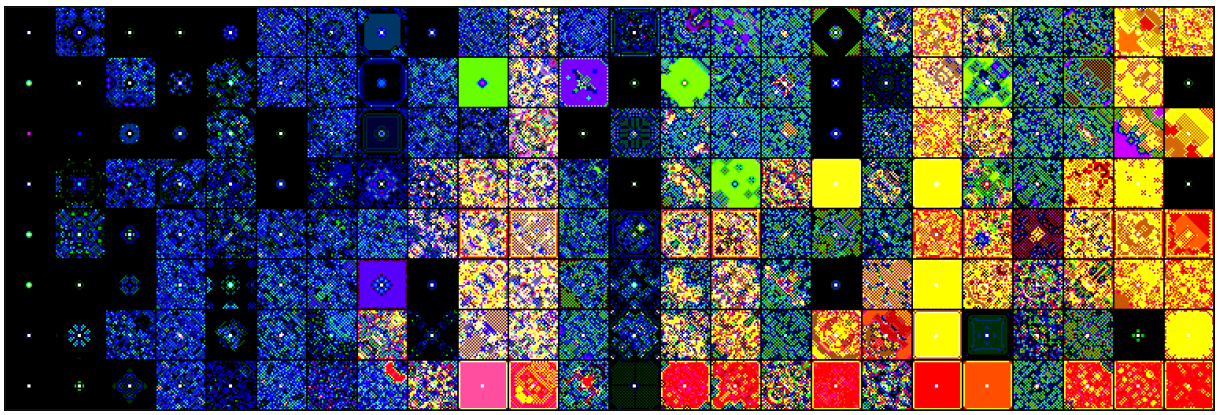


LABORATORY OF BABEL

Explorations of Emergent Virtual Structures

Thesis submitted for the degree of Doctor of Philosophy
Goldsmiths, University of London

Lior Ben-Gai
July 2024

For supplementary materials, algorithms and software, please visit: utomata.net/labofbabel

Direct links are also available throughout the thesis via the icon.

Abstract

Cellular Automata (CA) are discrete, grid-based computational models where simple local rules generate complex, emergent patterns. Despite decades of research, the vast potential of CA remains largely unexplored, with countless possible algorithms as yet undiscovered. This work introduces novel methods for open-ended exploration of CA, expanding the scope of inquiry to reveal virtual phenomena that might otherwise be overlooked.

A core contribution of this thesis is *Utomata*, a new computational framework designed for exploration and study of a wide range of CA. Its versatility is demonstrated through case studies featuring both established and novel algorithms. Building on this foundation, the thesis presents *Spatial Mapping*, a high-level exploration method, accompanied by a custom online software implementation. This method enables systematic navigation of the combinatorial space of variations for any given algorithm.

Central to this thesis is the introduction and in-depth study of *Type-U*, a newly identified CA family. Using the frameworks and methods developed, this study reveals distinctive properties of Type-U algorithms and their combinatorial spaces, supported by qualitative analyses. The thesis concludes by discussing the broader implications of this open-ended approach for the study of virtual phenomena and outlining avenues for future research.

Keywords: *Cellular Automata, Computational Arts, Artificial Life, Emergence, Algorithmic Exploration, Computational Modelling, Structure Synthesis, Procedural Content Generation, Abstraction, Functional Programming.*

Table of contents

Acknowledgements

I am deeply grateful to my supervisor, Dr. Tim Blackwell, for his expertise, boundless curiosity, and above all, his patience. His guidance and support have been invaluable at every stage of this project.

I would also like to thank my former mentors and colleagues at Goldsmiths — Dr. Mick Grierson, Dr. Theo Papatheodorou, Dr. Simon Katan, and Dr. Rebecca Fiebrink — for generously sharing their insight and experience during the early phases of this research. I extend my sincere appreciation to the Department of Computing at Goldsmiths for fostering an environment where I could pursue my passions freely and without compromise.

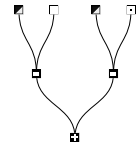
I am also grateful to the institutions and conferences that have provided platforms for presenting and discussing various aspects of this work: Print Screen Festival (2019), the University of Applied Arts Vienna (2020), the SEA Conference at Haifa University (2021), Davidson Institute (2022), Shenkar College of Design (2022), ISVIS – The Israeli Visualization Conference (2023), and the IED Summer School at the Royal College of Art (2024).

Lastly, I wish to express my admiration for the pioneers and contemporary practitioners in computer science, computational art, and artificial life. I imagine a future where discoveries in these virtual realms will not only complement but perhaps even surpass those made in the physical realm.

This work is dedicated to my parents, Yuval and Tehilla Ben-Gai. It is a distillation of my unique blend of qualities and flaws as a thinker and creator. In more ways than one, this is all your doing.

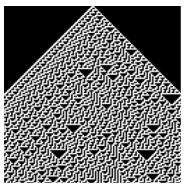
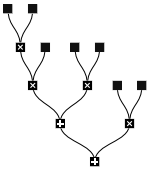
Lior Ben-Gai
July 2024

Index of Algorithms

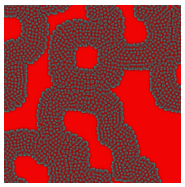
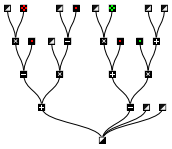
3.4.1
Conway’s Game Of Life
Adaptation of Conway’s Game of Life.

View in Editor Explore in Lab

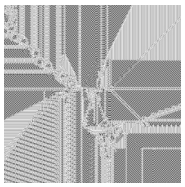
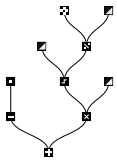
3.4.2
ECA Rule-30
Adaptation of Wolfram’s Elementary Cellular Automata.

View in Editor Explore in Lab

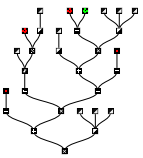
3.4.3
Grey-Scott Reaction-Diffusion
Adaptation of Gray-Scott Reaction-Diffusion.

View in Editor Explore in Lab

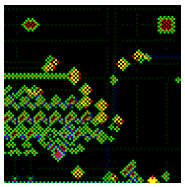
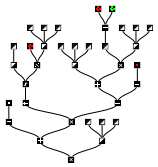
6.1.1
Type-C

View in Editor Explore in Lab

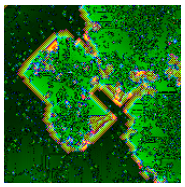
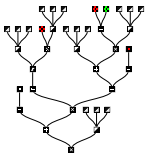
6.1.1
Digger-Dagger

View in Editor Explore in Lab

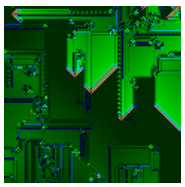
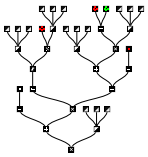
4.4
Red Nose Hexagliders

View in Editor Explore in Lab

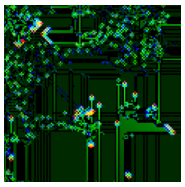
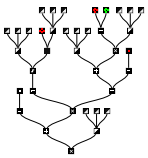
4.4
TC8 (Wildfur)

View in Editor Explore in Lab

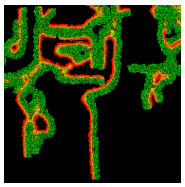
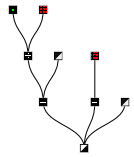
4.4
TC5

View in Editor Explore in Lab

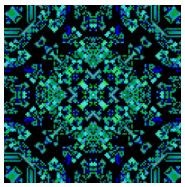
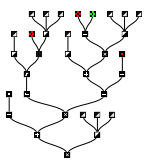
4.4
RNX2

View in Editor Explore in Lab

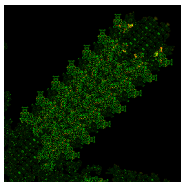
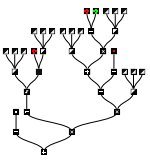
4.4
Fireworm

View in Editor Explore in Lab

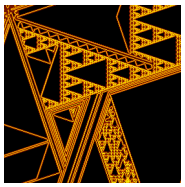
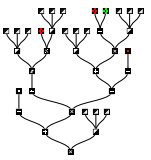
4.4
RNX4

View in Editor Explore in Lab

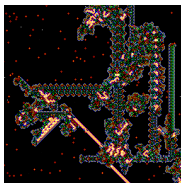
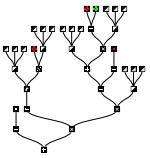
4.4
TC15

View in Editor Explore in Lab

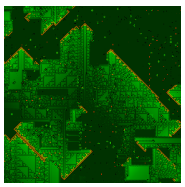
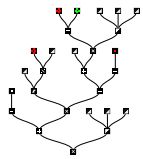
4.4
TC12

View in Editor Explore in Lab

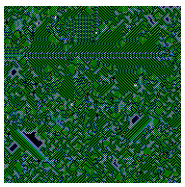
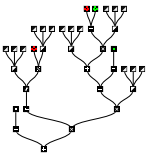
4.5
TC9

View in Editor Explore in Lab

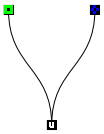
4.5
TC8B

View in Editor Explore in Lab

4.5
TC13

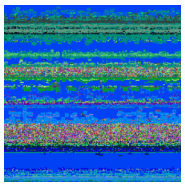
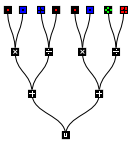
View in Editor Explore in Lab

6.1
TU0

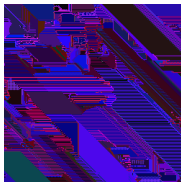
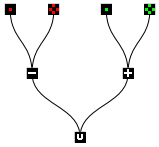
First-order Type-U

View in Editor Explore in Lab

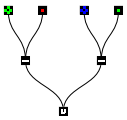
6.1.1
TU81
Third-order Type-U.

View in Editor Explore in Lab

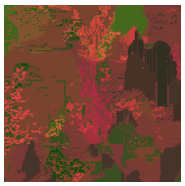
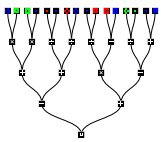
6.1.1
TU82

View in Editor Explore in Lab

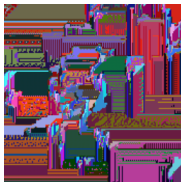
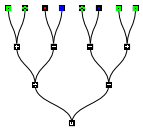
6.3.2
TU59

View in Editor Explore in Lab

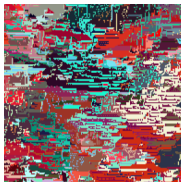
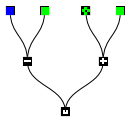
6.4
TU57

View in Editor Explore in Lab

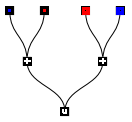
7.1
TU8

View in Editor Explore in Lab

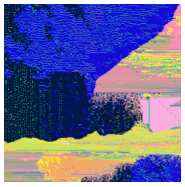
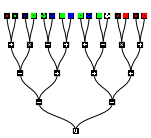
7.2
TU72

View in Editor Explore in Lab

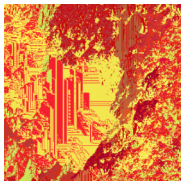
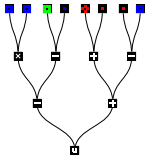
7.3
TU70

View in Editor Explore in Lab

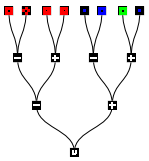
7.4
TU76

View in Editor Explore in Lab

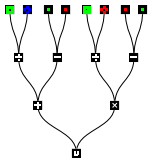
7.5
TU37

View in Editor Explore in Lab

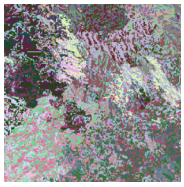
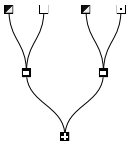
7.6
TU78

View in Editor Explore in Lab

7.7
TU79

View in Editor Explore in Lab

7.8
TU68

View in Editor Explore in Lab

List of Figures

1.Introduction

1.1. A Total Library

> "The impious maintain that nonsense is normal in the library and that the reasonable (and even humble and pure coherence) is an almost miraculous exception."
> [1]

The Library of Babel by Jorge Luis Borges [1] describes a vast physical library that encompasses the assortment of all possible books. This fictional short story from 1941 is narrated by one of countless librarians who wander the library's endless halls in a literal search for meaning. When attempting to imagine such a vast combinatorial space, one is immediately struck with both a sense of awe and cosmic terror. [1] A *total library* would hold the complete documentation of everything that has ever happened, told by all possible authors and in all possible genres. It would also hold the complete documentation of everything that did NOT happen, as well as all things that cannot happen, but can be described. However, since this particular library is randomly sorted, all of these stories, along with every other coherent utterance will forever be lost in a virtually endless ocean of books containing nothing but utter nonsense. The frightening scarcity of meaning in such a space can be experienced by visiting Jonathan Basile's 2015 online recreation of the Library [2], which offers a poetic and daunting glimpse into this literary abyss [3].

1 A term popularised by H.P. Lovecraft, describing the fear of the unknown vastness and indifferent universe, highlighting humanity's insignificance.

The *Laboratory of Babel* is a thought experiment envisioning a total library of Cellular Automata (CA) algorithms. CA are among the earliest computational models to exhibit nonlinear dynamics [4], [5]. They typically consist of a finite grid of cells, each assigned a numerical value. The cells continuously update their values based on the states of their neighbouring cells, leading to complex interactions that can give rise to higher-order forms and processes. The inherently chaotic and nonlinear nature of CA renders their behaviour highly unpredictable, and the range of phenomena they can produce remains largely unknown.

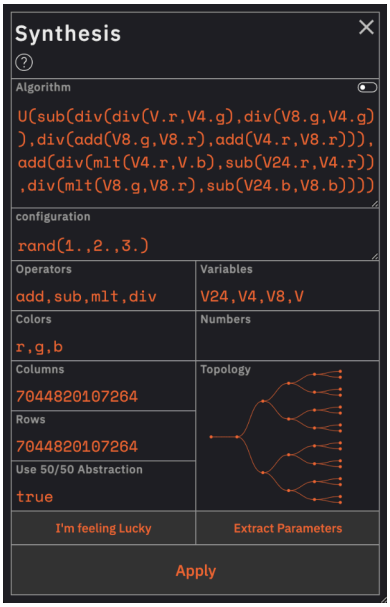


Figure 1.
Synthesis panel in Utomata Lab, featuring a 4th order Type-U algorithm.

Similar to Borges' library, the Laboratory of Babel would follow strict rules for algorithm formulation; like books, the number of possible, valid, and finite algorithms would be unfathomable. It is therefore not unreasonable to assume that almost all will result in utterly incoherent or vacant behaviour. Yet, within this endless ocean of nonsensical computational processes, there must also be countless *"miraculous exceptions"* taking the form of persistent, complex structures that exhibit properties unlike any ever observed in the physical realm. Over the past 80 years of research, many CA algorithms have already been articulated and studied, but due to their sheer number, most will never be. Yet crucially, unlike the Library of Babel, this laboratory need not be randomly sorted.

1.2. Open-Ended Exploration

In 1673, Antonie Van Leeuwenhoek created a small spherical lens using a simple, yet novel technique of his own design. This new instrument allowed him to observe a microbial life form for the first time and, in-effect, to become the "father of microbiology" [6]. Leeuwenhoek had not intended to discover any life

forms. He was a fabric tradesman and lens hobbyist who wanted to obtain a closer look at his merchandise; his work may be regarded as exploratory rather than scientific, especially considering that until that point in time, the scientific study of microbial life did not exist.

Exploration and scientific enquiry are tightly coupled but they do not necessarily overlap. If science is concerned with explaining and predicting observable phenomena, then exploration is concerned with obtaining new phenomena to observe. Both a scientist and an explorer employ speculative thinking, but the former strives to minimise its scope in order to derive new knowledge about a particular phenomenon, whereas the latter often strives to widen it in order to discover as-yet-unknown phenomena.

