends can not be explained by the energy function alone, since trends demonstrated by it are radically different and do not resemble logarithmic distribution (see empty_energy_capped_low.csv in simulation folder on Github).

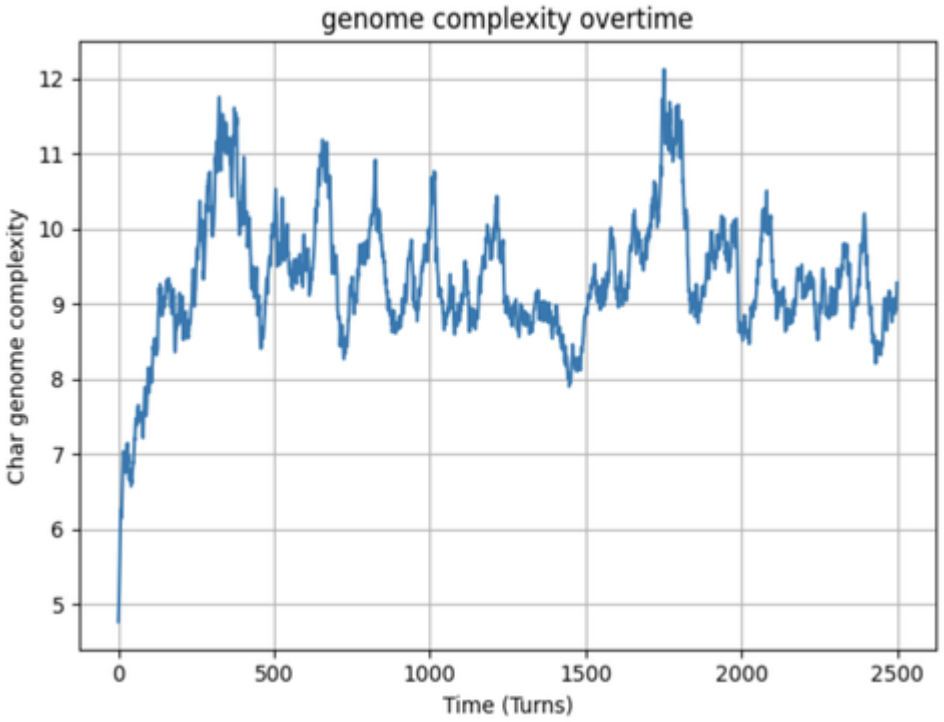


Diagram 5.3B

This diagram depicts the complexity of the genome over the first 2500, the time if the simulation is regretfully insufficient to be able to clearly see anysort of visible trends so far it seems to be oscillating between 8 and 12. Genome complexity is measured by the size of delta of brain activity learning: the smaller it is the less new data affect the creature brain, implying that the creature already learned a lot. Also it is summed with the DNA complexity metric, defined by the number of globally recurrent patterns in the individual genome.

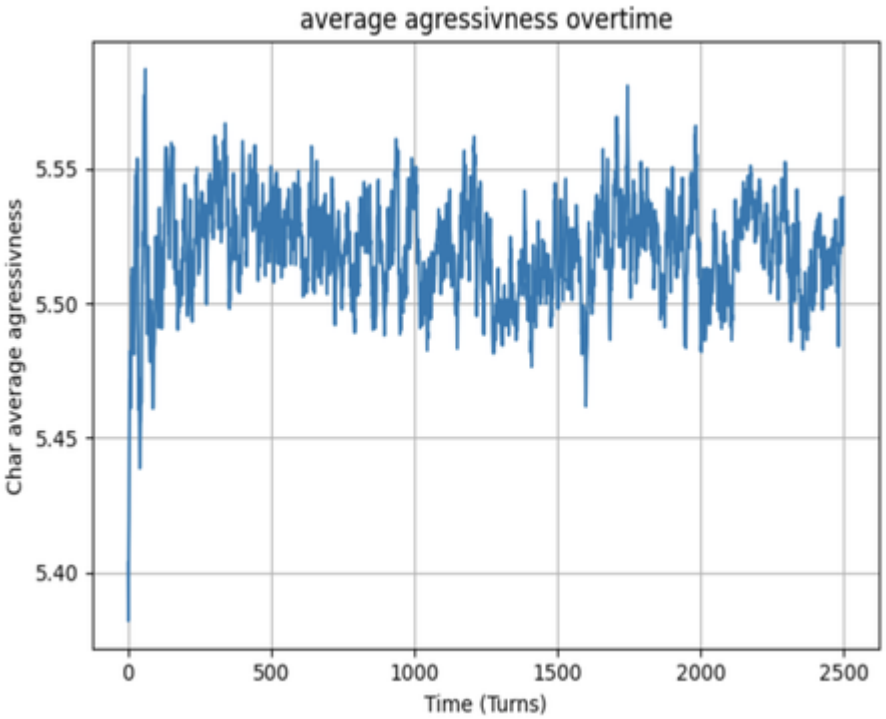


Diagram 5.4B

This is an expanded average aggressiveness graph over time, this is a simulation of a different trial with different starting genomes. Here the situation is quite different, because a higher aggressiveness of 5.4 prevailed. This range turned out to be much more comfortable and the average has not changed much. This just demonstrates how important it is to run multiple trials, since the results and their convergence are so different for the same simulations. I was able to identify an overall sinusoidal trend with surprisingly decreasing x-amplitude meaning that the situation becomes more chaotic overtime, that is an indirect proof that the development and macro processes not only did not reach equilibrium, but are on the contrary speeding up. This is good since it brings us more towards successful completion of our goal. Yet more time and experimentation is needed to turn this evidence into the direct one, or some other metrics (more on this later).

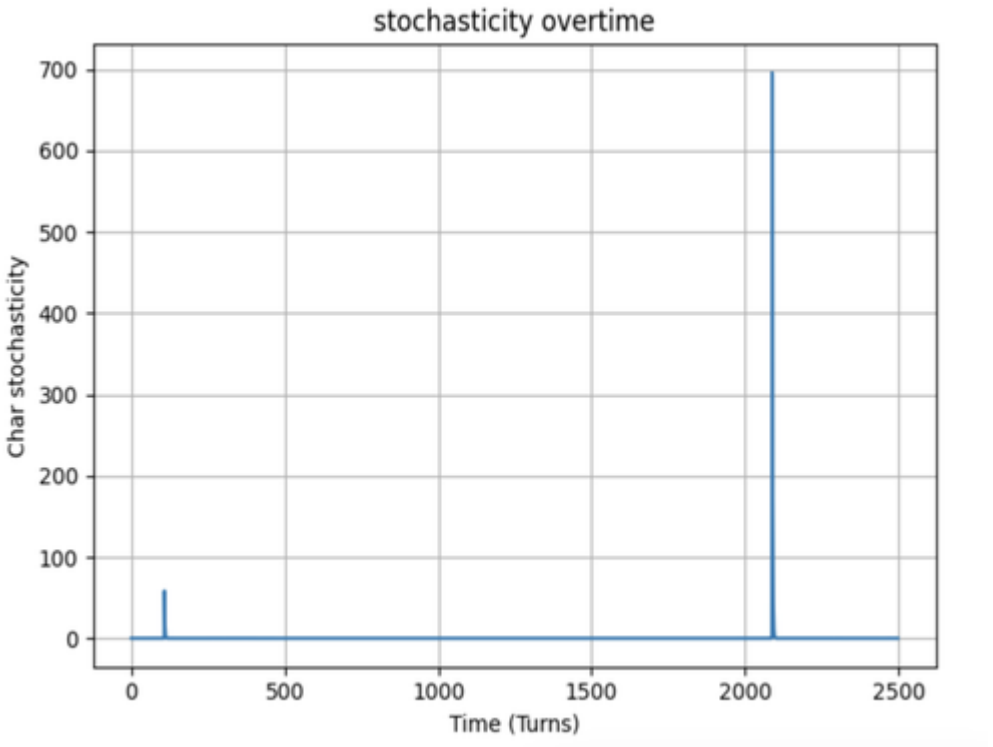


Diagram 5.5B

This graph depicts an average stochasticity over 2500 turns of time. This metric shows how strongly genomes are changing on average as compared to the last round.

We are able to see that there exist two very strong changes of eras at around turn 100 (when pangea is collapsing into continents) and at the turn 2,100 where no visible trends apart from slightly grown fragmentation of continents were visible.

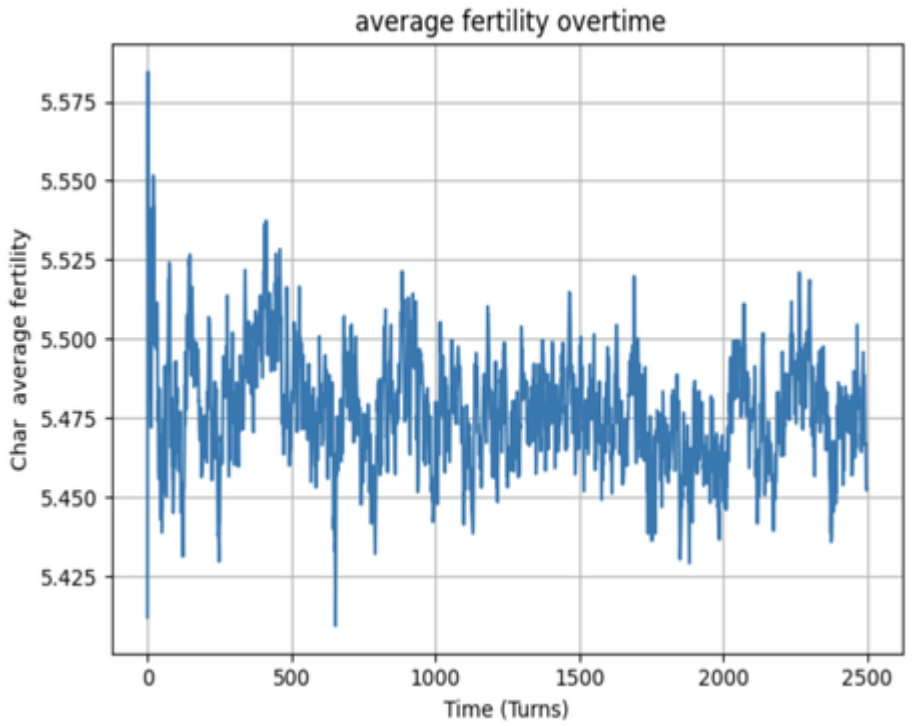


Diagram 5.6B

Average fertility over time for creatures. Used to be calculated from DNA fertility predisposition, attractiveness (to account for the period before enough children are made to judge, later on defactored (from 10 to 0.001) and become negligible).