The advent of computational modelling in the 20th century has brought about a revolutionary shift in science by providing a level of insight and predictive capability to the study of nonlinear systems that would have otherwise been unattainable [7]. The process of explaining and predicting the behaviour of physical systems has become so influential that it is often synonymous with the process of implementing nonlinear systems in software. Consequently, computational modelling is predominantly perceived as a representational endeavour, as evidenced by its name.

For example, the utilisation of CA as instruments for scientific modelling has been particularly fruitful in studying various natural phenomena, including physical [8], biological [9], and chemical dynamics [10], [11]. This application has provided valuable insights and predictions for real-world phenomena in situations where direct observation or analytical descriptions were impossible or impractical. While such algorithms have also been extensively incorporated into creative media, many of them were initially devised as scientific instruments: either as simulations of physical systems or as tools for solving well-defined problems. Consequently, it can be argued that the exploration of CA algorithms, like many other computational models, has been primarily guided by their instrumental value.

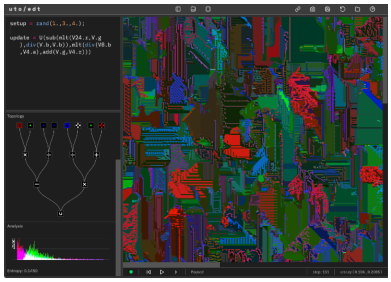


Figure 2.
Type-U algorithm in Utomata Editor.

The field of Artificial Life (AL), initiated by Christopher G. Langton in 1987 as the study of “life-as-it-could-be” [12], serves as a conceptual foundation for this research project. Langton envisioned AL as a synthetic approach to biology, whereby living systems would be assembled rather than taken apart. As with many interdisciplinary fields of research, AL has struggled to achieve a sustainable balance between empowering creativity and maintaining scientific rigour [13]. Early work in AL has largely focused on attempts to synthesise novel phenomena, rather than to analyse observable phenomena. It thus stands out as a notable exception to the aforementioned traditional view of computational modelling.

Within AL there are two prominent schools of thought: proponents of strong-AL, including Langton, contend that life is a property of form, not matter. Consequently, they assert that living things may literally be realised within virtual environments. On the other hand, proponents of weak-AL argue that living things can, at best, be simulated. While this distinction has been a source of great debate from the very first days of AL [14], both schools of thought ultimately agree on and employ a synthetic approach to biology — a scientific study of living systems through the use of computing technology.

While this work draws significant inspiration from AL in regards to exploration of emergent phenomena, this project aligns with neither the strong nor weak school of thought. Instead, it only argues that a non-instrumentalist approach may be advantageous for open-ended exploration — a research approach that

encourages continuous and dynamic investigation of systems without predefined goals or endpoints, enabling the discovery of novel phenomena. By diverging from the scientific-instrumentalist approach, this thesis seeks to shed light on potential possibilities and implications associated with the exploration of novel virtual phenomena in the spirit of strong-AL, while acknowledging and accepting the theoretical limitations put forward by weak-AL.

According to this approach, the principal divide between the strong and weak schools of thought does not revolve around whether or not living systems can literally be realised in software. Instead, it focuses on the exploratory potential of computational modelling. This reframing also suggests a compromise: proponents of strong-AL would only need to concede that instances of virtual phenomena may not be fully realised as living entities, while proponents of weak-AL would only need to acknowledge that they are nonetheless worthy of exploration, not only as scientific models but as potentially valuable subject matter in and of itself.

The willingness to sever the well-established bond between virtual and physical phenomena does not negate the weak-AL position, nor does it necessarily argue for the strong-AL position. It simply states that the act of implementing emergent systems within virtual environments can serve both for the simulation of some objects and the realisation of others. By adopting this perspective, one can reclaim the notion of “life as it could be” — not as a scientific hypothesis but as an act of creative discovery; the acquisition of new phenomena to observe and study; a class of emergent systems that, in historical terms, has just recently become within reach through advancements in computing technology.

1.3. Emergent Virtual Structures

Existing terms such as “model” and “simulation” have historically been rooted in the scientific domain and inherently imply a representational function. Consequently, the use of such terms may inadvertently perpetuate a bias that values them primarily for their ability to imitate physical reality. Thus a study of explicitly non-representational computational models presents a conceptual challenge. This thesis therefore advocates for the use of alternative terminology, referring to a broad class of software based systems of interacting components as *emergent virtual structures*. This, in order to avoid characterising its subject matter as derivative objects of representation.

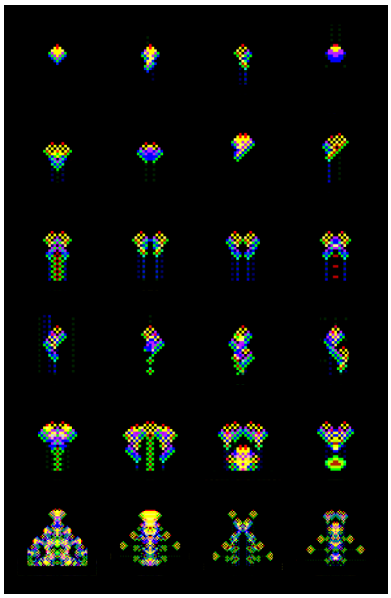


Figure 3.
Compound glider formations in Red Nose Hexagliders.

It is important to emphasise that this does not constitute an argument against representational modelling in general. Rather, it suggests that adopting this concept may be beneficial for open-ended exploration. By shifting the focus away from familiar constructs and functionality, this approach sets out to explore a deliberately vast and uncharted space of plausible forms and processes, considering these as something to be discovered rather than created. This approach suggests that exploration can be guided by observation and intuition, employing a methodology that shares some commonalities with exploration and cultivation of physical natural phenomena.

The concept of emergent virtual structures is not intended to be novel, nor does it constitute a contribution in itself. Rather, it serves as a useful framework for referring to the subject matter this project aims to explore. By employing well-accepted terms, this thesis aims to avoid contention and provide clarity. As such, “emergence” refers to the concept of weak emergence, as articulated by Bedau [15]. Weak emergence explicitly pertains to a class of nonlinear systems whose emergent properties can only be fully understood or predicted through simulation. By adopting a simulation-based concept of emergence, this thesis

appropriately narrows the scope of its subject matter, while avoiding any contentious issues associated with strong-AL.

In regards to “Virtuality”, this thesis embraces a philosophical view that considers the virtual realm as a vast, uncharted domain of potential phenomena, and is driven by an urge to explore it without bias or preconceived notions. Virtuality can be characterised by its opposition to the actual, rather than the real. It denotes immaterial yet genuinely existent constructs [16, p. 208]. Virtual entities can possess concrete attributes such as topology, persistence, and interaction, they exhibit the capacity to undergo changes and respond to external influences. Recognising that a study of virtual phenomena that disassociates it from physical reality may be perceived as non-scientific, this thesis willingly relinquishes this conventional scientific stance. Instead, it focuses on open-ended exploration as its primary research aim.

Lastly, Frigg’s concept of a “structure” [17] offers a useful framework for this thesis’s non-representational approach. Defined by its components and their relationships to one another, a structure does not inherently represent a real-world system unless explicitly supplemented with intent to do so. The use of the term structure, rather than model, emphasises its intrinsic, rather than instrumental value, aiming to detach the act of exploration from some of the constraints of traditional approaches to computational and scientific modelling.

The realm of possible emergent virtual structures is unfathomably large. By any reasonable means, it can be assumed to consist almost entirely of trivial, mundane, cacophonous, or otherwise seemingly ineffectual formations. Nonetheless, at the heart of this thesis lies a contention that this realm must also encompass countless forms and processes, the likes of which have never been observed in the physical world. The sheer magnitude of this search space and the unique attributes of this subject matter should provide more than enough motivation for it to be further explored.

However, this approach presents a number of formidable challenges: In lack of a predefined target system or a well-defined problem, there may be little to guide an explorer towards well-defined outcomes. Moreover, since findings are not necessarily meant to serve a particular purpose, there is no inherent way to evaluate their potential value, as traditional metrics based on predictive or explanatory power may not apply. An additional challenge lies in computational complexity, which is typically significant in large systems of interacting components. The inherently unpredictable nature of such systems necessitates that they be implemented and run, potentially for sustained periods of time, in order for observation to even take place, making classification, autonomous evaluation or taxonomical studies largely intractable.

1.4. Methodology

A central premise of this thesis posits that open-ended explorations of virtual phenomena can greatly benefit from the active involvement of artists, designers, hobbyists, and multidisciplinary practitioners. These individuals are regarded as the primary target audience: creative practitioners who may be motivated to explore the vast and overwhelmingly barren space of emergent virtual structures in search of *miraculous exceptions*.

This thesis therefore relies on qualitative evaluations of algorithms and their outputs, considering their aesthetic merit, novelty, complexity, rarity and capacity for human interaction. This aims to reduce reliance on established scientific concepts, which are often driven by an instrumentalist perspective, thus bridging the knowledge gap for artists and designers who may be

unfamiliar with advanced mathematics, computer science, and natural science. This approach aims to make this field of research more accessible to these audiences so that they may, in turn, introduce innovation, leading to further discovery of novel phenomena.

A scientific approach to exploration may offer appropriate theories and tools for studying emergent structures. However, scientific methodologies are typically concerned with describing observable physical phenomena. On the other hand, an artistic approach may offer conceptual tools more suited for open-ended exploration, yet these are typically under-equipped to handle the theoretical and technical rigour involved with the study of nonlinear systems.

This project seeks to bridge the methodological gap between computational modelling and artistic exploration in the context of the study of emergent virtual structures. It identifies an opportunity to explore novel forms and processes not just as instruments, descriptions or fabrications, but as cultural artefacts imbued with intrinsic value — a characterisation more commonly associated with works of art. The underlying contention is that such artefacts are not inherently rare. Rather, existing concepts, tools and methods have been primarily developed towards their utilisation as instruments and are thus not optimised for their exploration.

Research Aims

This thesis aims to propose and validate tools and methods specifically designed for open-ended exploration of CA, demonstrating their effectiveness through the discovery of novel algorithms. It introduces an innovative computational framework for navigating large combinatorial spaces of algorithmic variants, providing insights and methods for distinguishing findings of interest from their surrounding overwhelming noise. By adopting a non-instrumentalist approach to exploration, this work aims to uncover significant untapped potential for new algorithms and behaviours, showcasing how emergent forms and processes within virtual environments can be explored and evaluated independently of physical metaphor or function.

Research Objectives

1. Develop and demonstrate a new computational framework, called **Utomata**, specifically designed for exploration of CA algorithms.
2. Introduce and employ a high-level method for navigating large combinatorial spaces of algorithms, termed **Spatial Mapping**.
3. Demonstrate the effectiveness of these tools through exploratory studies of two new families of algorithms, called **Type-C** and **Type-U**.
4. Provide **qualitative evaluations** of novel findings.

1.5. Structure of This Thesis

Chapter 2 reviews foundational topics and key terminology relevant to this research. It explores the relationship between computational modelling and scientific inquiry, focusing on implications to the study nonlinear systems. It provides an introduction to the fields of AL and CA, reviewing notable milestones and algorithms. The chapter also reviews the work of practitioners who engage in exploratory programming and examines various programming environments suited for this purpose.

Chapter 3 introduces a new computational framework, Utomata, designed for the exploration and study of a wide range of CA algorithms. Utomata includes a custom programming syntax and a hardware-accelerated implementation, enabling live-coding of CA algorithms as nested algebraic statements. The chapter demonstrates Utomata's capabilities through examples of well-established algorithms, highlighting its ease of use for mutating algorithms and exploring their outputs interactively.

Chapter 4 showcases the use of Utomata for open-ended exploration through direct (low-level) algorithm manipulation. It discusses the application of functional programming, interactive parameter tweaking, and field parameterisation as methods for navigating the space of algorithmic variants. The chapter presents case studies of novel CA algorithms, offering initial qualitative characterisations of their behaviours.

Chapter 5 introduces Spatial Mapping, a novel high-level approach to CA exploration. Spatial Mapping involves constructing a combinatorial space of permutations of any given algorithm, and mapping them onto a large two-dimensional field. Accompanied by an online software implementation, this method facilitates exploration of vast landscapes of possible CA algorithms, potentially uncovering novel phenomena. The chapter presents Spatial Mappings of previously discussed algorithms, revealing numerous novel variations.

Chapter 6 introduces a new family of CA algorithms, termed Type-U. It provides a rigorous definition of Type-U algorithms and explores their variations through Spatial Mapping. The chapter offers initial classifications and qualitative accounts of Type-U's combinatorial spaces, as well as their characteristic phenotypic and genotypic properties.

Chapter 7 presents a collection of noteworthy Type-U specimens, accompanied by qualitative characterisations of their behaviours.

Chapter 8 discusses the main contributions of this thesis and their potential implications for open-ended exploration of CA, as well as other computational models and algorithmic domains. It also suggests possible future studies that could build upon or extend this work.

2. Background

This chapter offers an introduction to computational modelling and Cellular Automata (CA) in the context of artificial life (AL). Its principal aim is to establish the necessary conceptual and technical foundations for the tools, methods and explorations detailed in the following chapters.

Section 2.1 establishes computational models as software implementations that can simulate the behaviour of nonlinear systems — systems featuring a large number of interacting components. The concept of emergence is introduced and the distinction between simulation and realisation is discussed in the context of the strong-weak debate in artificial life. A further distinction is made in regards to the attribution of value with which computational models are imbued. This discussion draws the focus away from whether or not living systems can literally be realised in software, instead arguing for the merit of open ended exploration of emergent virtual phenomena. It refers to a broad class of non representational computational models as emergent virtual structures, setting the stage for the non-instrumentalist approach to exploration advocated by this thesis.

Section 2.2 provides an introduction to CA, which includes historical context, key terminology, as well as influential algorithms and applications. This section highlights the use of CA as an abstract model of nonlinear dynamical systems, making the case for exploration of CA beyond traditional, instrumentalist uses. Section 2.3 focuses on explorations of CA and other computational models, highlighting notable practitioners in both the sciences and the arts. This section serves to illustrate the interdisciplinary nature of exploration of emergent virtual phenomena. Additionally, it offers an up to date review of software tools and frameworks which can be used for open ended exploration, thereby bridging the gap between theory and practice.

Through this structure, the chapter aims to contextualise CA research within the broader context of open-ended exploration of virtual phenomena, which is closely related to the field of AL. Subsequent chapters build upon this foundation by introducing novel methods for exploration of CA, as well as demonstrating their effectiveness through the introduction and preliminary study of novel CA algorithms.

2.1. Computational Modelling

2.1.1 Nonlinear Problems

Scientists of the Enlightenment saw the natural world as a great machine that operates according to the laws of Newtonian mechanics. This approach was driven both metaphorically and in-effect by the Industrial Revolution. However, by the early 20th century, this mechanistic worldview was becoming difficult to reconcile with new research involving systems featuring a large number of interacting components [18], [19]. The problem was that, unlike a machine, such systems tend to lose their defining characteristics when taken apart for examination and analysis. For example, when studying the behaviour of an ant colony, separating the ants from each other and studying them in isolation effectively destroys the primary subject of research — the colony, along with most of its defining characteristics.

A fundamental property of such systems is called *nonlinearity*. A system is said to be nonlinear if its behaviour does not follow the superposition principle, [1] which states that the combined effect of two or more independent systems can be represented by the sum of their individual effects. For instance, when considering a parallel arrangement of springs or struts, the total forces exerted can be computed by summing the forces acting on each element separately. However, in nonlinear systems, the output does not exhibit a proportional relationship with the input, and the response of each component is not independent. Nonlinear systems often exhibit intricate interactions and interdependencies between their components, leading to phenomena such as feedback loops or amplification that cannot be explained by superimposing the behaviour of individual parts.

Nonlinear systems can be precisely described by analytical methods such as partial differential equations, providing a well-defined framework for understanding their dynamics. However, in practical terms, these equations are often unsolvable due to the presence of numerous open variables representing the components of the system. Thus, while an analytical approach can effectively capture the general properties of nonlinear systems, it falls short in accurately predicting their behaviour. [2] Since the ability to make accurate predictions is a fundamental objective of scientific research, the traditional mechanistic, analytical worldview of the 19th century proved inadequate when studying a wide range of observable phenomena. By the 1940s, it became increasingly evident that nonlinearity is not the exception but the rule, as famously noted by Stanislav Ulam, “Using a term like nonlinear science is like referring to the bulk of zoology as the study of non-elephant animals”. [21]

To obtain an accurate prediction of a nonlinear system, one must employ a numerical method that involves iteratively computing the operations of each component in the system — step by step. Performing such computations manually is extremely impractical, even for systems with a small number of components. Therefore, some analytical models of nonlinear systems resort to approximating them as linear systems. For example, Hooke’s law, which describes a linear relationship between the force exerted by a spring and its displacement from equilibrium, is only accurate for small displacements. By simplifying the nonlinear relationships into linear approximations, these models attempt to capture the essential dynamics of the system. However, such approximations inherently introduce inaccuracies, which in turn greatly impede the quality of the predicted behaviour.

2.1.2 Synthetic Solutions

Computers possess a distinct advantage over humans when it comes to solving nonlinear problems: their ability to rapidly carry out large sequences of numerical operations step by step. While computational modelling has long been applied to linear systems — such as in missile trajectories, graphics pipelines, and sorting algorithms — its capacity to iteratively calculate the behaviour of each component in a nonlinear system makes it uniquely suited for addressing the complexities of nonlinear dynamics. This approach results in a *synthetic solution* that can be highly effective in predicting the behaviour of nonlinear dynamical systems across various scales and disciplines. It is crucial to note that this does not constitute an analytical solution, nor does it offer analytical insight in the traditional sense.

Nevertheless, this computational approach has been employed by many of the pioneers of computer science towards the study of nonlinear phenomena across a wide range of disciplines. These included mathematical physics [22], self-replication [5], biochemistry [23], intelligence [24], neuroscience [25], evolution

1 *According to the superposition principle [20], if the following equality holds true the system is said to be linear:*
$$f(a\,x + b\,y) = a\,f(x) + b\,f(y)$$

2 *Though it should be noted that physics has achieved great success in studying nonlinear systems by focusing on cases where the nonlinear component is a small but important correction to the underlying linearity. Techniques such as perturbation theory allow for continual refinement of these approximations, improving the accuracy of predictions.*

[26] and homeostasis [27]. Thus, it can be argued that computational modelling emerged as a natural extension of the scientific study of nonlinear systems. The evolution of computing technology has exhibited a close correlation with the growing understanding of nonlinearity, with discoveries in one field often rapidly influencing developments in the other [28].

The complexities inherent in nonlinear systems, characterised by their intricate interdependencies and unpredictable behaviour, necessitate research practices that transcend traditional analytical methods. Computational modelling offers a way to leverage the power of digital computing to iteratively simulate the behaviour of large systems of interacting components. While this method is not perfect, as it does not offer analytical insight and remains highly dependent on initial conditions, it provides a practical approach to understanding these systems. It offers more precise predictions and deeper insights into phenomena that would otherwise be unmanageable through manual calculations or linear approximations. By enacting the dynamics of such systems, computational models serve as vital tools in bridging the gap between theoretical understanding and practical application, offering scientists and researchers unprecedented means to study and comprehend nonlinear systems across a wide range of disciplines.

2.1.3 Emergence

Emergence is intricately linked to nonlinear systems and is fundamental in understanding how complex behaviours arise from simpler interactions. In nonlinear systems, the global state of the system is often driven by local, lower-level interactions between its constituent parts. This bottom-up progression, where lower-level interactions result in higher-level behaviour, sets emergent systems apart from broader concepts like complexity or nonlinearity.

Early references to emergence, traced to 19th century philosophy, describe systems as being more than the sum of their parts [29]. These principles evolved with early self-organisation theories [18], which consider the interplay between the whole and its parts. Emergent systems exhibit properties or behaviours not evident in individual components, making global patterns difficult to predict by observing local interactions [30]. Darley argues that even with perfect knowledge about an emergent system, predicting its behaviour can be challenging, pointing to simulation as the best way to predict its behaviour [31].

Within the realm of emergent systems, Bedau's concept of *weak emergence* holds notable relevance [15]. According to Bedau, a weakly emergent system is one whose global properties can be deduced from its underlying local dynamics, albeit exclusively through simulation. In contrast, real and phsyical emergent systems possess irreducible global properties and therefore are said to feature *strong emergence*. Subsequent definitions include a grammar based set-theoretic approach to formalising emergence [32], a mathematical characterisation of strong emergence [33] and a correlation between emergence and natural selection [34]. Thoren and Gerlee characterise artificial life as "a search for the surprising" [35] and note that emergence is commonly used as a label for the formation of "surprising" higher order structures.

2.1.4 Simulation vs. Realisation

> "There is nothing in its charter that restricts biology to the study of carbon-based life; it is simply that this is the only kind of life that has been available to study."
> [12, p. 2]

The first workshop on Artificial Life (AL) was held in Los Alamos, New Mexico in 1987. It was initiated by Christopher G. Langton and had brought together an interdisciplinary group of researchers who shared a common interest in the "simulation and synthesis of living systems". The workshop's proceedings [12, p.2] commence with a comprehensive essay by Langton, simply titled *Artificial Life,* introducing a new field of research that constitutes a synthetic approach to biology, by which living systems would be assembled, rather than taken apart. [3] According to Langton's vision, AL would be dedicated to the study of *life-as-it-could-be,* complementing biology's study of *life-as-we-know-it* by synthesising novel instances of living systems *in silico* — in the form of computer programs. [4]

The notion of artificial living systems was not an entirely new idea at the time. Langton cites the much earlier work of John Von Neumann [12, p. 13] and Alan Turing [12, p. 12] as the foundations of this newly established discipline. Moreover, computational modelling has, by then, been used extensively as a scientific instrument for simulating biological systems. Even the attribution of "life-ness" had been previously ascribed to computer programs [36]. Nonetheless, this vision had sparked overwhelming interest in the realisation of life-like processes in silico.

By the early 1990s, AL had become an active field of research, attracting scientists, designers, artists and engineers from across disciplines towards dedicated international conferences and publications. Although the work of Langton and his colleagues was tightly coupled with physics, chemistry, biology, computer science and mathematics, early AL research was characterised by a unique approach, whereby computational modelling was no longer regarded as an instrument for describing and predicting observed physical phenomena — but as a medium for realisation of novel virtual phenomena.

> "There is nothing wrong with a good illusion, as long as it does not claim it is reality."
> [14, p. 74]

The above distinction between simulation and realisation was put forward by Pattee at the very first AL workshop in 1987, effectively foreshadowing one of the core debates in the field. Pattee argued that a simulation is a representation of an existing system, whereas a realisation is a "literal, substantive, functional device" [14]. Not a derivative object or representation, but an object by its own regard. From a scientific point of view, the purpose of a computational model is to approximate the behaviour of a particular observable phenomenon. This, in an effort to obtain new knowledge about a system in question in situations where an analytical model is either impossible or impractical. Thus the assertion that living systems can be directly implemented in software posed a substantial conceptual challenge and has faced significant controversy.

Biology lacks a unified theory of living systems, with even the classification of edge cases such as viruses [37] and super-organisms [38] remaining in dispute. Moreover, widely accepted definitions of life often include physical properties, such as maintaining metabolic processes or being composed of living cells [39]. As a result, research projects seeking to create virtual living entities tend to focus on those rooted in consensus, rather than exploring the boundaries and edge cases of biology. The OpenWorm project [40], which strives to simulate a complete multicellular organism, is a notable example of this approach. While such projects offer valuable contributions to science, they seem to fall short of being considered by the scientific community as creating actual living systems. According to this perspective, computational models, no matter how precise, are necessarily derivative objects and therefor never the real thing.

3 *The term Artificial Life may also pertain to work involving wetware and hardware implementations. However, since this thesis is strictly concerned with software, unless otherwise stated all mentions of AL refer to software implementations as such.*

4 *The term "in silico" was originally coined by Langton [12] and pertains to biological experiments conducted exclusively through computational means. It contrasts with "in-vivo", "in vitro", and "in situ" approaches.*

Langton's own vision of a synthetic approach to biology was seen by many in the scientific community as reckless miss-use or even misunderstanding of what a computational model actually is [13]. Critics of AL essentially viewed it as the act of taking a tool for describing physical phenomena and deeming it capable of creating it. Bullock discusses this conflict in the context of artificial worlds. He refers to artificial life as a practice which deliberately courts insecurity by focusing on the notion of life as it could be [41]. In a 1995 article, John Horgan highlights Langton's perspective:

> "Langton, surprisingly, seems to accept the possibility that artificial life might not achieve the rigour of more old-fashioned research. Science, he suggests, may become less linear and more poetic in the future. 'Poetry is a very nonlinear use of language, where the meaning is more than just the sum of the parts,' Langton explains. 'I just have the feeling that culturally there's going to be more of something like poetry in the future of science.'"
> [13]

Horgan refers to AL as a "fact free science", a critique that is not unreasonable considering Langton's stated goals. If scientific inquiry is grounded upon observation of physical subject matter, it should follow that the scientific legitimacy of research that deliberately avoids this would come into question. Langton's body of published work is surprisingly sparse considering his attributed stature as the father of AL [13]. He typically makes no direct attempts to imitate, model or represent any particular life form and his language involves cautious use of biological metaphor. Instead, he studies the general properties of processes associated with living systems such as self-organisation and self-replication [42], as well as more abstract properties such as phase transitions between order and chaos [43].

2.1.5 Strong vs. Weak

The question of whether or not life can indeed be realised in software took form as the field split into two prominent schools of thought, otherwise known as the strong-weak debate in AL. It embodies contrasting interpretations of the term "artificial", as well as inherent challenges in defining "life". While proponents of both schools of thought employ computational modelling techniques to synthesise emergent phenomena, it is the attribution of life-like qualities to these synthetic objects that necessitates a departure from the traditional scientific framework, whereby a model cannot be regarded as creating the very thing it aims to model.

The weak position accepts the role of computational modelling as a scientific instrument, which means it must categorically deny any instance of virtual phenomena from being considered truly alive, on the basis that it is merely a model or simulation. On the other hand, the strong position contends that virtual phenomena may exhibit life-like behaviours. Therefore, it follows that a computational model can indeed become more than a scientific instrument. This dichotomy can be exemplified through the approaches of Langton, the founding editor of MIT's Artificial Life Journal, and Mark A. Bedau, a prominent critic of strong-AL, who assumed Langton's editorial position in 1998.

As a proponent of strong-AL, Langton considers artificial entities as "human-made" creations. This view does not impose a clear separation between natural and artificial, therefore permitting the possibility of living systems to be created by artificial means. He further asserts that life is a property of form — not matter — and can therefore exist within artificial, virtual environments. At least in

theory. Langton's published body of work typically downplays any discussion on practical applications; he posits open questions about his subject matter and hypothesises over potential future applications, but the underlying motivation seems grounded upon the contention that virtual living systems are indeed possible and that this alone is reason enough to study and explore them.

Bedau, a vocal proponent of weak-AL, considers an artificial construct to mean a derivative object. He asserts that such objects cannot capture the complexity and essential qualities of natural life forms, or rather that life emerges from specific arrangements of matter and processes that cannot be reproduced through artificial means. Bedau argues that while computational models may offer valuable insights by simulating various aspects of living systems, they ultimately fall short in regards to creating the phenomena itself. Bedau even goes as far as to consider physicality as one of the innate properties of living systems [44], suggesting that the notion of virtual life may not be feasible to begin with.

Bedau is motivated by the urge to understand what life is [45]. He contends that software is and would continue to be an invaluable tool in obtaining an increasingly better understanding of living systems through simulation, as well as in offering novel solutions to some of humanity's current and future problems [46]. In other words, He explicitly ascribes instrumental value to software implementations of AL, considering them as tools in the service of scientific inquiry; he formulates questions and employs various modelling techniques to derive answers. Whereas for Langton, software implementations of AL are themselves the answer — they extend the natural world by the sheer virtue of their existence.

While both Langton and Bedau argue for the potential of a synthetic approach to biology in uncovering truths about living systems, their goals and interests diverge. Bedau's listing of open problems in artificial life [46] demonstrate a quest to define life, explore its limits, understand its innate properties, and examine its relationship with machines, culture and the human mind. Bedau's philosophical and methodological approach, as exemplified in the desire to simulate a complete life cycle of a unicellular organism [46, p. 367], reveals a strong motivation to fully comprehend what life is, but ultimately demonstrates no interest in exploring what it *could be*.

The paradigm shift from strong to weak AL as the dominant school of thought in the field towards the end of the 20th century emerged as a response to criticisms regarding its scientific legitimacy. It marked a return to the traditional view of computational modelling as a tool for describing phenomena rather than creating it. Subsequently, the exploratory and open-ended nature of early research in AL gradually gave way to more focused efforts for understanding the inherent properties of observable living systems.

2.1.6 Model vs. Structure

An object is said to possess instrumental, or extrinsic value if its purpose lies in serving a function. For example, a chair may be imbued with instrumental value because it was created for the purpose of sitting on. In contrast, an object may be said to possess intrinsic value if its purpose lies in its own existence. For example, a mountain can be regarded as serving no particular function, or alternatively, that its value is self-evident. The attribution of value is often a moral position and a highly subjective one; a carnivore might argue that a cow possesses instrumental value because it was brought into existence for the purpose of human consumption, whereas a vegan would likely argue that cows, as do all living creatures, possess intrinsic value.

It should be emphasised that the distinction between intrinsic and instrumental value is rarely clear cut. While the typical function of many chairs may be obvious, some chairs may have been created with other functions in mind, or none at all. Moreover, the decorative or aesthetic features of a chair, which, by definition, are not functional, or may even hinder its functionality, do not automatically imbue it with intrinsic value.

Computer programs are frequently designed as instruments, with word processors, web browsers, code editors, and digital audio workstations serving as clear examples. These applications are typically valued based on their effectiveness in performing specific functions within their respective domains. In contrast, computational art, experimental software, generative compositions, theoretical constructs, and, to some extent, computer games, are less commonly regarded as instruments.

As for computational models, their extensive use as scientific instruments for the study and prediction of observable phenomena has been so impactful that it often overshadows alternative perspectives. The term “computational model” itself suggests a representation or derivation of something pre-existing, rather than an independent subject of research. This is true even in the field of AL which, in many ways, was founded with the explicit intention of studying un-observable phenomena. Grüne-Yanoff encapsulates this idea by stating, “Models are representations; they are good models to the extent that they are good representations” [47].

This perspective highlights an often overlooked aspect of computational modelling: it is inherently goal-directed, designed to imitate rather than enact. This predisposition toward representation necessarily constrains how models are conceived and valued, potentially making them ill-suited for studies concerned with open-ended exploration. It thus follows that suspending this instrumentalist view — ceasing to evaluate computational models based on their ability to achieve predefined goals or represent external phenomena — could open new avenues of inquiry. Such a shift may allow the discovery of novel, non-representational systems that are neither derived from nor tied to pre-existing phenomena.

Given that the primary aim of this thesis is to conduct open-ended explorations of emergent virtual phenomena, the term ‘model’ itself becomes problematic when restricted by its traditional representational connotations. What, then, constitutes a non-representational model?

Frigg [17] defines a “structure” as a composite entity comprising the following components:

1. A non-empty set U of individuals called the domain of the structure
2. A set O of operations on U (which may be empty)
3. A non- empty set R of relations on U

According to Frigg, structures, in and of themselves, do not inherently represent elements of the real world. To serve a representational role, structures must be supplemented with a specification of their relationship to the target system. Only with this relational aspect do they acquire representational capability and become “about” something. Furthermore, establishing this relationship does not automatically render a structure an isomorphic model of a target system. Frigg emphasises the significance of intentionality in this process: a structure can only be considered a model when it is explicitly intended for

representational use. This intended use is not merely an external facet appended to the model but forms an essential part of the model itself.