As we can see, besides the early “bifurcation” (unclearness) period the fertility rate has been declining very slightly with up and down trends locally. Yet the fall is quite insignificant. It is safe to say that fertility remained relatively constant over time. At 300 turns trial the simulation was different as initial starting fertility was lower.

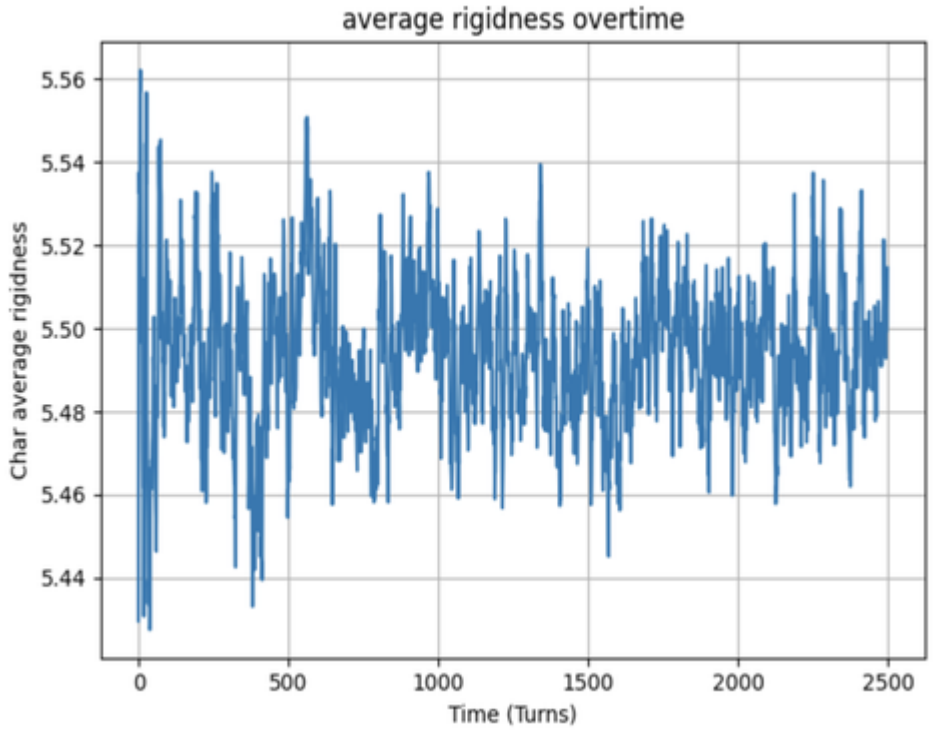


Diagram 5.7B

Is rigidness, strength of the creature, factor by which attack damage is countered, but not counter attack. The counter attack strength is decided by the creature's strength, not rigidness. This parameter was also stuck in the small comfortable range and did not noticeably change. For reference, like for many other values here the rigidness parameter is 0 - 10 inclusive for 0 and exclusive for 10.

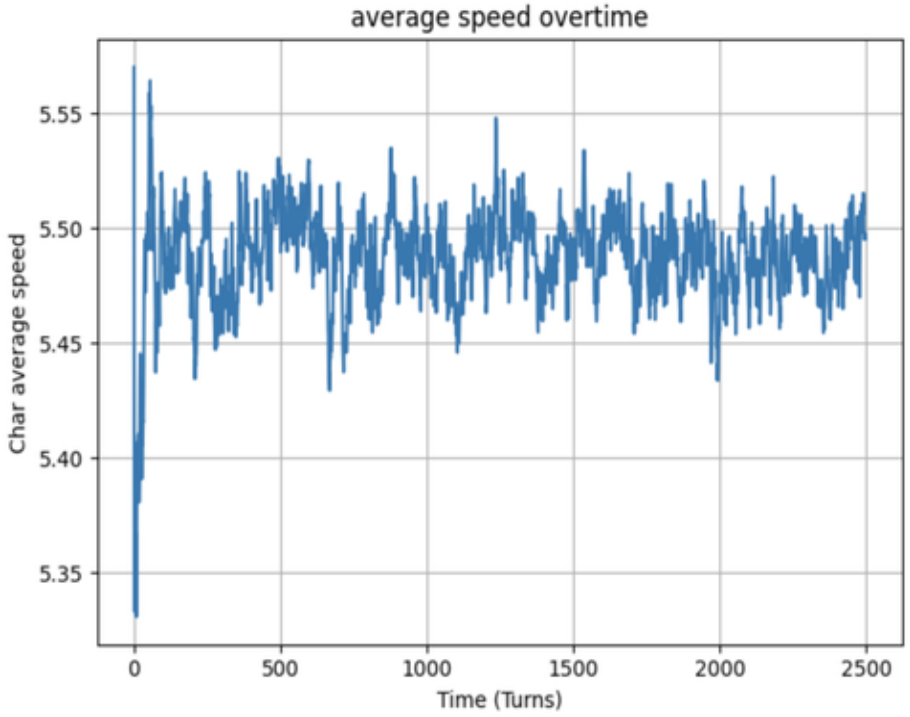


Diagram 5.8B

Is essentially the propensity to move. This graph (like the previous three ones) also do not demonstrate interesting trends as the propensity was changing and fluctuating only within a mere 5% of its total range, meaning that a convenient spot was found, which the actual simulation was able to find a convenient niche, which no crisis were able to break as time went by.

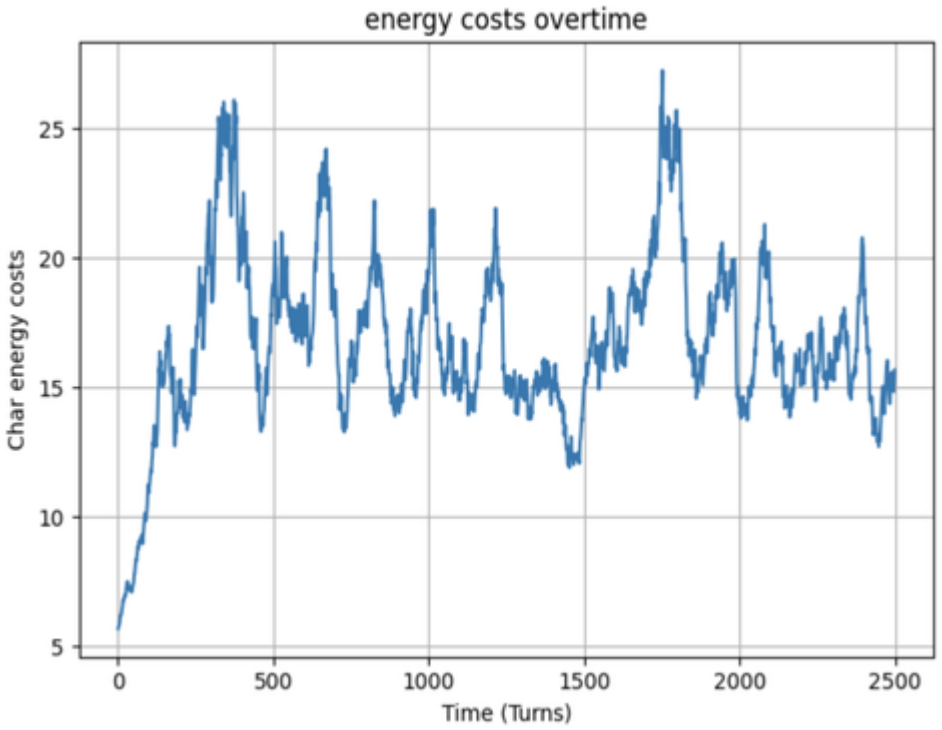


Diagram 5.9B

This is a very interesting diagram, since this time we can see clear trends. An energy cost is how much a creature needs to spend per turn energywise to survive (and do nothing): action dependent energy costs are SREDA-determined factor arrays which are applied to energy cost to calculate a solid spending function, which is deducted from energy storage. Killing a creature means that the task is unfulfillable or the energy is less than 0. There is no maximum value for the energy cost, for sake of our simulation, since some perks have exponential growth functions per additional level, meaning that some setups can cost 1000s and onwards. Obviously there exists a theoretical maximum level, but in short and even medium turn it is barely achievable. As we can see the behavior of the function is spiky; there are short periods

in which energy costs are high, and periods in which they are low. There are very limited periods of tranquility, always a change is present. The temporal continent collapse is hypothesised to be strongly correlated towards the spike in energy consumption function. Since as colonies die more energy is needed to survive on your own. This can imply the existence of colonial symbiosis, since if everyone would have lived by themselves no such spikes would have been happening as colonies collapse. This means that there exists some feeding infrastructure and that overtime cells unite (despite having to prewritten functionality to do so) in colonies. The death / collapse of colonies can be hypothetically attributed (this time not enough data to notice correlation) to the fact that predators find a way to break them and when colonies are rebuilt they evolve a defence mechanism against this predator. This is great, since it involves a continuous self-development cycle of both predators and colonists and according to our model and continued energy consumption spikes this cycle is ongoing and has trends of stopping even by the late 2000th. As we will later, see the overall development is increasing, confirming my line of thought.