The distinction between a “model” and a “structure” provides a highly suitable framework for the subject matter of this thesis. By employing a subtractive approach, any computational model can be dissociated from its connection to a target system and any intent to represent it. What remains is a structure — a well-defined, autonomous entity that transcends derivative and instrumental categorisation. While this distinction may seem semantic, it reflects a pivotal conceptual shift: exploring structures, rather than models, can significantly broaden the scope of inquiry. As the following chapters demonstrate, this approach can indeed enable the discovery of novel virtual phenomena that may otherwise be overlooked within the narrower confines of a model-based perspective.

2.2. Cellular Automata

The following section provides an introductory review of Cellular Automata (CA), which form the central focus of this thesis. It aims to achieve two primary objectives: first, to introduce key terminology and discuss the historical significance of CA as one of the pioneering computational models, and second, to review three notable algorithms that illustrate the practical application and effectiveness of the proposed computational frameworks introduced in Chapters 4 and 5.

CA are regarded as one of the first computational models, originating as early as the 1940s. Their development is attributed to Stanislav Ulam and John von Neumann, who pioneered early research on crystal formation [4] and self-replication [5], respectively. Von Neumann’s groundbreaking work introduced the Universal Constructor — an intricate theoretical machine capable of self-replication. This research, preceding the development of digital computers and thus devised largely on graph paper, was originally envisioned as the basis for physical self-replicating machines. However, Von Neumann later settled on a purely virtual implementation [48, p. 879]. Since then, the topic of autonomous self-replication has inspired numerous researchers [49], leading to simpler and more elegant solutions using CA and related computational models [42], [50].

At their core, CA consist of a regular grid of discrete entities, or cells, each associated with a numerical value or symbol — its *state* — that evolves over time according to a global set of rules called the *transition function.* The transition determines how each cell’s state changes based on the states of its neighbouring cells, collectively referred to as its *neighbourhood*. The evolution of the grid typically occurs synchronously at discrete time steps, with the same transition applied uniformly across all cells. Despite their often simple rules, CA are capable of generating remarkable complexity. Local variations in cell states can produce intricate patterns and dynamic behaviours, ranging from simple, localised periodic structures to large-scale emergent patterns.

CA have a long and rich history, closely intertwined with the development of digital technology in the 20th century. The following section covers the fundamentals of CA within the narrow scope of this thesis, which focuses on exploration and emergent behaviour. For readers seeking a broader introduction to CA, Wolfram [48] provides a comprehensive exploration of their potential to model complexity across diverse domains. Toffoli and Margolus [51] offer a practical perspective on CA implementation, effectively bridging theoretical concepts with computational applications. For a more rigorous and extensive treatment, Ilachinski [52] covers both foundational theory and real-world applications.

2.2.1 Key Terminology

State

In the context of CA, the state of a cell represents its current condition and can take on a variety of values. Two well-known CA models, which are discussed in the next section, are Conway's Game of Life [53], [54] and Wolfram's Elementary Automata [55]. Both models use binary states, where each cell's state may only be either 0 or 1, sometimes regarded as "off" or "on", or rather "dead" or "alive". However, many CA models go beyond binary states to allow multiple discrete values. For example, the Abelian Sandpile [11] and Wireworld [56] each use four discrete states, while von Neumann's Universal Constructor used twenty-nine.

In addition to discrete states, some CA models use continuous values to better simulate physical and chemical processes. For instance, reaction-diffusion [57], also discussed in the next section, represent each cell's state using two real numbers between 0 and 1, corresponding to the concentrations of two interacting chemical substances.

Neighbourhood

The concept of the neighbourhood is central to CA, as it defines how cells interact with their surroundings, forming a critical component in emergence of complex patterns. Different CA models adopt varying approaches to neighbourhood definitions, which influence how cell states throughout the system evolve over time.

In a *totalistic* CA, the transition function depends solely on the sum of the states of neighbouring cells, without considering their individual values. Conway's Game of Life is a well-known example of this approach, which provides a simplified yet powerful model capable of producing intricate emergent patterns. Figure 4 visualises three commonly used totalistic CA neighbourhoods: Von-Neumann, Moore and Extended Moore. In contrast, *outer-totalistic* CA employ a more fine-grained approach by considering the individual states of neighbouring cells. This allows for a broader range of possible transitions and nuanced control over local interactions. Models such as Elementary Automata and the Abelian Sandpile utilise this approach, forming behaviours that may not emerge in totalistic models.

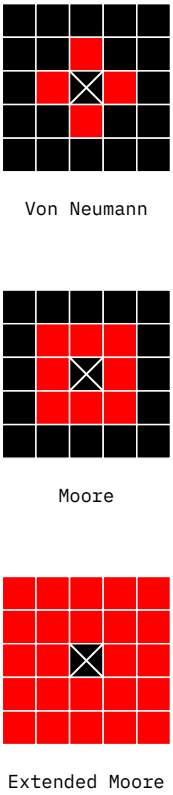


Figure 4.
Commonly used totalistic neighbourhoods.

Transition

The transition function is the central mechanism that drives the evolution of CA over time. It determines how each cell's state changes based on its current state and the states of its neighbours. While the same transition is typically applied to all cells in the grid, results vary due to varying local state and neighbourhood values.

Transition functions can be expressed in various ways, depending on the complexity and purpose of the model. In simpler cases, such as Conway's Game of Life, the transition function is typically implemented using straightforward conditional statements — checking the number of "live" neighbouring cells and applying simple if-else rules to determine the next state. In contrast, Wolfram's Elementary Automata utilise rule tables, which resemble truth tables, where each possible neighbourhood configuration corresponds to a predefined output state for the transitioning cell. More advanced CA models, such as reaction-diffusion systems, define transition functions using algebraic equations that describe continuous changes over time, often drawing from differential equations to simulate physical and chemical processes.

Configuration

The initial state of all cells in a CA is known as the *configuration*, and it plays a key role in shaping the system's evolution. Since the transition function determines how each cell updates based on its current state and the states of its neighbours, the configuration serves as the foundation from which all subsequent patterns and behaviours emerge.

Configurations can be categorised into different types depending on their structure and origin. Predefined configurations, such as a specific pattern or arrangement, are often used to study well-known behaviours, such as gliders or oscillators in Conway's Game of Life. In contrast, random configurations, where cell states are initialised randomly, are commonly used to explore statistical properties and emergent phenomena within the system. Some models may also use structured configurations, derived from real-world data or external inputs, to simulate physical or biological processes.

Determinism

While many known transition functions in CA are often relatively simple, they have a remarkable ability to produce highly intricate and emergent patterns [48] [53]. In many cases, this complexity does not typically arise from randomness but rather from the inherent feedback mechanisms between neighbouring cells. In *deterministic* CA, a given initial configuration, transition function, and number of steps will always yield the same outcome, ensuring consistency across multiple independent runs.

Some CA models do introduce elements of randomness by incorporating random or pseudorandom numbers into their transition functions. These models, known as *stochastic* CA, are often used to simulate real-world phenomena that involve varying levels of uncertainty, such as in epidemics [58] and ecological systems [59]. Notably, deterministic CA can still exhibit *chaotic* behaviour, where any change to the configuration, or slight perturbation, can result in vastly different outcomes over time. This sensitivity to initial conditions is closely tied to CA's potential to develop complex, unpredictable patterns from simple rules.

Dimension

CA grids may span any number of dimensions [5]. In one-dimensional (1D) CA, cells are arranged in a linear sequence. A classic example is Wolfram's Elementary Automata, where neighbourhoods consist of only two adjacent cells. More common are two-dimensional (2D) systems, such as Conway's Game of Life. 2D CA feature cells arranged orthogonally along an the x and y axes, with neighbourhoods typically consisting of the four or eight adjacent cells.

Less commonly studied, three-dimensional (3D) CA extend this concept into volumetric space, enabling the formation of intricate 3D structures. In a 3D lattice, each cell has twenty-six adjacent neighbours. 3D CA have been used in a variety of applications, including physical modelling [60], simulation [61] and urban planning [62]. Higher-dimensional CA (4D and beyond) have also been explored to study Complex systems [63] and theoretical constructs [64]. [6]

Bounds

CA systems operate within finite grids, as all cell states must be computed within a well defined space and time. Handling grid boundaries is crucial, as they potentially introduce changes to behaviour along the edges, which can rapidly percolate into the entire system. In 2D systems, two common approaches are widely used:

5 *This should not be confused with state dimensionality, which refers to representing the state of a cell using a vector of values rather than a single numerical value.*

6 *See [65] for an extended discussion on dimensionality and alternative CA topologies.*

Clamped boundaries impose strict confinement, preventing interactions beyond the grid's edges. Cells at the boundary assume a predefined state, typically zero. *Toroidal boundaries* create a seamless, wrap-around effect, where the neighbours of cells along one side are the cells along the opposite side. Topologically, this creates a torus shaped space, effectively simulating an unbounded space, thereby preventing foreign effects along the edges. Unless otherwise stated, this approach is used throughout the studies presented in this thesis. It is particularly useful for studying systems with periodic or cyclic behaviour over extended periods of time.

Computational Complexity

CA are considered highly computationally efficient due to their reliance on local interactions, where each cell remains stationary and interacts only with a fixed set of neighbours. This ensures that the cost of updating a single cell remains constant, leading to an overall computational complexity that scales linearly — $O(n)$ — with the number of cells in the grid. In contrast, many particle systems involve moving entities that require pairwise interaction calculations, where each particle interacts with every other particle at each step. Without optimisation, this results in a significantly higher computational complexity of $O(n^2)$, making such systems computationally expensive as they scale.

The efficiency of CA was particularly valuable during the early days of computing when processing power was limited, and it remains advantageous today, especially in large-scale simulations where local interactions are sufficient to capture the system's dynamics.

The next section introduces three notable CA algorithms that have gained considerable popularity and continued research interest. Their widespread adoption extends beyond scientific domains, towards the realms of arts and design as well. Moreover, these algorithms vary in a number of aspects, such as their state type, dimension, and their algorithmic expression. This makes them ideal case studies for illustrating the use and effectiveness of the computational framework outlined in Chapter 4, as well as ideal starting points for the exploration methods presented in Chapter 5.

2.2.2 Game of Life

The popularisation of CA can be traced back to the publication of Conway's Game of Life (GOL) in 1970. In a Scientific American article [53], Martin Gardner described a "zero-player game" created for recreational purposes by mathematician John Conway. Despite its name, GOL does not simulate any particular living system, nor was it designed to. Nonetheless, its metaphor and incredibly simple rule system attracted immediate interest in CA from outside the academic ranks for the first time. Gardner detailed the rules of GOL as follows: [7]

- Survivals: Every counter with two or three neighbouring counters survives for the next generation.
- Deaths: Each counter with four or more neighbours dies (is removed) from overpopulation. Every counter with one neighbour or none dies from isolation.
- Births: Each empty cell adjacent to exactly three neighbours--no more, no fewer--is a birth cell. A counter is placed on it at the next move.

7 *Gardner refers to cell state as counters and the game uses the totalistic Moore neighborhood.*

The deceptively simple rules of this algorithm [8] result in a particular dynamic between cell states, from which instances of stable higher-order patterns emerge. These are formed by distinct local arrangements of neighbouring cells that preserve their structure over time. They may take different forms, most notable of which are *oscillators* — formations that cycle through a particular arrangement in place, and *gliders* — formations that traverse the grid diagonally.

GOL has enjoyed an abundance of further explorations of its possible behaviours, mathematical and statistical properties, as well as possible outcomes from different configurations. These include a demonstration of its universality by Randell [66] and Stephen Silver's compiled Lexicon, which features over 1,350 terms relating to GOL [67]. Johnston and Greene recently provided an in-depth investigation of GOL rules, variations, configurations and underlying maths [68]. Smoothlife [69] is an extension of GOL into a continuous domain. Eppstein's custom abbreviated notation, "Life-Like" [70], allow parametric explorations of GOL variants [71]. More recently, a numerical analysis by Peña and Sayama [72] quantified the complexity of Life-like cellular automata, providing a systematic evaluation of their dynamic behaviors and identifying patterns that exhibit life-like characteristics beyond Conway's original formulation.

2.2.3 Elementary Cellular Automata

In his book: A New Kind of Science [48] Stephen Wolfram provides an in-depth discussion on how complex patterns may emerge from the simplest possible rules and configurations. His method differs from traditional scientific experimentation in that, rather than observing a system and making a hypothesis regarding its behaviour, Wolfram defines a space consisting of all possible permutations of a given behaviour, in this case: one dimensional, binary state, outer-totalistic CA. These, so called Elementary Automata (ECA) have exactly 256 possible transition functions, making a rigorous examination of all of them a tractable task. According to Wolfram:

> "In a traditional scientific experiment, one sets up a system in nature and then watches to see how it behaves. And in much the same way, one can set up a program on a computer and then watch how it behaves. And the great advantage of such an experimental approach is that it does not require one to know in advance exactly what kinds of behaviour can occur. And this is what makes it possible to discover genuinely new phenomena that one did not expect."
> [48, p. 108]

Wolfram's research considers CA as a way to implement abstract structures in which behaviour can be explored without any need for an equivalent physical system to be observed, measured or compared against. Acknowledging the innumerable size of the space of all possible CA programs, his solution is to methodically comb the smallest well-defined subspace of these algorithms and then evaluate findings visually. Wolfram defined ECA transition functions using a rule table that enumerates all possible states for a given neighbourhood and subsequent transition state. In a 1D outer-totalistic CA, there are exactly 8 possible neighbourhoods. [9]

Since any given neighbourhood in a binray state CA results in the transition function returning either 0 or 1, a simple algorithm can be defined by explicitly specifying the output for all eight possible neighbourhood configurations. Consequently, there are exactly 256 unique algorithms, each corresponding to a different combination of outputs for these neighbourhoods. These algorithms

8 *The rules of GOL can be formally expressed using the following pseudocode (using V to denote the state of the cell and N to denote its Moore neighborhood total):*

```
For each cell :
if V == 1 :
    if N == 2 OR N == 3 : V = 1
    if N >= 4 OR N <= 1 : V = 0
if V == 0 :
    if N == 3 : V = 1
```

9 *The eight Possible neighborhoods, including the transitioning cell, in a binary state 1D CA are :*

```
000 | 001 | 010 | 011
100 | 101 | 110 | 111
```

are all accounted for by a rule table that enumerates the results of the transition function. For instance, Rule 30 [10], represented here in binary notation, denotes the output of the transition function for each of the eight neighbourhood configurations.

***10** Rule 30 in binary notation*

0	0	0	1	1	1	0

Rule tables constitute a different form of algorithmic expression compared to the more common if-then rules used for defining GOL. Rule tables provide an explicit result for every possible scenario. ECA can be also defined using logical notation [73]. This form is covered in depth in Chapter 4.

In 1984 Wolfram and Packard [74] proposed four distinct classes of CA behvaiour, offering examples in ECA for each one. This classification method, listed below, is still widely referred to in literature to date.

1. Evolution leads to a homogeneous state.
2. Evolution leads to a set of separated simple stable or periodic structures.
3. Evolution leads to a chaotic pattern.
4. Evolution leads to complex localised structures, sometimes long-lived.

It is important to emphasise that these classes do not ascribe well-defined properties to CA, nor provide analytical insights per se. Descriptive terms such as “simple”, “stable”, “chaotic pattern”, and “complex localised structures” are not formally defined. Wolfram refers to this classification as a “qualitative characterisation of elementary automata.” He later extends his investigations to include empirical, statistical, and combinatorial analyses, as well as studies of CA behaviour in higher dimensions [75]. However, these investigations focus exclusively on binary-state CA.

Subsequent efforts have been made to classify CA, including notable contributions by Martinez et al. [76], Sutner [77], [78], Adamatzky [79], and Braga et al. [80]. However, despite these efforts, a universally accepted, general-purpose classification framework for CA algorithms and the diverse phenomena they exhibit has yet to be established.

2.2.4 Reaction-Diffusion

Reaction-diffusion (RD) systems are a fundamental concept in the study of pattern formation and self-organisation. These systems describe the interplay between chemical reactions and the diffusion of substances within a medium, leading to the emergence of intricate spatial structures. RD models have broad significance across multiple scientific disciplines, including chemistry, biology, physics, and computer science.

The origins of RD modelling can be traced back to Alan Turing’s seminal 1952 paper, The Chemical Basis of Morphogenesis [23], in which he demonstrated how simple chemical interactions could give rise to complex patterns observed in nature. Turing’s mathematical framework provided insights into various natural phenomena, such as embryonic development, animal coat patterns, and plant morphology. This foundational work established a theoretical basis for morphogenesis — the process by which biological structures develop — paving the way for advancements in computational biology and the study of self-organising systems.

Building on Turing's foundational work, the Gray-Scott model [81], provides a canonical implementation of RD systems. This model simplifies Turing's original equations, making them more computationally tractable. The Gray-Scott model involves two chemical substances, U and V, that react and diffuse on a two-dimensional grid, leading to the emergence of various organic-like patterns. The equations driving the model describe the rates of change of U and V based on reaction kinetics and diffusion.

John E. Pearson further popularised this model through his 1993 study [10], which demonstrated the rich diversity of patterns it could generate. Pearson explored the simulation and study of "chemically reacting and diffusing systems" in silico, showcasing a remarkable diversity of patterns through the use of phase diagrams that depicted varying concentrations of two simulated chemical substances along the x and y axes. This work is referred to as "Pearson's parameterisation" and was later implemented as interactive software by Munafo [82] and Sims [83]. This method is central to the practice-based research presented in this thesis. It represents an early example of a study involving exploration of continuous state CA and serves as the foundation for one of the proposed exploration methods detailed in Chapter 5.

RD systems continue to be studied and applied in diverse scientific domains. As effective models of chemical dynamics, they contribute to the development of new materials and catalysts. In biology, RD mechanisms are crucial in elucidating the dynamics of cellular processes, such as cell signalling, morphogenesis, and tissue patterning. They provide insights into the formation of biological structures and contribute to fields like developmental biology and regenerative medicine [84]. Moreover, RD has been highly influential in computer graphics, texture synthesis, and generative design. An implementation by Witkin and Kass [85] served as a significant milestone in the field of artificial chemistry [86], [50]. Subsequently, it laid the foundation for other algorithms that generate novel virtual phenomena such as Lenia [87] and Smooth-Life [88].

RD models are often regarded as a distinct computational paradigm from CA due to their use of continuous states, which allow for more precise modelling of organic and diffusive processes. In contrast, many CA models are explicitly designed to represent interactions between discrete entities. However, despite these differences, both models share fundamental characteristics, as they operate within discrete space and time. Given that the primary focus of this thesis is exploration of novel CA algorithms, it is logical to adopt a broad definition of CA that attempts to encompass a broad range of algorithms, including RD.

2.2.5 Other Notable Algorithms

The three algorithms discussed in the previous section are among the most extensively studied and widely recognised CA models, commanding significant research and academic attention. Their prominence is particularly evident in artistic contexts, where the most well-known algorithms often gain widespread recognition and visibility. However, these algorithms represent only a fraction of the broader landscape of CA. Numerous other algorithms with diverse characteristics and applications exist, as highlighted in a survey by Bhattacharjee et al. [89].

The Abelian Sandpile model, introduced by Bak, Tang, and Wiesenfeld in 1987 [11] is of particular significance to this thesis. Embodying the concept of a self-organised criticality (SOC), the model consists of a finite grid where each cell can accumulate a certain number of "sand grains". When a cell exceeds its maximal

capacity (typically four grains), it topples, distributing its grains to its four adjacent neighbours. What makes this model model particularly intriguing is its inherent ability to reach a critical state without the need for fine-tuning or external parameters. As sand grains are continuously added and redistributed, large-scale avalanches occur, ultimately restoring the system toward a balanced state. SOC dynamics are studied in various scientific disciplines, capturing how complex systems naturally settle into equilibrium. The Abelian Sandpile model has been extensively studied for its applicability to a wide range of phenomena, including earthquakes, forest fires, and stock market crashes [90]

Langton's Ant [91], originally proposed by Christopher Langton, is a moving cellular automaton — a special type of CA where the transition function is only applied to one cell at a time, determining both the next state of the transitioning cell, and the cell to be transitioned on the next time step. Despite its remarkably simple rule set, Langton's Ant has been recognised for its ability to demonstrate universal computation, making it capable of simulating any Turing machine. Further studies, including the work by Gajardo et al. [92], have explored the computational properties, patterns, and behaviour of Langton's Ant.

Langton's Loop [42] is a model of self-replication, originally proposed by Langton and later extended by Tempesti [93]. The model provides a simplified framework for exploring the dynamics of autonomous self-replication. It made use of a significantly simpler algorithm, instruction set and number of states, compared to von Nuemann's Universal Constructor.

WireWorld, initially developed by Brian Silverman, is a CA model that gained popularity through a Scientific American article by Dewdney [56]. The introduction of new rules and modifications by Gladkikh et al. [94] turned it into effective tool for simulating and studying computational systems and digital circuitry. In WireWorld, cells can have one of four states: empty, wire, electron head, or electron tail. The behaviour of the system revolves around the movement of electrons through the wires, which allows for the emulation of logical gates and complex electrical circuitry. This is best exemplified by the WireWorld computer [95], a Turing complete and fully programmable computer by David Moore and Mark Owen.

From the 1970s to the mid-1990s, CA and similar lattice-based or discrete methods experienced peak popularity in the simulation of physical phenomena [8]. This widespread adoption was largely due to the relatively low computational complexity of CA, making it well-suited to the hardware limitations of the time. However, with advances in computing power, alternative models such as swarm-based [96] and physics-based simulations [97] gained traction, offering enhanced capabilities for representing complex physical systems that feature interactions between moving agents.

Despite the rise of these alternative approaches, CA continues to be applied in various domains where its discrete, grid-based nature provides unique advantages. For example, CA remains an optimal approach for texture synthesis, where is it still widely used [98]. Additionally, CA is still relevant in disciplines where it can effectively model systems characterised by discrete, localised interactions, such as physical simulations [99] and architectural analysis or synthesis [100], [101].

2.3. CA Exploration

Although numerous CA algorithms have been developed and rigorously studied since the model's inception in the 1940s, exploration of CA rule systems and dynamics has largely been conducted manually. Some algorithms, such as RD

systems, were devised, refined, and studied as models of observed natural phenomena. Others, like Wireworld, were created as simulation tools for computational processes. Still, others, such as GOL, were designed as purely theoretical or mathematical models.

However, there have also been direct attempts at large-scale exploration of CA, several of which were discussed in the previous section. Among the most prominent examples are the works of Langton, Pearson, Eppstein, Wolfram and Sims. Each of these investigations explicitly set out to explore CA dynamics but employed distinct strategies and pursued slightly different goals.

2.3.1 Parametric vs. Algorithmic

Some approaches focus on parametric exploration, where the algorithmic structure remains fixed, and exploration targets the space of numerical parameters. In systems with continuous state parameters, this space becomes effectively infinite [11], potentially offering vast possibilities for dynamic behaviours.

11 *Though, due to the inherent constraints of digital media, this space is ultimately still discrete.*

Langton's parametric exploration of CA [43] aimed to identify a phase transition between ordered and chaotic behaviour, a region he termed the "edge of chaos." This concept became pivotal in demonstrating how simple rule-based systems, like CA, could exhibit behaviours analogous to those found in complex natural systems. By quantifying the balance between order and apparent randomness, this study provided a framework for understanding how complexity arises in both natural and computational contexts.

Similarly, Pearson's parametric strategy [10] enabled simultaneous visual examination of two key parameters within a continuous RD system. Each unique parameter combination produces distinct patterns, forming an instant comprehensive mapping of a wide variety of complex, self-organising structures spanning the entire grid.

Eppstein's Life-like CA notation [70], [71] represents another parametric approach. Retaining the rigid algorithmic structure of GOL, this notation simplifies the process of exploring discrete variations in neighbourhood configurations, as well as birth, death, and survival parameters. Through this framework, a broad set of diverse, GOL-like behaviours can be easily explored and studied.

In contrast to purely parametric methods, other explorations have spanned entire algorithmic spaces. Perhaps the most notable of these is Wolfram's work on ECA [48], [55], which involved a complete enumeration of all one-dimensional, binary-state, two-neighbourhood CA rules. It should be emphasised that conducting such a comprehensive exploration across a complete algorithmic space is only feasible when the subset of CA being studied is extremely small and well-defined. In this case, arguably the smallest possible non-trivial class of CA, featuring only 256 unique algorithms.

Expanding the grid dimensionality, increasing the number of states, or considering larger neighbourhoods would necessarily lead to an exponential growth of the rule space, rendering complete enumerations computationally infeasible, or when considering continuous state — technically impossible. Wolfram's ECA study, therefore, stands as a unique case where the entire algorithmic landscape could be systematically explored and rigorously studied.

2.3.2 Evolutionary Algorithms

In order to explore larger CA rule spaces, researchers have turned to automated computational techniques, such as evolutionary algorithms. These methods enable exploration of vast algorithmic landscapes, uncovering emergent behaviours that would be infeasible to come across through manual exploration. Sims described a method for evolving two-dimensional CA [102]. This involved creating a random population of CA transition rules, defined as lookup tables. He allowed users to observe their execution and select those exhibiting interesting or complex patterns. Those selected were then subjected to reproduction with variation, forming a progressive process of evolution towards more aesthetically pleasing and increasingly intricate behaviours over successive generations. [12]

Other studies demonstrate how evolutionary algorithms, guided by aesthetic fitness criteria, can effectively explore complex, emergent behaviours across scientific and artistic domains. Ashlock [103] applied evolutionary algorithms to steer CA toward visually compelling patterns, while Forbes [104] developed interactive CA systems that emphasise user-driven manipulation of rules to generate novel artistic outputs. Heaton [105] further showcased how continuous CA can be fine-tuned to create dynamic, flowing visuals that transcend the rigid, grid-based structures typical of traditional CA.

Evolutionary algorithms offer a powerful strategy for navigating vast algorithmic spaces, however their efficacy largely depends on the definition of a fitness function — a process that inherently excludes almost all potential variants. In interactive evolutionary algorithms, fitness is determined through aesthetic preference, allowing human intuition to guide the selection process in an effectively open-ended manner. Conversely, automated evolutionary approaches are employed in cases where fitness can be quantified and tested, using metrics such as pattern recognition, stability, or entropy. While these methods are significantly faster, as they eliminate the need for manual evaluation, the reliance on quantifiable fitness criteria often directs exploration towards more goal-oriented research.

For example, Mitchell et al. [106] and Das et al. [107] evolved CA to perform computational tasks like density classification and global synchronisation, with fitness functions measuring task accuracy and stability. Similarly, Sipper [108] introduced cellular programming, where localised fitness assessments evolved CA for parallel computation. Packard [109] and Juillé & Pollack [110] employed entropy measures and coevolutionary dynamics to evolve CA exhibiting complex behaviours. While these studies uncovered novel computational properties and, in some cases, emergent behaviours, their fitness-driven focus inherently constrained exploration to specific, predefined objectives, limiting their potential for broader, open-ended discovery.

2.3.3 Custom Hardware and Machine Learning

Parallel to the development of algorithmic exploration techniques, hardware-specific solutions like CAM-6 facilitated real-time CA experimentation. Developed by Tommaso Toffoli and Norman Margolus at MIT in the mid-1980s [111], CAM-6 was a plug-in board for IBM PCs designed to accelerate two-dimensional CA simulations. This ad-hoc hardware enabled rapid, interactive exploration of CA dynamics, providing a computational environment far more efficient than the general-purpose computers of the time.

CAM-6 allowed users to manipulate CA parameters and observe emergent behaviours in real-time, offering a hands-on approach to studying complex systems. While this was primarily a scientific tool, its emphasis on real-time,

12 *Sims also applied this evolutionary approach to continuous dynamical systems, using hierarchical Lisp expressions to define differential equations. This method serves as a notable precursor to the framework presented in this thesis, which similarly employs nested Lisp notation — though applied to defining CA transition functions rather than continuous dynamical systems.*

visual interaction opened new pathways for artistic explorations of CA, blurring the lines between scientific modelling and creative experimentation.

More recently, explorations of CA have expanded through hybrid frameworks that incorporate CA with machine learning techniques. A novel class of CA known as Neural Cellular Automata (NCA) has emerged, utilising neural networks to generate complex, previously unseen behaviours within CA systems. For instance, Mordvintsev et al. [112] demonstrated NCA's ability to autonomously regenerate intricate, predefined multicellular patterns in two dimensions. Niklasson et al. [113] applied NCA to texture synthesis, successfully replicating the general appearance of various predefined textures. Furthermore, Earle et al. [114] utilised NCA for procedural level generation in video games, showcasing the generative potential of CA in commercial and creative contexts. These advancements represent new frontiers for CA research, where the exploration of novel CA algorithms — regardless of their original scientific purpose — gains relevance in fields ranging from artificial life to digital art and game design.

Summary

Exploration of CA has evolved from manual study of simple rule-based models to the use of sophisticated computational techniques for navigating vast algorithmic spaces. Parametric approaches demonstrate how slight variations in numerical parameters can yield a wide range of behaviours within a fixed algorithmic structure. In contrast, exhaustive algorithmic explorations are possible but inherently limited to very small, well-defined rule spaces. Evolutionary algorithms provide a powerful means of traversing larger rule spaces, yet their outcomes remain constrained by either aesthetic preferences, in the case of interactive systems, or predefined objectives in the case of automated evaluation. Emerging techniques, such as NCA, show great promise in uncovering novel CA dynamics that generate complex, previously unseen behaviours. However, it remains to be seen whether these methods will be applied to truly open-ended exploration.

This section has reviewed a range of strategies for exploring CA dynamics and rule systems over time. While not exhaustive, it offers a broad perspective on the diverse methodologies employed in CA research. The following section widens the scope to examine the work of notable practitioners who engage in algorithmic exploration across various disciplines, demonstrating algorithmic explorations that extend beyond CA into the broader realms of science, art, and design.

2.4. Notable Practitioners

The following section presents the work of a number of creative practitioners and researchers who employ the use of CA and related algorithmic approaches towards exploration of emergent systems. Their work often ventures beyond the realms of scientific research to encompass artistic, educational, or theoretical practices. Many of these individuals have developed their own custom software tools and have made these publicly accessible, affording others the same freedom of expression. Particular attention is given to projects that are most relevant to this thesis, namely, those that prioritise novelty and aesthetic experience over practical applications.

This review representes a curated body of work that is speculative, non-representational, abstract or otherwise detached from physical reality. It is important to note that the selection criteria is highly subjective. Featured here are works whose approach and methodology towards the study of emergent

phenomena have substantially influenced and inspired this thesis. The bulk of literature on CA and AL may appear to lack sufficient emphasis on artistic expression and exploration, as noted by Aguilar et. al. [115] [13]. Nonetheless, the compilation of works presented here is meant to indicate at least a partial fulfilment of Langton's vision for "something more like poetry" in the future of artificial life.

13 The Authors classify AL research into 14 themes and rate their popularity according to publications in MIT's "Artificial Life" journal between 1993 and 2014. They note that "Some themes are poorly represented, such as art, because artists usually choose different venues to publicise their work"

Karl Sims is widely regarded as one of the early pioneers of AL, with influential contributions spanning both artistic and scientific domains. Beyond his previously discussed projects on CA, Sims' diverse body of work includes advancements in particle systems [116], genetic algorithms [117], [118], image processing [119], fractal geometry and fluid dynamics [120], as well as interactive installations [121], [122]. He also developed an online tool for interactive exploration of RD, based on Pearson's parameterisation [123], which is particularly relevant to this thesis.