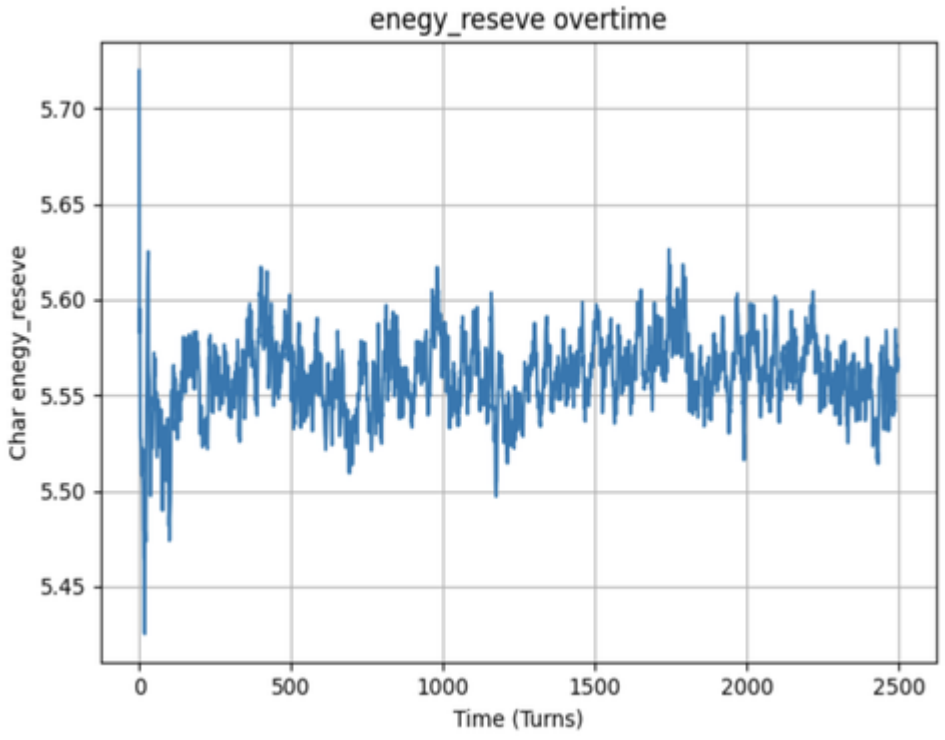


Diagramm 5.10B

This diagram is telling us the DNA propensity to store energy. Unlike previous data it consists of two parts: first part is the said *propensity* which is set to 5 and another is the percentage of

fulfilment of energy storage of an individual cell (that is, how much power a cell has during a given stage, on average). At first the values for it are around 0.75% (cells start with this power when born) and then it fluctuates between 50% and 60% meaning that since they do not fall below during famines. A famine among creatures also reduces the “Gini coefficient” of energy distribution rapidly for the average to stay the same. A rich creature should be even more full with energy whilst poorer creatures should stay at around 0. Example 0.5 + 0.5 = 0.5 average and 1 + 0 = 0 average. The fact can be explained by the fact that when colonies collapse the colony member cells die from hunger (and release some energy as corpses upon their death) whilst the predatory or parasitic creatures (breaking into colonies like a Trojan horse) become extremely well fed.

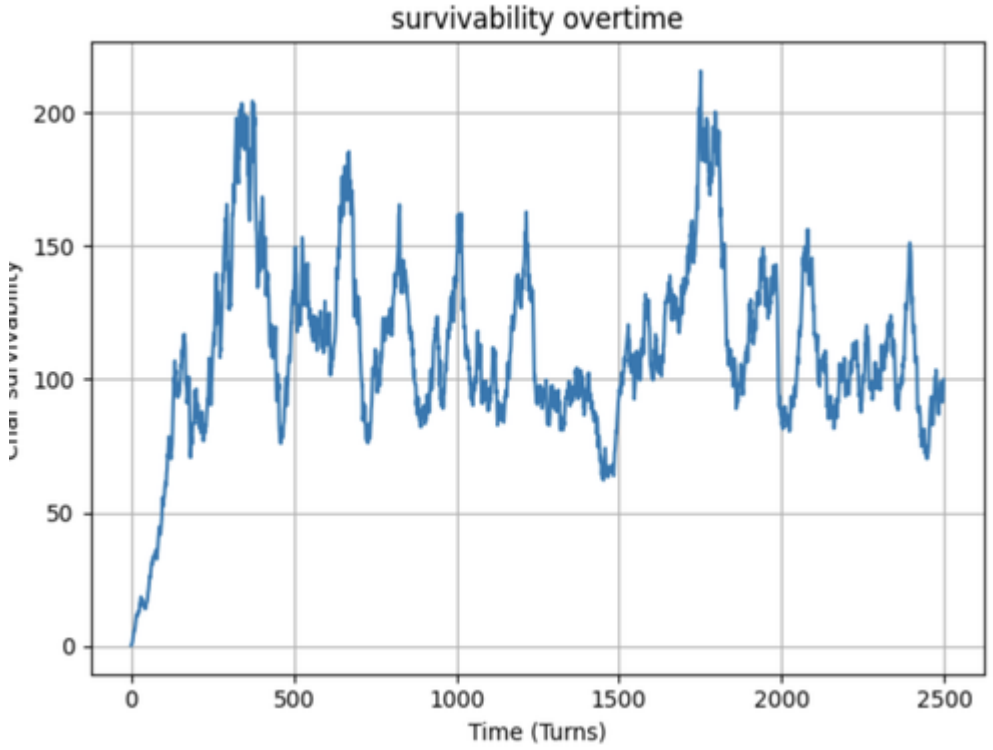


Diagram 5.11B

A very important metric. One of the components of developmental function. Recalling from the previous chapter, it is defined as cell life expectancy (defined as memory length, but is essentially life expectancy since memory length starts at 0/1 and is augmented 1 each turn) together with its energy reserve factor and recurrent DNA patterns. As we can see over time the creature survivability increases rapidly (at start it is not really 0, but since all creatures lived for 0 turn survivability formula yields 0). At a longer distance we see strong spiles at

turn 450, when era 2 changed to era 3 with entire places with living potential on the planet colonised and one at 1750 preceding the early 2000s major crisis.

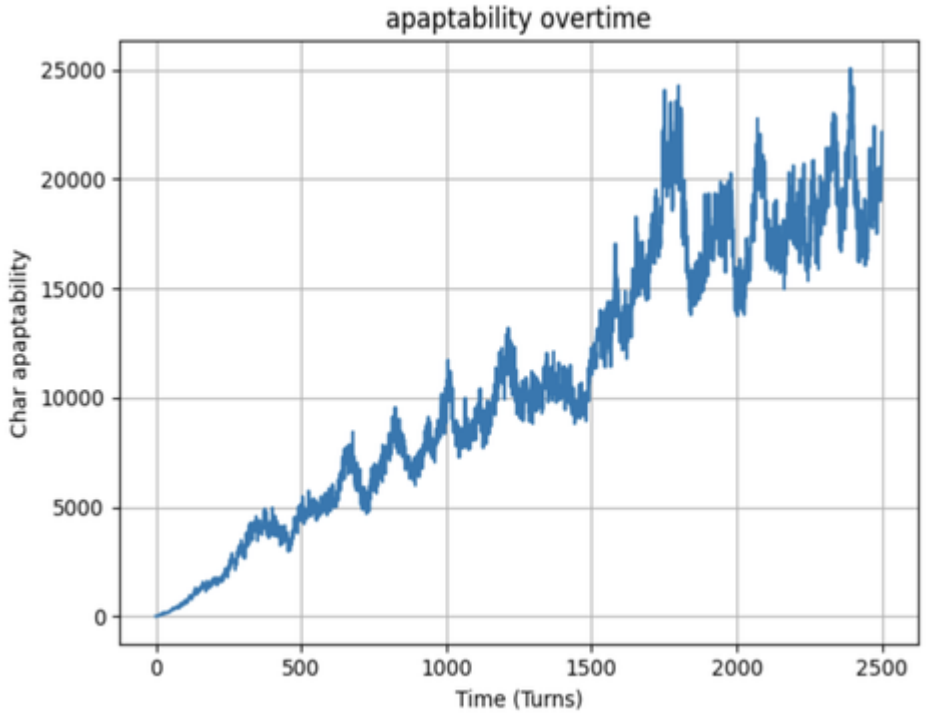


Diagram 5.12B

Adaptability is the measure of average longevity together with an average change of genome per generation and an average change in intellect (intellectual biases per creature). It demonstrates strong upward trends, which however slowed down by turn 1750 (times of last survivability peak). This is quite interesting and a further running of simulation might shed a light on it.

As you can see this metric is unbounded and has large values, so in calculation of the development function it is multiplied by a very small factor, to make roughly similar power gradation with the rest of the variables used.