Andy Lomas' work explores abstract generative forms, morphogenesis and emergent processes, often featuring large scale 3D systems with millions of interacting components. His 2005 aggregation series [124] featured explorations of Diffusion Limited Aggregation (DLA) applied at a significantly larger scale than previous implementations of this approach [125]. In Cellular Forms [126] Lomas developed a novel approach for cell replication and organisation at a local level. This methodology was further developed and perfected in Hybrid Forms [127]. A more recent collaboration with Jon McCormack features methods for exploration of emergent forms and processes using evolutionary algorithms and machine learning [128], [129]. The emergent structures typically featured in Lomas' work seem to strike a careful balance between biomorphic and abstract.

Tim Hutton's work on self-replication and artificial chemistry rely heavily on exploration of novel CA algorithms and feature a high degree of analytical accounting of his findings [130], [131]. Hutton is also a co-author of the Golly [132] and Ready [133] software packages, which allow exploration of Cellular Automata and Reaction Diffusion algorithms respectively (see next section). Hutton's body of work is situated at a unique junction of exploration, problem solving and tool making. His interests span multiple disciplines related to exploring, utilising and rendering emergent behaviour within virtual environments.

Sage Jensen explores speculative forms of Physarum Polycephalum (slime mould) patterns in virtual environments [134]. They use a hybrid technique based on a method developed by Jones [135]. This approach combines two separate systems: one layer featuring a swarm of freely moving agents in continuous space and a second layer featuring an orthogonal discrete grid of cells. This approach is also reminiscent of the multi-agent programmable modelling environment netLOGO by Uri Wilensky [136]. Jensen develops custom software using C++/Openframeworks/GLSL, utilising hardware acceleration and parallel processing where possible. The output often bears striking visual resemblance to the behaviour and appearance of its physical subject matter. However it can also be seen as abstract and removed from any particular physical process. Taking this creative liberty in representation allows Jensen to focus on the aesthetic aspects of the phenomenon while pushing forward its creative, visual and aesthetic potential.

Casey Reas is a co-creator of the Processing programming environment which originated at the MIT Media Lab in 1999. As such he has become a prominent figure in the modern creative coding movement of the 2000's. He is a published author on both tools [137] and their potential output [138]. In his artistic

practice, Reas explores abstract forms driven by simple dynamical, multi-agent systems for large scale installations [139], [140] and print work [141].

Other notable explorations of CA include Ben Kraakman's collection of CA experiments featuring Multiple neighbourhood CA (MNCA) [142] [143] (also refer to VulcanAutomata in the subsequent section); experiments by YouTube user CellularAutomataUploader [144]; Simon Alexander-Adams' 3D cellular automata experiments and tutorial [145], [146]; Artificial Life software experiments and shader programming workshops by Arsiliath [147]; experimental video documentations, software engines and algorithmic expressions by Luke Wilson [148]; and generative landscapes and mathematically evolving structures by multimedia artist and composer Jazer Giles [149], [150].

Other notable artworks which explore emergent phenomena (not necessarily using CA) include glitch and ASCII art by Kermi Safa [151], Kim Asendorf [152] and Julian Hespenheide [153]. Andreas Gysin's pattern and ASCII formations for installation and net-art [154]; algorithmic playgrounds by Alex Miller & Alex Nagy [155]; Clusters - a particle micro world with ambiguous entities by Jeffery Ventrella [156]; Particle and fluid based installations by Josef Pelz [157]; generative designs of Sander Sturing [158]; LED wall installations by @codeofconquer [159]; Andreas Hoff's generative web experiments and tutorial series [160]; generative jewellery by Nervous System [161];

On a broader spectrum of generative systems, notable exploratory works include interactive software experiments by digital artist LIA, which often feature multi agent systems and emergent patterns [162]; The minimalist and organic Flash applets of Laurie and Jared Tarbell [163]; genealogical studies, data driven virtual sculptures and artificial life experiments by Norman Leto in his feature length film SAILOR [164], [165]; speculative biomorphic wearable objects by Neri Oxman [166] and Filippo Nassetti [167] and generative sculptures by Driessens and Verstappen [168]; compound fractal geometry explorations and software tools by Tom Beddard [169]; and the art-science documentary films by Ruth Jarman and Joe Gerhardt which often employ a hybrid of data driven and generative methods [170].

2.5. Notable Tools

The following section reviews a range of software packages, frameworks, and code libraries that can be used to implement CA, as well as other computational models such as particles, swarms, and networks. These tools span a broad spectrum of functionality, generality, and scope, catering to different target audiences and use cases. While all of the tools discussed can be applied to CA exploration, not all were explicitly designed with this purpose in mind.

Arranged from the most general-purpose frameworks to the most specialised CA tools, each comes with a distinct trade-off between generality and specialisation. This trade-off, while common in software development, appears to be particularly pronounced in the context of CA research, which has come to encompass multiple distinct models without a clear mainstream software or unified paradigm. The following review examines how different tools navigate this balance, highlighting their strengths and limitations. The next section identifies the research gap, building on the insights gathered from this review.

General Purpose Frameworks

A creative coding library can be thought of as a collection of functions that abstract many common, often cumbersome tasks related to computer graphics,

such as plotting, vector and matrix operations, trigonometry, image processing, and 3D rendering. The use of such libraries typically requires literacy in their underlying programming languages, which may present a notable barrier for some creative practitioners.

Processing [171], a Java-based programming environment developed at MIT in 2001, was originally designed as a teaching tool and software sketchbook. It has since evolved into a reliable, open-source framework for interactive applications and installations. Beyond its core components, Processing features an extensive library of add-ons that significantly extend its functionality. More recently, it has been adapted into the web-based JavaScript library P5.js, significantly broadening its accessibility for web development. [172], [173]

OpenFrameworks [174] brings Processing's design principles to C++, offering a more robust, expandable, and production-oriented programming environment. Similarly, Three.js [175], initiated by Ricardo Cabello, is a powerful JavaScript 3D graphics library that provides advanced tools for web-based graphics and interaction.

Some applications and authoring tools provide alternatives to text-based programming, offering more intuitive interfaces for artists and designers. For example, Cycling '74's Max/MSP [176] and its open-source counterpart PureData [177], [178] have opened new avenues for algorithmic exploration in digital music and multimedia art. Both environments utilise a node-based visual programming paradigm, commonly referred to as *data-flow programming*. In this paradigm, patch cords are used to connect algorithmic operators, creating a visual representation of the flow of data through a system. Over time, data-flow programming has gained widespread popularity and has been extended beyond music into visual art, interactive installations, and real-time graphics.

TouchDesigner [179], developed by Derivative, is a node-based visual programming environment designed for real-time interactive multimedia content. Initially popular in live performances and projection mapping, it has become a versatile platform for generative art, data visualisation, and interactive installations. TouchDesigner leverages a highly modular UI, allowing users to design complex visual systems without traditional coding. While it is not explicitly designed for CA, its ability to manipulate real-time graphics, shader programming, and procedural logic makes it suitable for creating custom CA models, particularly in artistic contexts, though it may not be as user-friendly for large-scale algorithmic exploration compared to more specialised tools.

3D Applications

Another class of software that supports algorithmic exploration is game engines, such as Unreal Engine [180], Unity [181], and Godot [182]. While primarily designed for game development, their support for procedural content generation, real-time interaction, and programmable shaders makes them applicable for broader computational art and design contexts. However, due to their heavy footprint, game engines are not the most efficient platforms for large-scale CA simulations. CA-specific use cases in game engines would typically be applied to terrain or biome generation, visual effects, and texture synthesis, rather than conducting research.

A number of 3D design and animation packages have integrated data-flow programming interfaces to support procedural generation and algorithmic modelling. These tools provide powerful environments for exploring complex geometries and dynamical systems, which can be applied to visualising and experimenting with CA, particularly in three-dimensional contexts. While their focus on procedural workflows aligns well with CA principles, these tools are

primarily designed for visual effects, simulation, or CAD design, rather than dedicated CA research.

Grasshopper [183], a longstanding tool in this category, is integrated with Rhinoceros 3D [184]. Focused on architectural design and parametric modelling, it allows designers to create algorithmically-driven structures. Houdini [185] is renowned for its procedural generation capabilities and is widely used in visual effects and 3D animation. It excels in creating complex simulations, including fractals, particle systems, and fluid dynamics. More recently, Blender's Geometry Nodes [186] has emerged as a popular, extensible tool, offering a rich node-based programming interface for procedural modelling, simulation, and parametric design. Its flexibility makes it a suitable platform for conducting CA experiments in 3D space.

Multi Paradigm tools

Wolfram Mathematica [187] occupies a unique position between general-purpose computational frameworks and specialised CA tools. Primarily designed for symbolic computation and mathematical modelling, Mathematica offers built-in capabilities for CA exploration, reflecting Stephen Wolfram's foundational contributions to the field. Mathematica's strengths in numerical analysis and data visualisation make it an effective tool for both theoretical research and visual exploration of CA dynamics. However, its general-purpose design introduces certain trade-offs: performance can be limited in large-scale CA simulations, and it lacks real-time interactivity, which can constrain exploratory workflows.

NetLogo [136] is an agent-based modelling environment originally developed by Uri Wilensky in 1994. It features a hybrid model that combines a fixed cellular grid with a system of moving agents. These two layers can operate independently or interact dynamically, enabling implementation of a wide range of computational models, including CA, swarm intelligence, and neural networks. NetLogo comes with an extensive library of pre-built scientific models and a user-friendly interface, making it accessible to both researchers and educators. While its primary focus is on agent-based modelling, its flexibility allows for meaningful CA experimentation. More recently, NetLogo has been ported to the web [188], broadening its accessibility and facilitating online simulation sharing.

Visions of Chaos by Jason Rampe [189] is a comprehensive application for exploring chaotic systems and complex dynamics. It supports an unusually broad spectrum of models, including CA, agent-based models, fractals, diffusion-limited aggregation (DLA), fluid dynamics, particle simulations, and even machine learning algorithms. Visions of Chaos features a graphical UI, hardware-accelerated rendering, and a vast library of pre-configured examples for each model type. While it excels as a visualisation tool with extensive configurability, it is best suited for exploring existing models rather than deep algorithmic customisation, positioning it as an educational and exploratory platform rather than a dedicated CA research tool.

Specialised Software

There are a number of software packages explicitly designed for CA research, offering dedicated tools for exploration, study and analysis. These packages are often optimised for performance and come bundled with an extensive set of relevant examples. However, their specialised nature may impose limitations on the range of algorithms they can effectively handle, constraining them to specific use cases, and their output to specific families of algorithms.

Golly [132] is a desktop software application designed for executing and interacting with various cellular automata algorithms. It features a graphical interface which makes it highly suited for individuals with no programming experience. However, it is important to note that the algorithms implemented within Golly are optimised for performance through hashing techniques. While this optimisation enhances computational efficiency, it also poses practical challenges when it comes to developing or modifying algorithms within the software. Consequently, the process of algorithm development or customisation within Golly may prove to be impractical for users seeking extensive modifications or novel algorithmic implementations. [14]

***14** See [147] for a comprehensive online list of algorithms bundled with Golly*

Ready [133] follows a similar approach to Golly and applies it to continuous state CA. The software is explicitly designed for conducting experiments with reaction-diffusion systems in both two-dimensional and three-dimensional spaces. Similarly to Golly, Ready is bundled with a collection of pre-existing algorithms, providing users with a solid foundation from which to conduct further explorations. Algorithms within Ready are written as OpenCL kernels. This means modifying or developing algorithms necessitates advanced programming skills and familiarity with the programmable shader pipeline.

DDLab by Andy Wuensche [190], [191] is an interactive graphics software for researching discrete dynamical networks. Aimed towards the field of experimental mathematics. The package can construct, visualise, manipulate and analyse a broad class of discrete systems, including CA, random boolean networks, discrete dynamical networks and random maps.

VulkanAutomata by Ben Kraakman [192] is designed for exploring Multiple Neighbourhood Cellular Automata (MNCA): a class of CA that uses large custom neighbourhoods in parallel to produce robust emergent structures [193]. The application allows exploration of a wide range of algorithms created by the author, as well as randomly generated algorithms of this class. MNCA appears capable of yielding exceptionally complex and resilient high order structures, reminiscent of algorithms such as Lenia [87] and Smoothlife [69].

Neural patterns by Max Robinson [194] is a web toy for exploring Neural Cellular Automata (NCA). It uses convolution and activation functions for cell transitions. It comes with a number of examples and allows simple live coding of rules. The Life Engine [195], also created by Robinson, is an online virtual ecosystem that allows organisms to reproduce, compete, and evolve.

Other notable specialised software packages include CA Lab [196], a legacy software originally released for DOS in 1989 by Rudy Rucker and John Walker. It was rewritten for Windows in 1996 by Walker and again in Javascript in 2017. Life Viewer by Chris Rowett [197], a browser based scriptable pattern viewer for GOL and a number of other 1D and 2D CA. Lastly, CellPyLib by Luis M. Antunes [198], a python library for defining and analysing 1D and 2D CA.

2.6. Research Gap and Motivation

The author's 2015 MA thesis project, Semicolony [199], consisted of an ontology of software experiments, developed over the course of a year and assembled as a "laboratory for the study of fictional organisms." Strongly influenced by early research in AL, the project employed a diverse range of algorithmic approaches, including swarm dynamics, Superformulae, L-systems, evolutionary algorithms, and CA. Contextualised as a computational arts project, Semicolony made significant attempts to detach these algorithms from their real-world counterparts, exploring their aesthetic, formal and interactive potential rather than their traditional scientific interpretations.

While the project successfully showcased the creative and interactive potential of the algorithmic systems it studied, it also faced significant limitations in terms of algorithmic exploration, real-time interaction, and the inherent trade-off between generality and specificity in available tools. These challenges not only revealed constraints within the project itself but also pointed to broader gaps, as they relate to the integration of emergent systems within creative media.

2.6.1 Generality vs. Specificity

During the initial CA experiments featured in Semicolony — from which this thesis ultimately derives [15] — many of the software tools reviewed in the previous section were considered. This process revealed a notable trade-off between generality and specificity in existing tools for CA research.

15 This experiment is discussed in detail in Section 4.2

The review covers a spectrum from general-purpose frameworks to highly specialised tools, each offering distinct advantages and presenting distinct limitations. General-purpose frameworks like Processing, OpenFrameworks, and Three.js offer broad flexibility, enabling users to build custom CA models from the ground up. However, this flexibility comes with inherent limitations. Adapting these tools for CA exploration requires significant technical effort, including expertise in software engineering, user interface design, and graphics programming. This represents resources that may be unavailable for researchers focused on the theoretical, artistic, or experimental aspects of CA rather than technical development.

Another limitation of general-purpose frameworks is the scarcity of advanced examples and starting templates. For instance, many CA implementations in Processing are designed primarily for teaching programming concepts rather than facilitating CA exploration, resulting in basic and often sluggish functionalities. [16] Furthermore, most available examples are typically limited to the most popular models like GOL and ECA, leaving limited opportunities to explore more complex or novel CA.

***16** Exceptions, like the Conway shader example included in the Processing IDE, are rare.*

Conversely, specialised software tools like Golly and Ready provide robust, feature-rich environments tailored for CA exploration. However, their focus on specific use cases and algorithmic structures limits their broader applicability. For example, Golly excels in exploring configuration patterns and includes an extensive library of CA algorithms. Yet, it lacks support for continuous-state systems, is poorly suited for real-time parametric or algorithmic exploration, and does not allow for the concurrent execution of multiple algorithms. Similarly, while Ready offers powerful tools for analysing continuous-state CA, its reliance on OpenCL kernels for defining transition functions makes it cumbersome to prototype new rule systems or explore algorithmic variations.

3D applications, especially those with integrated parametric design tools like Blender's Geometry Nodes, offer promising environments for experimenting with CA. However, these platforms are primarily designed for CAD or 3D animation, and adapting them for CA exploration often requires substantial workarounds, especially in terms of algorithmic expression. Additionally, their feature-rich environments come with a steep learning curve, posing barriers for researchers and artists focused on algorithmic exploration rather than 3D modelling.

Multi-paradigm tools such as NetLogo and Visions of Chaos offer a wide array of models but are not optimised for deep algorithmic exploration or customisation. These tools excel in providing ready-made models and flexible environments for

simulation, but fall short as platforms for building or modifying algorithms from scratch.

2.6.2 Algorithmic Exploration

A central challenge in Semicolony was the difficulty of engaging in true algorithmic exploration. The project's focus on novel emergent phenomena required extensive investigation of algorithmic behaviours. To facilitate this, most experiments were implemented using OpenFrameworks, necessitating the development of custom software from scratch. Despite these efforts, exploration remained largely confined to adjusting numerical parameters and reconsidering visual representations of well-established algorithms, rather than modifying the underlying algorithms themselves or creating entirely new ones.

This limitation reflects a broader trend in the use of computational modelling within creative media, where exploration often relies on superficial adjustments rather than deep algorithmic modification. For instance, many creative implementations of the Boids algorithm [96] primarily experiment with diverse visual representations, or explore variations in numerical parameters, such as tweaking the model's separation, cohesion, and alignment behaviours. However, the core structure of the algorithm typically remains untouched. [17]

This reliance on parametric and aesthetic exploration often stems not from a lack of interest in alternative boid-like dynamics, but from technical barriers. Modifying the underlying algorithmic structure of a model requires not only significant programming expertise but also a solid understanding of the dynamical system it represents. Moreover, traditional programming tools and syntax are generally structured with the assumption that the programmer has a clear goal in mind. While highly effective for precise, goal-oriented development, this assumption can pose challenges for open-ended algorithmic exploration, where flexibility and iterative discovery are key.

The absence of tools designed for true algorithmic exploration — beyond aesthetic or parametric adjustments — highlights a key challenge in both computational arts and CA research. Existing tools often demand extensive technical expertise or confine users to surface-level modifications, creating barriers to both scientific inquiry and artistic experimentation. Addressing this gap is a core objective of this thesis, which aims to develop novel frameworks and methods for CA exploration that enable flexible, open-ended engagement with algorithms across both domains.

2.6.3 Real-time Control

Another significant challenge encountered in Semicolony was managing the sensitivity of real-time control in emergent systems. The project required interactivity on multiple levels: as a means for exploration through immediate visual feedback, as part of interactive installations, and within live performance contexts. [18] Each of these scenarios introduced unique challenges in negotiating between meaningful user influence and the inherent unpredictability of emergent behaviours.

In the exploratory phase, interactivity is crucial for rapidly iterating on algorithmic behaviours. Immediate visual feedback allows for quick assessment of system dynamics, but this process is often hampered by tools that require compilation or otherwise lack real-time modification capabilities.[19] This limitation forces practitioners to work within relatively narrow constraints, as it imposes a strict separation between devising new behaviours and observing them in action.

17 *This echoes the discussion in section 2.3.2 regarding the Life-Like notation, which makes it easy to explore a large number of parametric variations of GOL but leaves the core algorithm as-is.*

18 *A software experiment from Semicolony, Colony Type-A, was featured in The Infinite Bridge, a multidisciplinary live performance in 2015. [200]*

19 *While some tools offer parametric flexibility, many fail to support live coding for algorithmic adjustments. A few exceptions — such as Visions of Chaos, Neural Patterns, VulkanAutomata, and scripting environments like Golly — allow for real-time parameter tweaking and limited*

In the context of live performance, emergent systems often require a delicate balance of control. A performer needs enough influence to steer the system toward desired behaviours, while maintaining the inherent unpredictability that makes these systems compelling. Too much control can render the system rigid and predictable, while too little can lead to chaotic behaviours that would easily veer into undesirable or irrecoverable states.

Similarly, in interactive installations, user interaction often drives system behaviour. However, unlike in live performances — where a trained performer can adapt to unexpected outcomes — installations engage a general audience unfamiliar with the system's intricacies. In this context, it becomes even more critical to maintain behaviours within reasonable bounds, ensuring that user interactions lead to engaging, but manageable, outcomes. Failure to control emergent behaviours in the context of interactive installations risks confusing, frustrating or even harming participants.

This teeter-totter dynamic — the constant push and pull between control and emergence — is a fundamental challenge in developing software for exploration of emergent systems, including CA. It underscores the need for tools and frameworks that support nuanced, real-time interaction with emergent behaviours, allowing users to guide outcomes without overpowering the system's intrinsic dynamics. Addressing this challenge directly informs the practical objectives of this work, as elaborated Chapter 3.

Summary

The limitations encountered in Semicolony, along with broader constraints in existing CA tools, indicate a clear research gap: the lack of software designed for open-ended, real-time CA exploration. General-purpose frameworks offer flexibility but lack features tailored for CA synthesis and analysis, while specialised CA software provides robust tools but remains restricted to specific algorithms and use cases. This gap affects both scientific inquiry and creative experimentation, underscoring the need for a framework that bridges the adaptability of general platforms with the specialised capabilities of CA-focused tools. Such a framework would enable researchers and artists to explore parametric and algorithmic variations, work with both continuous and discrete systems, and engage in real-time interaction without requiring extensive programming expertise.

2.6.4 Artistic Objectives

Despite extraordinary increases in computing power, few systematic efforts — beyond Wolfram's early study of ECA — have attempted to systematically explore combinatorial spaces of CA algorithms. Existing approaches barely scratch the surface, offering no real reference point for their diversity or sheer scale. At the heart of this thesis lies a fundamental drive: exploration. The act of exploring emergent systems is not merely a methodological pursuit but an artistic vision in itself. The vastness of combinatorial algorithmic spaces both forms and informs the conceptual foundation of this work, while simultaneously presenting a profound technical and creative challenge.

While the technical contributions of this thesis, which include the development of tools and frameworks specifically designed for CA exploration, fill a critical research gap, they also serve a deeper artistic purpose. The success of this work, therefore, should not be measured solely by the utility of the tools or the novelty of the algorithms discovered. It should also be evaluated in terms of its ability to offer a new perspective on the vastness of algorithmic spaces and their endless

rule modifications. However, these tools are either constrained in scope or require significant setup to facilitate open-ended algorithmic exploration, particularly when it comes to modifying core system dynamics without interrupting execution.

potential. An aesthetic engagement with the abyss of possible emergent phenomena is central to the artistic objectives of this work. It reflects a deliberate attempt to confront the unknowable, evoking a sense of wonder while coping with the cosmic terror that these infinite landscapes can induce.

This thesis aligns with a broader discourse in Arts and Computing Technology by regarding algorithmic exploration as both a scientific and artistic act. The tools, frameworks, and methods introduced in the following chapters are designed not just for structured inquiry, but for wholesale discovery — setting out to explore inherently unexplorable domains. Moreover, this work consciously avoids grounding itself in practical applications. In doing so, it aspires to minimise the temptation to explore with predefined goals or utilitarian outcomes. This is a deliberate artistic choice, intended to encourage a consideration of the larger space of potential outcomes.

Looking forward, the tools and strategies developed here are part of a long-term artistic pursuit. The aim is to build a foundation that maximises the diversity of artistic outcomes, both in form and process. This diversity would ideally serve as a benchmark for the success of these tools, reflecting their capacity to facilitate genuine, unbounded exploration of virtual phenomena. The specific aesthetics and properties of future outcomes, naturally, remain to be seen. But the aspiration is clear: to create a space where the author — and, hopefully, others to come — can continuously discover novel emergent phenomena.

In this way, the artistic objectives of this thesis transcend individual discoveries. They seek to evoke an aesthetic experience rooted in confrontation with the unknown.

2.7. Summary

This chapter provides a comprehensive introduction to key concepts, themes and methodologies related to computational modelling and CA within the context of AL. It begins by establishing the role of computational models as tools for simulating the behaviour of nonlinear systems, characterised as systems of interacting components. The chapter discusses the concept of emergence, a key feature of such systems, and distinguishes between simulation and realisation within the strong-weak debate in AL. This discussion aims to reframe the role of computational models, emphasising the potential value of open-ended exploration of emergent virtual phenomena over traditional representational approaches.

The chapter also provides an in-depth introduction to CA, providing historical context, key terminology, and a review of influential algorithms and applications. This highlights the potential of CA as abstract models for studying nonlinear dynamical systems, advocating for their use beyond conventional scientific modelling. Further, the work of notable practitioners and researchers who explore emergent virtual phenomena as part of their scientific or creative practice is examined. This review illustrates the interdisciplinary appeal of CA and lays out the research gap, setting the stage for the novel computational frameworks and methods proposed in this thesis.

The introduction to CA, its contextualisation within the context of AL, and the reviews of notable explorers and tools, all serve as the conceptual and technical foundations for the chapters to come. Chapter 3 details the development of a new computational framework for CA exploration. Its effectiveness in supporting a wide range of CA algorithms is demonstrated by implementing three notable algorithms reviewed in this chapter: Conway's Game of Life,

Elementary CA and Gray-Scott Reaction Diffusion. These algorithms are also used to showcase the Spatial Mapping method presented in Chapter 5.

Chapters 4, 6 and 7 present, discuss and evaluate algorithms, uncovered using the above computational frameworks. These cahpters constitute exploratory studies that demonstrate this project's non-instrumentalist approach, which is largely inspired by strong-AL. Chapter 8 contextualises and discusses these findings in regards to the philosophical views discussed in this chapter and evaluates them in light of the project's research aims. Lastly, a number of avenues for future research are suggested, encompassing both focused continuations of this work, as well as broader studies of CA and other computational models.

3.Computational Framework

This chapter introduces *Utomata*, a novel computational framework, designed for open-ended exploration of CA algorithms. Utomata aims to bridge the gap between artistic exploration and computational modelling by enabling implementation of a wide range of CA algorithms and allowing real-time manipulation of their behaviour. The framework is readily accessible online as an open-source Javascript library [201]. It employs a custom functional syntax that encourages a non-analytical approach to CA programming, which is suggested to enhance its expressive power and effectiveness for open-ended exploration of novel algorithms. This approach can reduce the reliance on prerequisite knowledge in computer and natural sciences, potentially making CA exploration more accessible to creative practitioners.

Sections 3.1 and 3.2 discuss the rationale and design principles upon which Utomata was developed. Section 3.3 offers a formal definition of its components and syntax, and details a number of unique software design decisions. Section 3.4 demonstrates how Utomata can be effectively utilised to implement and study three well-established algorithms, discussed in the previous chapter: Conway's Game of Life (GOL), Wolfram's Elementary CA (ECA), and Gray-Scott reaction-diffusion (RD).

3.1. Problem statement

The previous chapter reviews various software tools that can be used for exploration and study of CA. These can be categorised into three primary groups: General-purpose frameworks, multi-paradigm tools, and specialised software. The first category encompasses creative coding libraries such as Processing, Openframeworks, and P5js. These libraries, coded in different programming languages (Java, C++ and JavaScript, respectively), provide extensive code bases that abstract and consolidate common operations for graphics, sound, and user interaction. While their versatility makes them highly suitable for open-ended creative tasks, they typically lack optimisation and examples specifically related to implementation or exploration of CA dynamics. CA related examples for these frameworks typically involve implementations of either GOL or ECA.

Moreover, these examples often prioritise the educational aspects of programming a CA system through the use of variables, conditionals, arrays, and loops, rather than promoting exploration or examination of CA dynamics. For instance, Daniel Shiffman's P5js GOL implementation [202] comprises 88 lines of code, only 5 of which are dedicated to describing the transition function of GOL. Additionally, as these examples typically run on the CPU rather than the GPU, they are suitable for systems of limited size, often around 100 by 100 cells, which is not sufficient for supporting the emergence of complex structures in many cases.

The second category, represented by tools such as NetLogo [136] are general-purpose computational modelling environments with a strong educational orientation. They provide an extensive set of built-in CA models, making them accessible to both novices and educators. However, their primary focus is on facilitating learning and conceptual understanding of complex systems rather than fostering open-ended, creative exploration.

The third category encompasses software packages explicitly designed for CA, such as Golly [132], Ready [133], Life Engine [195], and VulcanAutomata [192]. These tools, unlike general-purpose frameworks, prioritise the efficient operation of large systems and are equipped with an extensive set of examples. However, these are tailored to specific domains or subsets of CA algorithms and, consequently, may not be well-suited for creative applications or exploration beyond these domains. For example, as its name suggests, Golly places a heavy focus on GOL and its many variants. Similarly, Ready focuses on RD-like systems and VulcanAutomata is specifically designed for exploration of *Multiple Neighborhood Cellular Automata* (MNCA), a unique class of algorithms formulated and explored by the software's author.

It is therefore suggested that creative practitioners and researchers aiming to embark on open-ended explorations of CA within creative or non-analytical contexts are faced with a dilemma. They must currently choose between a general-purpose approach, which offers versatility, but demands substantial preparatory work and prerequisite knowledge, or a specialised approach, which would streamline much of the process, buy may introduce significant constraints into creative or exploratory work.

3.2. Design Goals

This section outlines a set of design principles for a software framework designed explicitly for open-ended exploration of CA. These principles have been formulated alongside practical explorations and studies conducted throughout the practice based portion of this thesis. The next chapter describes such an exploratory study in detail and presents key findings.

General

Support a wide range of CA algorithms.
CA algorithms exhibit significant variation in state types, dimensionality, neighbourhoods, algorithmic expressions, and morphologies. Despite these differences, a large number of CA models share common properties. A substantial number of models consist of a 1D or 2D grid of non-moving cells, each associated with one or more numerical values. The proposed framework should encompass as many of these shared properties to enable the implementation of a wide range of CA algorithms.

Consistent

Utilise a unified form of algorithmic expression.
A robust and unified programming syntax would facilitate comparison, combination, analysis, allow systematic archival of CA algorithms, as well as enable large collections of algorithms to be formulated. This unification should transcend particular programming environments, offering a language agnostic representation of algorithmic expressions. A low-level mathematical approach would facilitate the above, and may also enable meta-programming techniques such as genetic programming and procedural generation of CA algorithms.

Useful

Empower exploration and examination of novel algorithms.
A framework designed for open-ended exploration should prioritise synthesis over analysis, supporting playful, expressive, and improvisational modes of programming. It should encourage accidental discovery and real-time mutability by allowing fast examination of multiple variations through visual, interactive or automated methods. A prioritisation of synthesis over analysis also has the potential to extend the outreach of CA exploration. Increased

engagement by creative practitioners may, in turn, promote a mass diversification of CA algorithms, similar to how music programming tools such as Max/MSP [176] helped to promote a mass diversification in sound design and electronic music.

Cross-Disciplinary

Accommodate a wide range of use cases.
The framework should accommodate, as much as possible, users with varying backgrounds and levels of expertise. It should constitute a foundational framework upon which a range of custom software tools can be developed for different contexts and use cases. Some of these tools would accommodate improvisation and intuition based studies that are grounded in subjective judgement and are aimed towards artists, designers, hobbyists and students. Other tools may allow precise examination, repetition and focused studies, allowing engagement by scientists, researchers and educators. Accommodating a diverse community of practitioners may be crucial for the adoption and long term success of the framework.

Accessible

Easy to access, share and extend.
The framework should be open-source, lightweight, highly extensible and allow implementation in multiple programming languages and environments. Additionally, emphasis should be placed on a web-based implementation, so that algorithms and derivative software could be freely and easily embedded and shared online. A language agnostic programming syntax is crucial for creating a standard form of algorithmic expression that can be easily transferred between environments.

3.3. Definitions

Utomata is a software framework for interactive explorations of CA, designed for creative media, education and procedural content generation. It features a set of unique variables, functions and and operators, as well as a custom programming syntax for succinct representation of CA configuration and transition functions. This section presents a technical clarification of how Utomata handles fundamental components of CA, including the grid, state, configuration, transition function, operators, and neighbourhoods. It is complemented by a javascript/WebGL implementation which has been made freely accessible online by the author in 2019, and has been steadily maintained by the author since then.