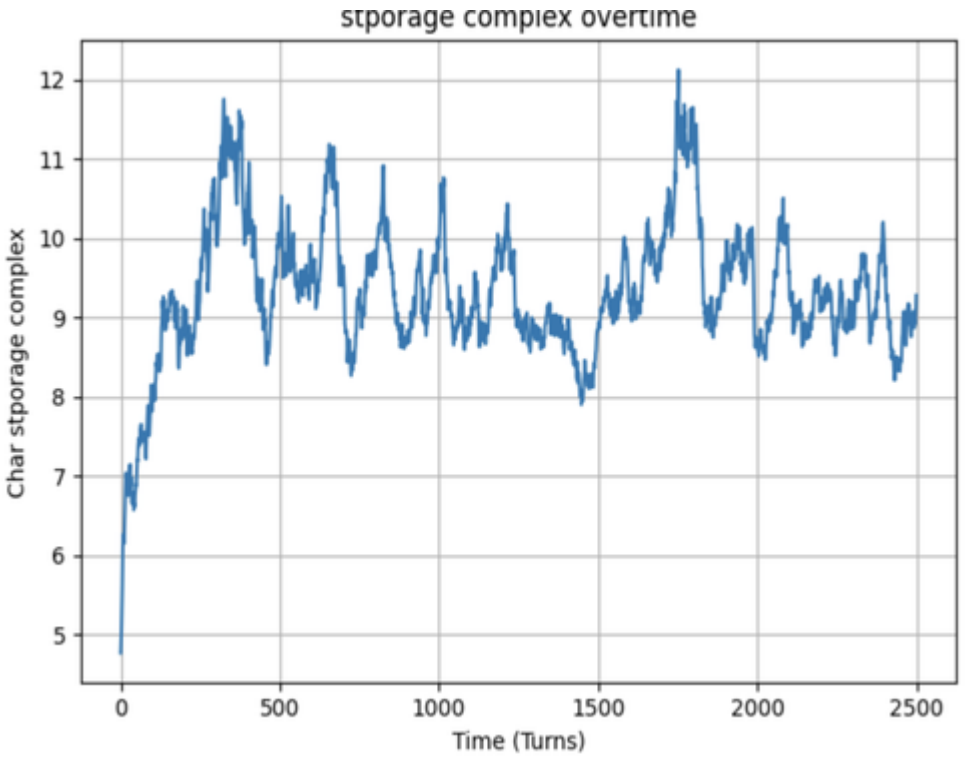


Diagram 5.13B

This is a complicated average storage by the creature, an ordinary storage variable summed with SREDA variables dedicated to interacting with the environment, particularly observing the storage genome and modifying its factor parameters to prevent it from capping and simulation from stagnating. Example if cap is 10 and it is reached the cap is unfactored to $10 \cdot 2 = 0$. This graph permits us to see that SREDA has a strong fluctuating effect of the storage of the genome, no apparent trends are visible (reduction maybe). This reduction can be justified by the fact that since vanilla complexity never outgrowth 6 the SREDA and environment reduce the potential energy storage cap by defactoring it, factoring other more needed and useful perks for this era.

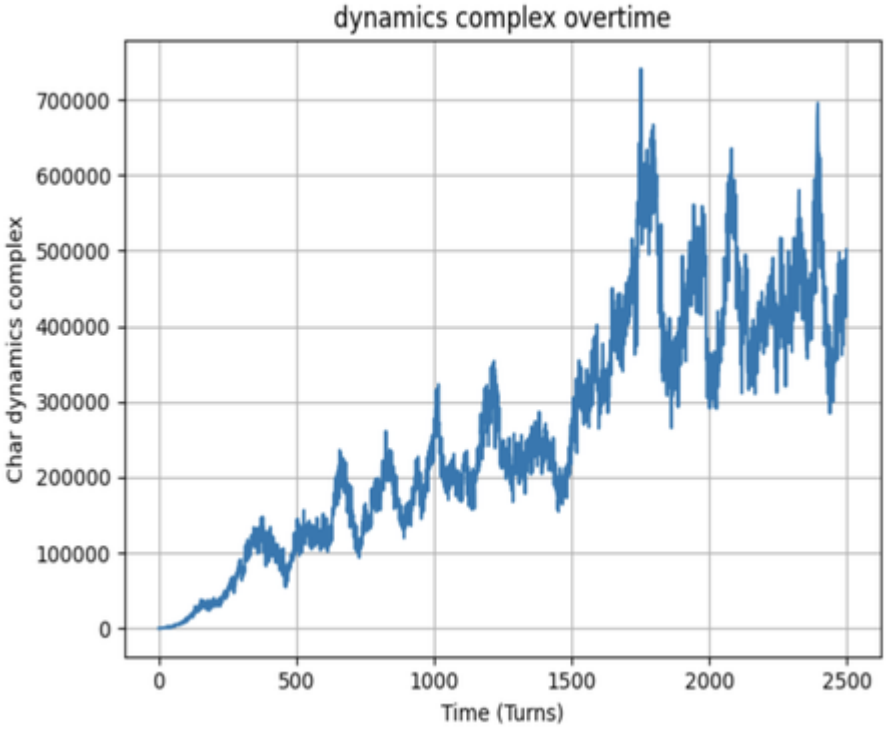


Diagram 5.14B

This graph represents dynamic complexity, a product of adaptability times corresponding SREDA variables (sreda_f). Is not very interesting on its own. Its only usage is to underline some trends which were previously not visible by comparing this graph to the adaptability. In the places of high correlation the adaptability can not be explained by stochastics factors (less of it can be).

And as the difference between this two appears the segment of adaptability becomes more stochastically justified, as sreda_f is directly correlated to overall simulation noise level presence detected (might be confounding with lowest (quickest) tier cycles in sinusoidal function of many rapidly (pseudo Weierstrass type, (in reality all have derivative albeit rapidly changing rara)) functions present fere).

Diagramm 5.15B.1

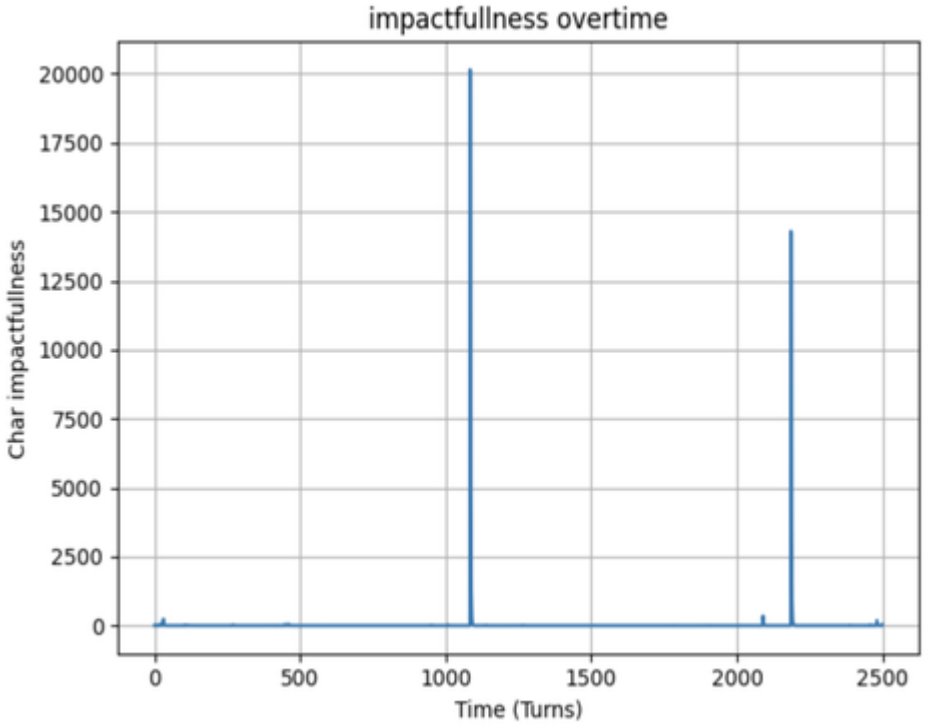


Diagramm 5.15B.1

This diagram measures the impact creatures have on the simulation's environment via their interactions with the SREDA block. We can see that 2100 there is a spike at around the same time there was an adaptability spike on one of the graphs we have studied earlier. There is also a spike at $x = 1100$, but it is not correlated with any previous events. The diagram is difficult to use and for further interpretation of this particular image a log scale would be used:

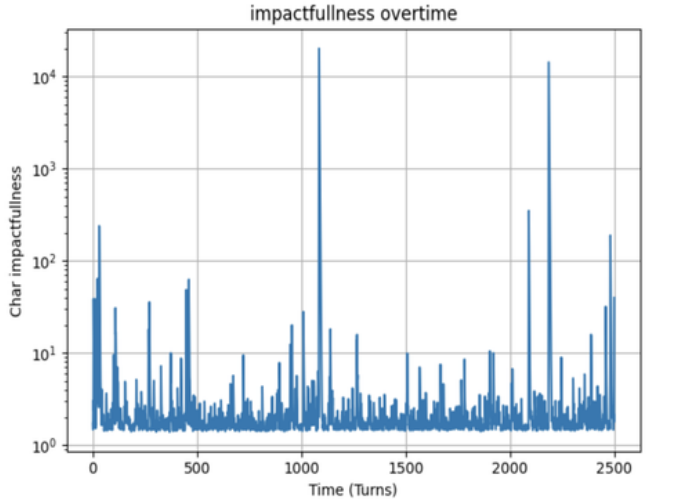


Diagramm 5.15B.2

We ended up also getting rid of the top and button 0.1 percentile in order to get a getter view of the data. The impact of creatures on the environment is constant, but the rate of such an impact rapidly varies with the tendencies of very strong spikes. Unlike with adaptability we were not able to visually see and observe the rapid changes in the rendered class when such spike happened, yet it could be due to the fact that since environment has a large inertia these changes take a long time to come to effect and the hit from spikes is either delayed, or exercised over time increasing the stochasticity of the simulation in the long term (impactfulness and stochasticity are 0.24 correlated linearly and an even better time delayed nonlinear correlation might exists). Creating a tier 1 (quickest) cycle which does not follow a direct pattern like a sinusoidal tier 2 cycle (med) and logarithmic tier 3 (longest) cycles do.

Now that we have described the data let us attach a few files from trial 1 to give a reader a sense of a somewhat (slightly broader picture) as well as some material from uncategorized other trials.

Diagram 5.AUX.1

Cell durability & rigitness over time. Resistance to being attacked. Growth over time.

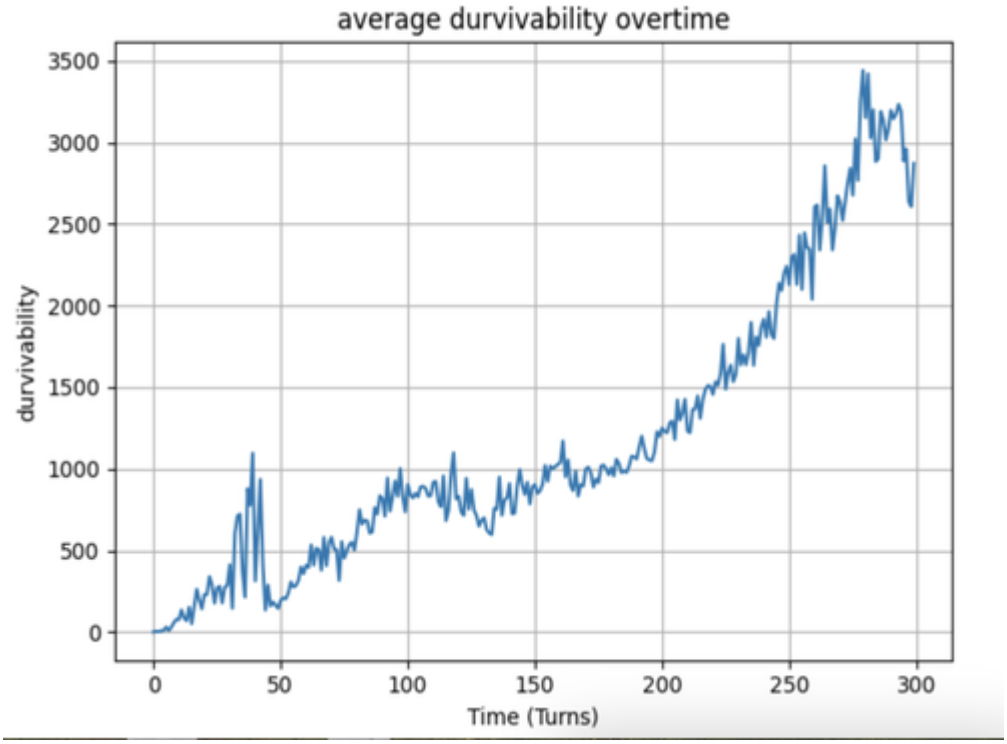


Diagram 5.AUX.2

Cell storage complexity and corresponding SREDA value - fluctuates, stagnates, growth, falls

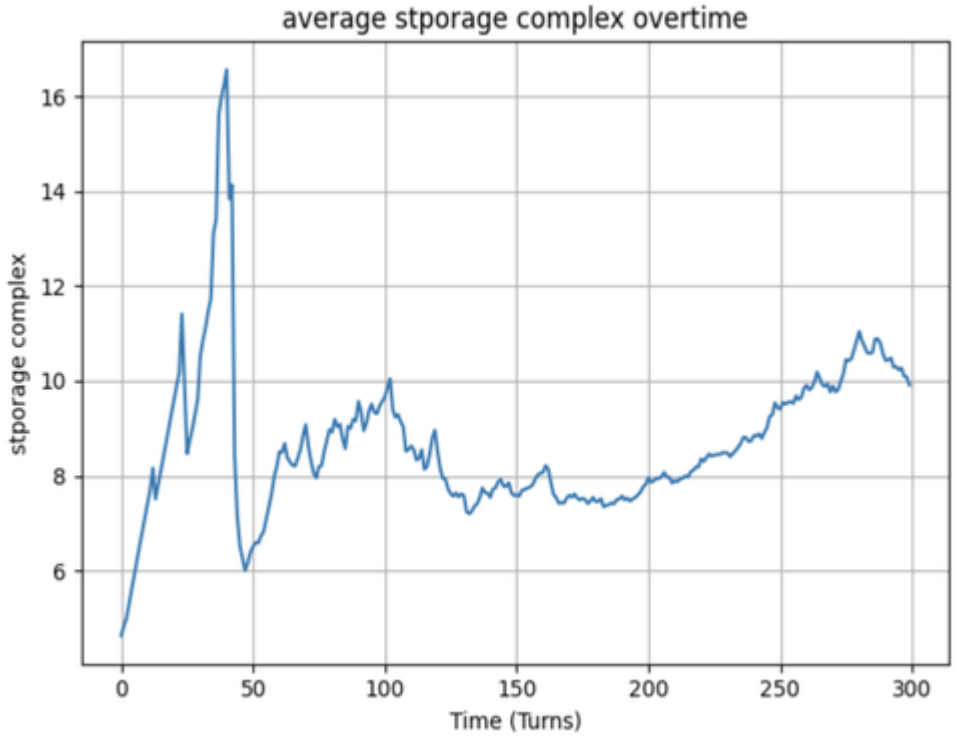


Diagram 5.AUX.3

Cell storage complexity derivative over time combined with memory information and corresponding SREDA block values.

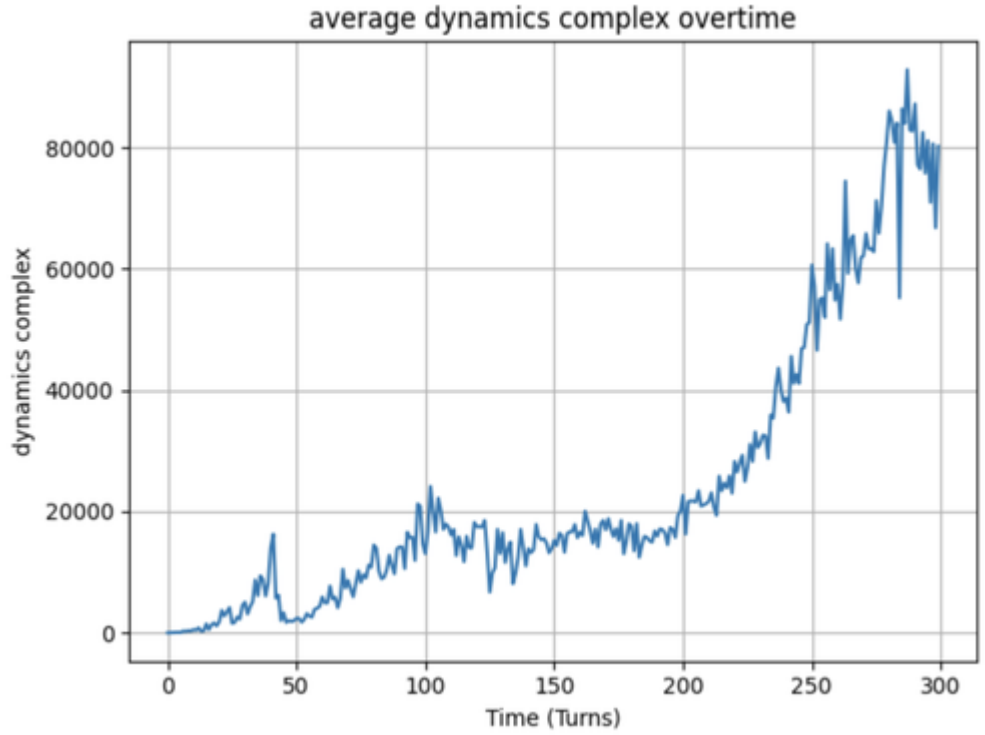


Diagram 5.AUX.4

Average impactfulness - same trend as with 2000 rounds, with strong outliers

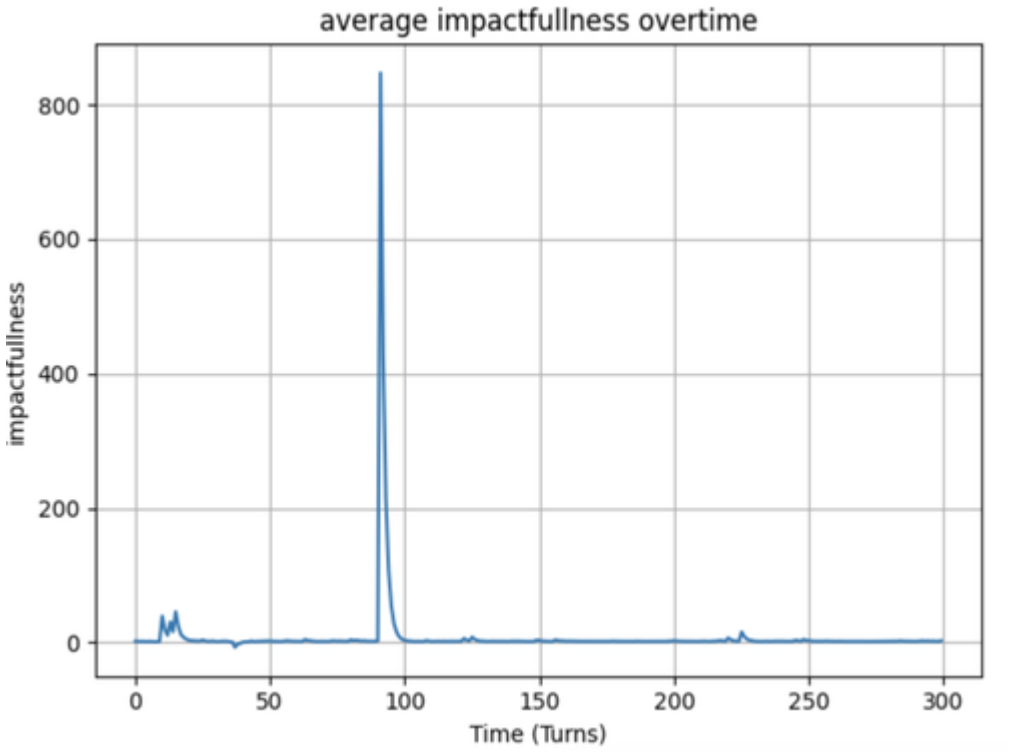
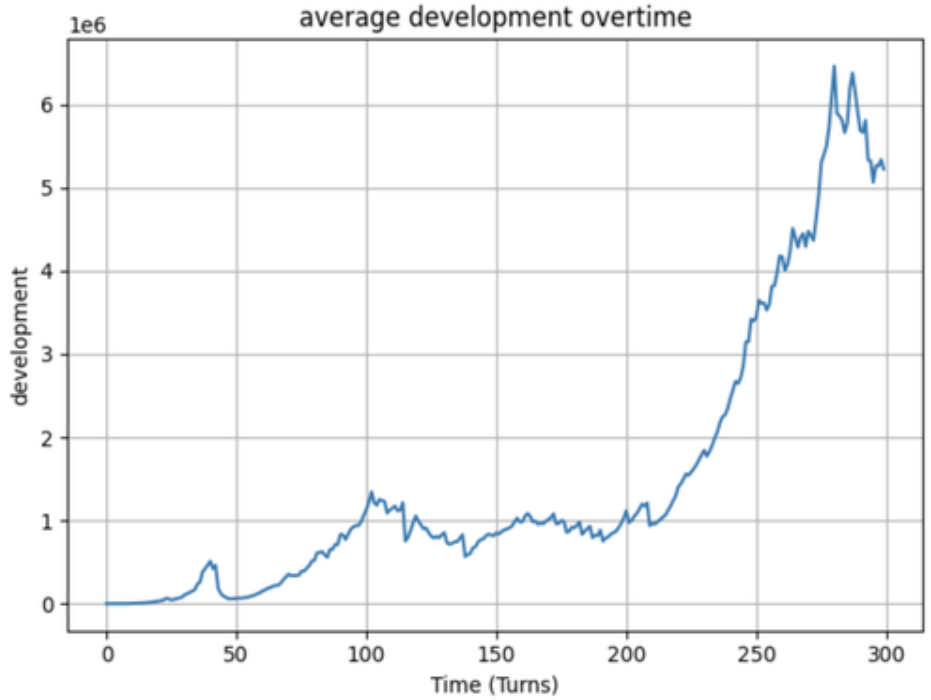


Diagram 5.AUX.5

Final diagram for this section. Notice that the average development is growing.



We have managed to successfully show that the average development overtime is growing in successful simulation and on average. More on that in the conclusion section.

5.3 Potential Limitations and Confounding Factors

Now that I am done with the main segment of the thesis I would like to dedicate some space to discussion concerning the potential limitations and confounding factors in the project. Just like any real experiment there are some biases and imperfections during the execution. Firstly and foremostly it is impossible to computationally simulate try randomness and therefore random processes can only be pseudo mimicked. I have used python and java given vanilla random libraries when I needed to get a random number quickly. And despite now being truly random for our intents and purposes it would be more than enough.

5.3.1 Limitations

A. A first major limitation of our project was the computational time. We were unable to compute the simulation for trillions of turns, which could have been extremely beneficial to understand the asymptotic convergence of such models. Such limitations could have been bypassed by running a simulation on a more powerful computer or using cloud computing services. A risk it produced lied in the fact that we had to assume that the trends that were present during the time of a simulation would continue without any sharp unexpected changes, which was proven to be wrong during the initial testing phases. Asymptotically the algorithm computation times were also frankly quite disappointing the code was taking a longer and longer time to execute, it was unclear whether is it a $O\left(n^k\right)$ with $k > 1$ or $O\left(k^n\right)$ with some small exponent: both are extremely slow in the long turn. But if we assume that the ruleset is irreducible nothing can be done to significantly speed up the computations, at least on their order. Some minor fixes would not change the global fundamental picture, only give a temporary boost and allow the simulation to be computable with a fixed computing resource for a finite number of steps longer.

B. The second major limitation of our project is the existence of a developmental maximum: the desired success was determined to be an existence of continuously self improving developmental function. But not only it can slow down, or fluctuate, what is way worse is that there might exist some sort of developmental maximum, a best configuration that no other could top, or a series of fluctuating best configurations depending on the current environment. My hopes to counter this issue regarding

implementing dynamic creature environmental interaction, but such an interaction might also end up being in a dynamic equilibrium killing the trial, all of them. A potential fix of the issue is the SREDA block aimed at dynamically complicating the environmental situation in order to allow for more and more advanced creatures. Imagine a hypothetical situation where creatures live in a world of ”Flammarion engraving”: they see the stars they want to reach, but as they do the stars turn out to be a mere facade, halting the development of technology in such a system. Instead, the SREDA block is supposed to detect a creature approaching the stars and generate space instead of a facade. Broadly speaking, it is supposed to catch the “attention” dedicated towards a particular segment of the created universe and attempt to make such a segment more complicated, to create a solution to a question posed, which did not have a solution prior. Creatures are supposed to pose “questions” and the environment creates “answers”. Somewhat similar learning principle to the GAN model, but this time there are no winners or losers, just a process of development.

5.3.1 Confounding Factors

Factors that disrupted our projects functioning, potentially unresolved issues that did not directly threaten my findings, but risked making my project more inefficient. Those factors excluded some minor code inefficiencies that remained undetected and only focused on the conceptual level and potential conceptual mistakes made in my project.

A. Issue number one is potential reducibility of a simulation. The overly complicated ruleset I have created for my simulation, could potentially be reduced to a simpler ruleset with same performance, but simpler rules. Due to the complex nature of cellular automatons rulesets, especially when handling genetically modified ones, one can create a gratuitously complicated model, that can be whitelist simplified keeping the utility function behaving in the same way.

B. A second confounding issue lies in the fact that python has a floating point error and our calculations both when it comes to graphics and to a large extent the simulation itself are based on floats. The script that generates faces and vertices for future rendering was also pythonic. This implies the existence of a negligibly small, at least

in the short term, but continuously growing mistake that threatens to create unpredictable monotonously increasing errors that would be difficult to detect and that would potentially hinder creature environment interaction and result in simulation crashing in super long term. In the medium term however (what we covered in the section above) those errors never showed themselves and therefore were practically negligible, that's why they are here and not in the limitation segment.

C. A final confounding factor is unexpectedly cross language interaction. During the project some code was written in java and some in python that created some difficulties when the project was to be united in the single repository. Organisational problems arose with compiling and executing code, primarily with the rendered, it refused to render anything and broke down when I first attempted to run it on an alt repository. Thankfully the majority of these issues were fixed in time, allowing me to pursue other tasks that I considered important and worth solving. A methodology that I used in the process involved using various testing methods, from manual testing output in console with print commands towards using logger, info, error and critical to understand what was going on. A big mistake was designing parts of the project as separate elements from the start. This created an unpleasant situation in which large chunks of code needed to be completely rewritten in order to make sure cross coordination is set up. The python script that generated edges and vertices needed to be completely remade from scratch along with some other aspects of simulation and graphics render. I have also conducted additional research on the existing implementation and 3D Lobachevsky geometry. The information I have found helped me better understand the practical implementations of visualisation algorithms and despite no code from other repositories being used, it was a great reference and inspiration source.

Lastly I would like to mention, concluding practical section, that before moving on towards final thesis conclusion it is important to clarify the following point, I have purposefully made such a strong of an emphasis on potential flaws and limitations of the projects to outline some space for future research - in the future it would be possible to know what parts of the project need improvement and which do not.

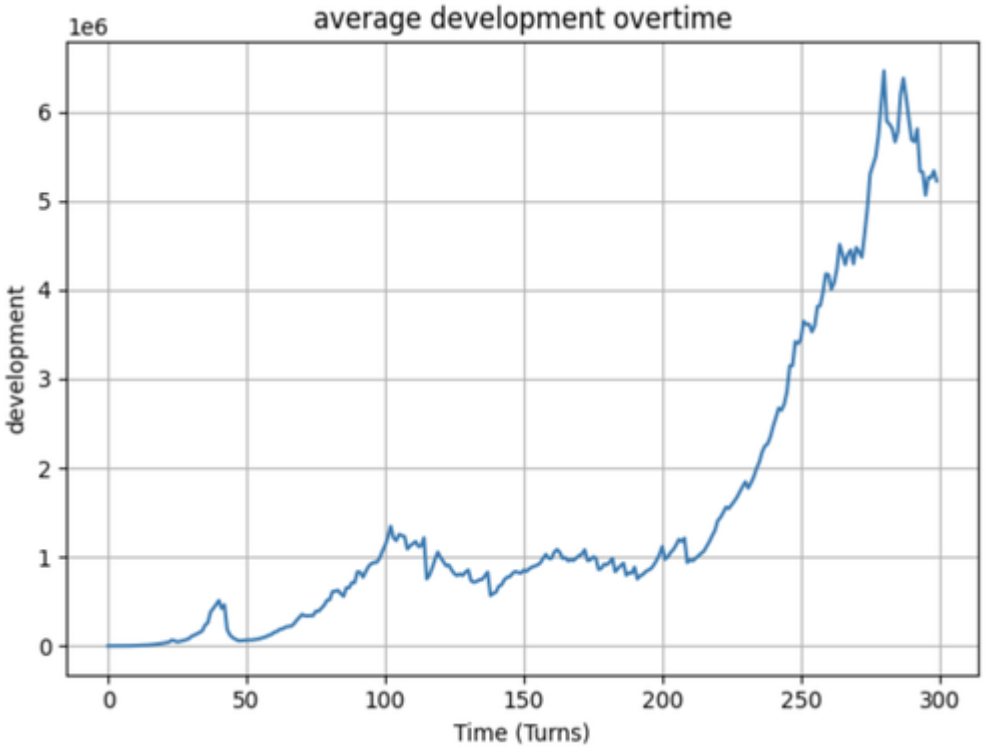
However the matters may go concerning potential confounding factors. I did not in fact encounter a real integer overflow or domain errors, despite running many simulations for a medium time duration. Which makes me conclude that the overall quality of the code is sufficiently adequate and is able to represent the simulation I wanted to represent initially.

5.4 Findings Compilation

In this section I will demonstrate the most important findings from the complex analysis section, that would provide crucial insights into the expected behaviour of an (an aggregated) development function over time.

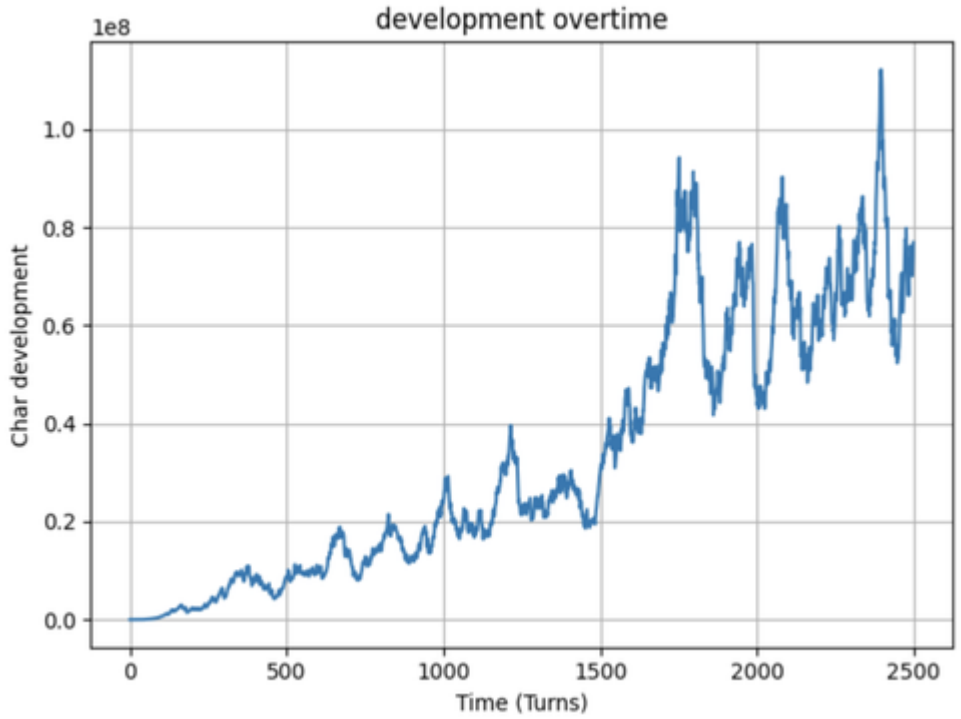
First of all I would like to post two graphs here. An aggregated development function from trial 1 (300 turns) (repost from section 5.2) and an aggregated development function for trial 2 (2500 turns) then I will briefly talk about what this function is (focus more on 5.2) and conclude the section.

Diagram 5.1C



Development over 300 turns. (Old metric, for new one //10)

Diagram 5.2C



Development over 2500 turns.

Development is an overall function defined in Chapter 2 and designed with the intention to measure whether the simulation is developing or not. It is a combination of complexity and adaptability of the cell with the environmental and SREDA factors, it includes the impact creatures have on the environment.

In both cases we are seeing the continuous increase in development. And due to development being an all encompassing complicated function capturing all individual aspects of creature - environment interaction we can confidently be able to conclude our findings, we have managed to create a simulation that was able to enter a continuous state of self-development. We are yet to know where or not it would last forever (probably at some point it is bound to stop.). But nevertheless the experiment is a success since the stopping point is so far away we did not even reach it (some slowdown was visible, but the rate of it was slow).

The future energy SREDA block can be recalibrated in a way to allow real ML problems to be substituted into the simulation. Given the rapid and consistent developmental (metric of

success) trends observed in these simulations, it is reasonable to confidently assume similar positive outcomes could be feasibly implemented in practice, to some extent.

6. Conclusion

6.1 Practical Applications

Since my degree is in “Management and Computer science”, I consider practical applications to be a noteworthy part in my work, since besides doing fundamental research I want to also illustrate the practical real life benefits this research can bring. First of all it is an obvious fact that during the last 10 years advanced AI as a branch of machine learning has seen some rapid development. A notable example being Open AI and its most prominent product Chat-GPT were able to gain the first million users in a matter of days.

This means that there is a very strong demand for powerful AI models on the market. The consumers expect the AI to be able to perform all sorts of tasks and perform them quickly. Currently the AI is limited in its developmental speed, since when classical training methods are used they are extremely power and energy consuming, take a long time and demonstrate clear and strong diminishing marginal returns trends.

Implementing evolutionary based approaches might allow us to greatly reduce the power investment required to train a decent AI model, a cycle of continuous self-development would require much less maintenance as well. AIs are currently faced with a problem of finding new data, whilst avoiding training on its own production for our approach it is not the case, because the information produced by the AI would be interacting with the environment and would not be creating “pollution” in further AI learning.

Such an AI can see a lot of potential usage in small startups which do not have the resources of big corporations. Broadly speaking it would be enough for a small startup company to define the environment and the most basic ruleset and start training the model, even on the personal computer to get the results. One of the advantages of cellular automatons is that they can yield an unconventional and unexpected result and make computations very efficiently, without the need of creating a complicated algorithm. By computational theory cellular automata can not be faster than classical algorithms (given that it is well designed), but the

creation of such a cellular automaton is in general much simpler and requires less expertise, therefore giving this method a strong competitive advantage as compared to classical models.

6.2 Possible Areas of Further Research

Regretfully due to the time constraints and infinite size of potential human knowledge I was not able to cover/research all topics that I found considered to be interesting. Below I will provide a general outline of interesting research questions I got during my work on the project and will briefly outline their perspectives.

A. In this project I have proven that all bounded non-stochastic cellular automatons with finite number of states are bound to be periodic, is the same proof possible for unbounded ones, I was able to prove it for some special cases (like gliders in CGL) but a general proof for all cellular automatons would have (if it is possible) a grand applications in computing theory, since some cellular automatons are Turing complete. There might also exist a link between this problem and the Busy beaver problem, since they both deal with computing machines and their limits.

B. More efficient rendering algorithms. There must exist a way to efficiently render special shapes like Goldberg's polyhedrons. Also studying 3D Lobachevsky geometry and designing a custom rendering engine, which would work quickly and efficiently. Is an interesting and rewarding challenge with some practical applications in gaming graphics design segments of IT.

C. Customizing the code to solve classical ML problems, yes observing the evolution and seeing system development increase as well as complexity is interesting and has future potential practical implementation, but it would be quite important milestone to actually apply this model to some classical ML tasks, like writing texts, making art and generating music, or at least guessing credit score and making consumer behavior predictions.

D. Lastly another interesting endeavour would be to run the existing simulation on a far more powerful computer to see late trends. As we saw on diagrams C10 and C11, the

pseudo asymptotic trends can be deceitful and plausibly given more computation power we could have received deeper insights on the simulation, on how it acts and what happens in the long run. We would be able to more generally derive the expectation of the developmental function, and see whether a hypothesis analogy of ”Gamblers ruins” (Section 3) exists for our (and other) rulesets.

6.3 Closing Remarks

Now the time has come to bring the experiment to a close and summarize the outcomes of all that has been undertaken. In the future it could be possible to continue researching in the aforementioned directions, but for now in this part I am planning to only talk about what was already completely researched and finished.

To reiterate I would like to restate the initial goal of our experiment. A creation of continuously self-improving evolutionary based cellular automaton that would be able to develop on its own. In the early sections we developed a theoretical basis and were able to better express what we mean by a cellular automaton and how we define development. We were able to derive a clear number metric, an agglomeration of them to be precise, that would have an ability to accurately measure the development of a system at any given state of time.

Using our save analysis tools we were able to construct the developmental function over time and heuristically, but performing multiple experiments with sufficiently large populations we were able to estimate this function’s behavior. We considered an existence of at least a fraction of successful simulations to be successful conclusion of our experiment, however I still theoreticized about existence of potential cellular automaton ruin, a situation where under every potential ruleset there would exist a scenario in which an expectation value of an aggregated developmental function monotonically decreases overtime.

Such a scenario would imply that given a long enough time passing the overall development would reach zero. This scenario stands for the situation when the stochastics factors are turned on and are influencing a simulation, because otherwise as we have proven in Theorem 2 the simulation would become positively periodic ($p > 0$). But as we noticed in the

limitations section we were unable to run the simulation for a time period prolonged enough that the macrotrends depicted here would have had a noticeable impact on the simulation.

In tests we have run, we were able as expected to identify cases with the positive trend in developmental function expected value over time. Those were only a fraction of the total trials we have tested, but we managed to strengthen the initial hypothesis concerning the existence of growing developmental functions, since the majority of failures lied in the first 100 steps of the simulation and as we went beyond that the amount of "ruined starts" rapidly decreased. If we would assume that the frequency of "ruined" starts would perform in a similar way, we can conclude in the long run the fraction of starts where developmental function never goes beyond death threshold would be continuously declining, but would never reach zero.

Therefore I would call the research a nice step forward to demonstrate the existence of a rule set for which on a bounded cellular automaton with stochastic function there exists at least one simulation with continuously growing developmental function. Plus we have not tested all the rule sets and we can not rule out a possibility that another ruleset that applied to our case would yield an even better average aggregate expected value function, that would be growing faster, continuously.

Appendix

This appendix aims to provide locations and organisational structure of key files from the github repository as well as their corresponding overview.

Project repositories used

1.
[Name]: Faces.json
[Description]: Preloaded information about the faces of the geodesic icosahedron.
[Link]: https://github.com/SKras2021/Bachelor-Thesis-Files-Skras

2.
[Name]: Vertices.json
[Description]: Array of coordinates of different vertices used by custom rendering engine as reference points, the form of the shape meanwhile is defined by faces. (We render triangles)
[Link]: https://github.com/SKras2021/Bachelor-Thesis-Files-Skras

3.
[Name]: render_core
[Description]: Custom triangle based 3D rendering engine I use for my project, plus icosahedron generation algorithm in math_operation.
[Link]: https://github.com/SKras2021/Bachelor-Thesis-Files-Skras
4.
[Name]: simulation
[Description]: A main simulation takes place here.
[Link]: https://github.com/SKras2021/Bachelor-Thesis-Files-Skras

5.
[Name]: saves
[Description]: A folder with some sample save files
[Link]: https://github.com/SKras2021/Bachelor-Thesis-Files-Skras

Other important notes

1.
[Name]: Project progress logs
[Description]: A folder containing all the intermediate reports sent to the supervisor over the course of the academic year. Starting from the project announcement and ending with a practical implementation log. Includes both architectural documentation as well as design choices justification. Does not document discussions which took place during office hours and emails. IMPORTANT : A LOT OF CODE IS STORED IN THESE NOTEBOOKS AND IS NOT THE PART OF FINAL VERSION IN THE GITHUB REPOSITORY.
[Link]:

References

Books

1. Wolfram, Stephen. *A New Kind of Science*. Wolfram, 2002.

2. Wilson, John H., and Tim Hunt, editors. *Molecular Biology of the Cell. Prob, 2002: A Problems Approach / John Wilson & Tim Hunt*. 4. ed, Garland, 2002.

3. McConnell, Steve. *Code Complete: A Practical Handbook of Software Construction*. Nachdr., Microsoft Press, 2001.

4.Coxeter, H. S. M. *Regular Polytopes*. 3d ed, Dover Publications, 1973. (Chapter 8)

5.Cundy, Henry Martyn, and Arthur P. Rollett. *Mathematical Models*. 2. ed., Repr, Clarendon Press, 1976.

Websites consulted

1. Stack overflow, Eddie Parker & Hkrish, mathematically producing Sphere shaped hexagonal grid. November 23 2017. Accessed February 2025
[Link]:
https://stackoverflow.com/questions/46777626/mathematically-producing-sphere-shaped-hexagonal-grid

2.Weisstein, Eric W. *Dual Polyhedron*. Accessed February 2025.
[Link]: https://mathworld.wolfram.com/DualPolyhedron.html

3. Weisstein, Eric W. *Truncated Polyhedron*. Accessed March 2025.
[Link]: https://mathworld.wolfram.com/TruncatedPolyhedron.html

4. Berto, Francesco, and Jacopo Tagliabue. ‘Cellular Automata’. *The Stanford Encyclopedia of Philosophy*, edited by Edward N. Zalta and Uri Nodelman, Summer 2025, Metaphysics Research Lab, Stanford University, 2025. *Stanford Encyclopedia of Philosophy*,
[Link]: https://plato.stanford.edu/archives/sum2025/entries/cellular-automata/.

5. Elgabry, Mariam, and Shane Johnson. ‘Cyber-Biological Convergence: A Systematic Review and Future Outlook’. *Frontiers in Bioengineering and Biotechnology*, vol. 12, Sept. 2024. *Frontiers*,
[Link]: https://doi.org/10.3389/fbioe.2024.1456354.

6. Weisstein, Eric W. *Cellular Automaton*.
[Link]: https://mathworld.wolfram.com/CellularAutomaton.html. Accessed March 2025.

7. Sapin, Emmanuel, et al. ‘Research of a Cellular Automaton Simulating Logic Gates by Evolutionary Algorithms’. *Genetic Programming*, edited by Conor Ryan et al., vol. 2610, Springer Berlin Heidelberg, 2003, pp. 414–23. *DOI.org (Crossref)*,
[Link]: https://doi.org/10.1007/3-540-36599-0_39.

8. *Cellular Automata*. Accessed 23 March 2025.
[Link]: https://natureofcode.com

Video and Multimedia materials used

Most of the videos are on English, even for those who are not a link to the chanel with adequate english translations is present.

1.- *YouTube*. Accessed Fall 2025.
[Link]: https://www.youtube.com/watch?v=bTQjwh7hLxU

2.- *YouTube*. Accessed Fall 2025.
[Link]: https://www.youtube.com/watch?v=00_wpA3pwjM

3.- *YouTube*. Accessed 1 May 2025.
[Link]: https://www.youtube.com/watch?v=_tdGKfAyRhw.

4.- *YouTube*. Accessed 1 May 2025.
[Link]: https://www.youtube.com/watch?v=vhRxwRr77dk

5.- *YouTube*. Accessed 1 May 2025.
[Link]: https://www.youtube.com/watch?v=Zgb30ixhntg

6.- *YouTube*. Accessed Fall 2025.
[Link]: https://www.youtube.com/watch?v=6M4EgoHh608

7.- *YouTube*. Accessed March 2025.
[Link]: https://www.youtube.com/watch?v=FNNqLseQ94M

8.- *YouTube*. Accessed Fall 2025.
[Link]: https://www.youtube.com/watch?v=4VMk1VgdoCU

9.- *YouTube*. Accessed Fall 2025.
[Link]: https://www.youtube.com/watch?v=pzQUR6xuO8s

10.Simulfie hub - Youtube, accessed Fall 2025
[Link]: https://www.youtube.com/@wallcraft-video/videos

Images and Figures

1.https://math.stackexchange.com/questions/4598316/interesting-discovery-made-in-the-conways-game-of-life

2. Vitaly Kaurov, Starting 3D cellular automata
[Link]: https://stackoverflow.com/questions/12988881/how-to-start-with-cellular-automata

3.Daniel M. How to simulate 3D clouds (questions), used images
[Link]:
https://stackoverflow.com/questions/50848067/cellular-automata-to-simulate-2d-clouds

4.Diosan, Laura & Andreica, Anca & Enescu, Alina. (2017). The Use of Simple Cellular Automata in Image Processing. Studia Universitatis Babeș-Bolyai Informatica. 62. 5-14. 10.24193/subbi.2017.1.01.

5.‘Julia Sets in Julia, Using Interact.Jl - I’d like to Get It Faster’. *Julia Programming Language*, 4 Mar. 2020, https://discourse.julialang.org/t/julia-sets-in-julia-using-interact-jl-id-like-to-get-it-faster/3553 3. (Julia sets were used both an inspiration, as well as during preliminary research)

6.Flammarion engraving. Author unknown. 1888. (Used as illustration)

Additional sources and Miscellaneous references

1.Brugger, Nils. *Nbrugger-Tgm/JRender*. 2021. 1 Jan. 2025. *GitHub*, https://github.com/nbrugger-tgm/JRender. Viewed code for inspiration ideas for my own custom engine.

2.Scanlon, Rob. *Arscan/Hexasphere.Js*. 2014. 20 April 2025, mod 20 May 2025. *GitHub*, https://github.com/arscan/hexasphere.js.

3.*Hexasphere.Js*. https://www.robscanlon.com/hexasphere/. Accessed May 2025. [ONLY USED AS REFERENCE]

4.‘CyberBiology’. *VK*, https://vk.com/public167044850. Accessed May 2025.

5.‘GeeksforGeeks’. *GeeksforGeeks*, https://www.geeksforgeeks.org/undefined. Accessed May 2025. Multiple articles about python and data science.

Some additional minor references which were not a part of the thesis proper and were only used occasionally during creation of progress logs or preliminary simulation are double listed both here, just in case and in the corresponding progress log files, since progress logs are indirectly part of a thesis as well.

Spring 2025, Rome, Italy, Earth.

The end of the thesis work