Topology

In Utomata, a finite, two-dimensional, orthogonal *grid* forms the foundational structure of any system. This grid consists of discrete entities, referred to as cells. Each cell's position is denoted by a normalised 2D coordinate, with the top-left cell positioned at [0.0, 0.0], and the bottom-right cell at [1.0, 1.0]. [1] The boundaries of the grid are configurable in one of two modes: toroidal or clamped. Under toroidal bounds, the spatial domain wraps along both the horizontal and vertical axes. For example, the cell immediately below [1.0, 1.0] is situated at [1.0, 0.0], and the cell to its right resides at [0.0, 1.0]. Conversely, when clamped bounds are used, any cell with an x or y coordinate that falls below 0.0 or exceeds 1.0 is considered inactive and set to a constant state of (0.0).

In addition, Utomata's grid can be divided into any number of self-contained horizontal and vertical tiles. In this case, each cell's x and y coordinates are determined relative to its normalised position within its containing tile.

***1** This section uses square braces to denote the [x, y] position of cells, and parentheses to denote their (r, g, b, a) state.*

State

Each individual cell in the system is assigned a numerical value, referred to as the cell's state. In Utomata, all state values are represented by a 4D vector where each component is normalised to be within the range of 0.0 to 1.0 (inclusive). State vectors are mapped to RGBA colour, such that any state is defined as (r, g, b, a). Values lower than 0.0 and higher than 1.0 are clamped to these limits. One-dimensional states can still be used in Utomata, yet these are defined and treated as vectors with identical components. Likewise, algorithms that feature discrete states are defined using fractional values. For instance, a binary state would feature values of either (0.0) or (1.0), and a 4-state CA may feature the states: (0.25), (0.5), (0.75), and (1.0).

Transition function

At regular intervals, all cells in the grid engage in a collective transition process. This transition function is expressed as a nested series of mathematical operations on 4D vectors. The result of this function is a new 4D vector that represents the updated state of each transitioning cell. During this transition, a given cell can access various input parameters, including its current state, the cumulative states of its neighbouring cells, and the individual state of any given cell. Utomata strictly operates in discrete time steps, such that every cell can only access the states of itself and other cells — as they were in the previous step. This precludes the possibility for old and new values to be mixed.[2] Additionally, Utomata does not inherently keep track of old state values, though long term memory can easily be emulated by using one of the vector channels. It is important to note that while Utomata's transition function enforces clamping on all returned state values, the transition function itself permits the used of any real-valued numerical constant.

***2** formally, the transition process can be represented as follows:*

$$V(t) = f(V(t-1))$$

Neighbourhood

A fundamental aspect of CA is the ability of a cell to "observe" the states of other cells, typically those adjacent to it, and incorporate these values into its transition function. Other state values are commonly referred to as a *neighbourhood.* These can be categorised as follows:

Null	The transition function includes no references to cell states.
Self	The transition includes only the state of the transitioning cell itself.
Totalistic (single)	The transition includes a single parameter whose value is the sum of specific neighbouring cell states, typically those in close proximity to the transitioning cell.
Totalistic (multiple)	The transition includes more than one parameter, whose value is the sum of specific neighbouring cell states.
Outer-Totalistic	The state of any individual cell in the grid may be taken into account during the transition.

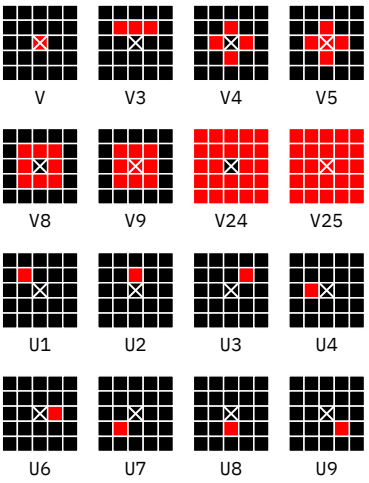


Figure 5.
Neighbourhood variables in Utomata.

Utomata is designed to accommodate all the above neighbourhood types, making it easy to implement commonly used neighbourhoods, thereby simplifying the creation of custom neighbourhoods, as well as allowing various mixed use cases. Figure 5 showcases built-in neighbourhood variables in Utomata, consisting of commonly used CA neighbourhoods. Note that there is no inherent difference between the state of the transitioning cell and any of its neighbours.

For custom and outer-totalistic neighbourhoods, Utomata offers convenient access to any cell in the grid through the `U(x, y)` operator. This operator returns the current state of any cell, based on relative discrete coordinates to the transitioning cell. These are referred to as deltaX and deltaY. The U function can be seamlessly integrated into any transition function either directly, or as a custom variable. This enables creation of outer-totalistic CA rules and the construction of custom neighbourhoods. For example:

```
vec4 N_vertical   = U(0,-2) + U(0,-1) + U(0,1) + U(0,2);
vec4 N_horizontal = U(-2,0) + U(-1,0) + U(1,0) + U(2,0);
vec4 N_crosshair  = N_vertical + N_horizontal + V;
```

The ability to combine different neighbourhoods and allow for custom ones is a key aspect of Utomata's broad expressive range. For instance, in the example on the right [3], a transition process involves taking the Moore Neighborhood, dividing it by the Von Neumann neighbourhood, and then multiplying the result by the state of the cell directly above the transitioning cell. This may seem unconventional, however, such expressive freedom aligns with Utomata's stated goal of empowering exploration, improvisation and experimental thinking in CA research.

***3** Example of a custom neighbourhood, which combines totalistic and outer-totalistic neighbourhoods:*

```
vec4 N = V8 / V4 * U(0,-1)
```

Operators

Utomata incorporates a set of custom operators, designed for use as nested functional expressions between vectors of different dimensions. This flexibility is meant to enhance the framework's expressive power, as well as to simplify and reduce potential errors in procedural generation of algorithms. Regardless of the dimensions of their input parameters, all Utomata operators consistently return a 4D vector as their output.

Binary Operators

`add(a, b)`	addition	`sub(a, b)`	Subtraction
`mlt(a, b)`	Multiplication	`div(a, b)`	Division
`pow(a, b)`	Power	`mod(a, b)`	Modulo
`min(a, b)`	Minimum value	`max(a, b)`	Maximum value
`lrg(a, b)`	Larger than	`sml(a, b)`	Smaller than
`eql(a, b)`	Equals	`not(a, b)`	Not equals
`dot(a, b)`	Dot product	`dst(a, b)`	Distance between

Unary Operators

`flr(a)`	Floor	`cil(a)`	Ceiling
`rnd(a)`	Round	`nrm(a)`	Normalise
`frc(a)`	Fractional part	`sgn(a)`	Sign
`sin(a)`	Sine	`asn(a)`	Arc sine
`cos(a)`	Cosine	`acs(a)`	Arc cosine
`tan(a)`	Tangent	`atn(a)`	Arc tangent

The particular way in which Utomata operators can mix and match parameters of different dimensions follows the same convention that is used to denote colour values in other creative coding libraries such as processing, P5js and Openframeworks. This is done using Utomata's `vec()` function, which accepts inputs of any dimension and returns a 4D vector.

Input	Output
`vec(x)`	`vec4(x, x, x, x)`
`vec(x, y)`	`vec4(x, x, x, y)`
`vec(r, g, b)`	`vec4(r, g, b, 1.0)`
`vec(r, g, b, a)`	`vec4(r, g, b, a)`

Following the above convention, Utomata contains multiple variations of each operator in order to ensure that a combination of any two values never yields an error. This presents an unconventional approach to per-component vector operations. It stems from the framework's philosophy, whereby intuitive improvisation and operator interchangeability are prioritised in order to empower open-ended exploration. Thus, all Utomata operators must first convert their input parameters using the `vec()` function before performing their own operation on a per-component basis. Consider the following examples:

Input	Output
`F(float x, float y)`	`F(vec(x), vec(y))`
`F(float x, vec2 v)`	`F(vec(x), vec(v))`
`F(float x, vec3 v)`	`F(vec(x), vec(v))`
`F(vec2 v, vec4 u)`	`F(vec(v), u)`

Boolean Logic

The use of conditional statements is discouraged in Utomata. Instead, boolean logic is implemented through custom operators. Specifically, the "equal", "smaller" and "larger than" operators function on a per-component basis, producing outputs of either 0.0 or 1.0 for each vector component. Furthermore, the "addition" and "multiplication" operators can be employed to combine these results, effectively serving as OR and AND operators, respectively.

Operator	Symbol	Description
`eql(a, b)`	`==`	return 1.0 if A equals B, otherwise 0.
`lrg(a, b)`	`>`	return 1.0 if A is larger than B.
`sml(a, b)`	`<`	return 1.0 if A is smaller than B.
`add(a, b)`	`||`	logical OR
`mlt(a, b)`	`&&`	logical AND

Configuration

A CA configuration function (config) determines the initial state of all cells in the grid before the transition function is first applied. As Many CA algorithms feature high sensitivity to initial conditions, the configuration often has a profound impact on the subsequent evolution of a system. The sole purpose of this function is to establish the initial states of all cells before the primary transition comes into play. In Utomata, the configuration is best regarded as a null-neighbourhood transition function. Utomata can accommodate a number of commonly used (as well as less common) configuration patterns via the following built-in functions and variables:

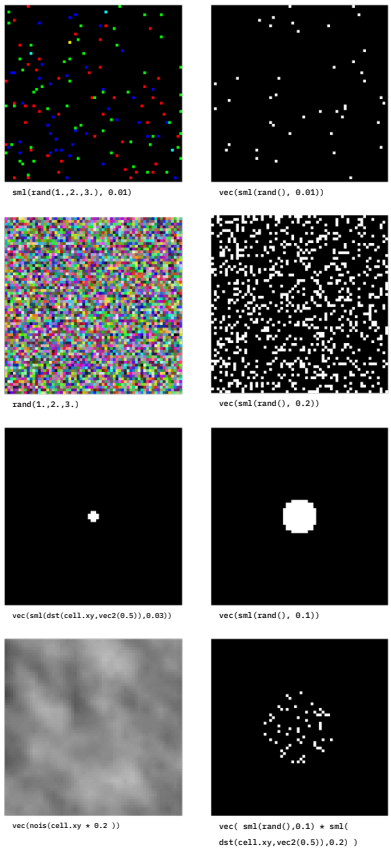


Figure 6.
Examples of useful configuration patterns in Utomata.

`cell.xy`	A 2D vector that holds the absolute normalised coordinates of the transitioning cell.
`crsr.xyz`	A 3D vector containing the absolute normalised coordinates of the cursor position. If the mouse is currently pressed, z equals 1.0; otherwise 0.
`grid.xy`	A 2D vector representing the number of columns and rows in the grid as integer values.
`time`	The current time step, expressed as an integer. This is particularly useful for implementing configurations or transitions that require more than one step to construct.
`rand(a, b, c, d)`	A function that returns a pseudorandom number, available in one, two, three, or four dimensions. The parameters act as seeds per colour channel.
`nois(x, y)`	A function for generating two-dimensional Perlin noise.
`set(x, y)`	A function that returns 1.0 if the transitioning cell is located at the absolute coordinates x and y. This can be multiplied by any vector to set it as the new state.

The configuration function can also utilise constant vectors or engage in compound operations with the above functions and variables. Importantly, these functions and variables can also be applied within the transition function to allow interaction, position dependent dynamics and non-deterministic CA algorithms which employ random values. Utomata.js also allows additional custom variables (uniforms) to be set via its JavaScript API. These features not only allow interactive configuration and transition functions, but also allow Utomata to be used in a wider range of creative applications and interactive software experiments beyond CA research.

Input

Utomata offers the capability of interconnecting multiple systems, allowing cells from one system to observe the current state of cells in another. This interaction is facilitated through input variables denoted as `I`. Variables `I` to `I25` return the local neighbourhood of a transitioning cell of the input system in the same way that `V` to `V25` do. Similarly, the operator `I(x, y)` can sample any cell in the input system in the same way that `U(x, y)` does.

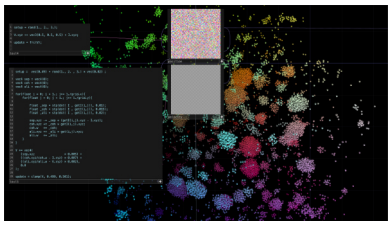


Figure 7.
Touchdesigner implementation of a swarm algorithm using two Utomata systems, each using the other as input.

This ability allows implementation of exceedingly intricate systems with any dimensionality, effectively creating a network of interconnected CA systems. Notably, an input system can either be static or operate at a different rate compared to its observer. Given that discrete time steps are employed, queries consistently reflect the current state of all interconnected grids in the chain. Furthermore, in such a network, systems are even not required to use the same grid size since cell coordinates are either normalised or relative. This capability potentially unlocks a range of novel possibilities which are not typically associated with CA. These may involve simple setups where two systems observe each other in a feedback loop to describe particle or swarm dynamics (as shown in Figure 7), or more intricate interconnected networks of CA grids, forming arbitrarily complex dynamics in higher dimensions.

3.4. Case Studies

The following section walks through the process of implementing three different CA algorithms in Utomata. These are: Conway's Game of Life (GOL), Wolfram's Elementary Automata (ECA) and Gray-Scott Reaction-Diffusion (RD). These particular algorithms have been selected primarily because they are among the most widely known and heavily researched CA algorithms to date, as discussed in their respective reviews in Chapter 2. This aims to benefit readers already familiar with implementations of these algorithms using a traditional programming approach.

An additional reason these particular algorithms were selected stems from their notable differences. GOL is a 2D binary state, totalistic CA which is often represented and implemented as a set of conditional statements. ECA is a 1D outer-totalistic CA, commonly represented as a rule table in which individual algorithms are encoded as an 8-bit binary value. RD is a 2D totalistic CA whose state values are continuous state vectors and is commonly expressed as an algebraic formula. The following case studies thus serve to demonstrate Utomata's unusual expressive range, as all three algorithms are implemented using the same programming syntax and settings.

3.4.1 Conway's Game of Life

Conway's Game of Life (GOL) is undeniably one of the most well-known CA algorithm, in large part due to its highly approachable rules and metaphor. State values in GOL are Binary, such that they can either be 0 or 1. The algorithm uses the totalistic Moore neighbourhood. As showcased in Section 2.2.2, GOL is commonly expressed through conditional statements that reference the binary state of the transitioning cell and its neighbourhood. Below is a reiteration of Conway's original rules for GOL [53]:

Survivals	Every counter with two or three neighbouring counters survives for the next generation.
Deaths	Each counter with four or more neighbours dies (is removed) from overpopulation. Every counter with one neighbour or none dies from isolation.
Births	Each empty cell adjacent to exactly three neighbours — no more, no fewer — is a birth cell. A counter is placed on it at the next move.

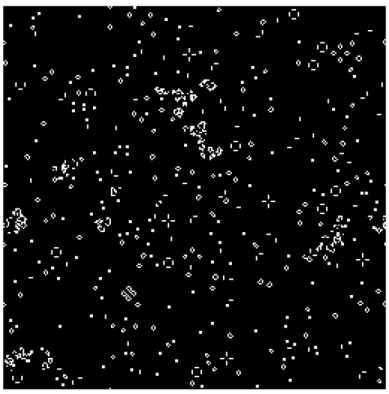


Figure 8.
Conway's Game of Life.

The above rule system can also be expressed by the following pseudocode:

```
For each cell:
if N == 2 OR N == 3  : V = 1
if N < 2  OR N < 3   : V = 0
if N == 3 AND V == 0 : V = 1
```

As mentioned in the previous section, since the use of conditional statements is discouraged in Utomata, boolean logic can instead be applied through the use of nested functional statements consisting of the built-in boolean operators (which return either 0.0 or 1.0). In this context, a functional statement refers to a nested series of mathematical operations, carried out on numerical values and variables. Notably, Utomata employs only unary and binary operators, which means that algorithms can effectively be represented as binary expression

trees. This results in overall more concise and minimalist algorithmic expressions. Below is the complete transition function of GOL in Utomata:

```
update = add(eql(V9, 3), eql(V8, 3))
```

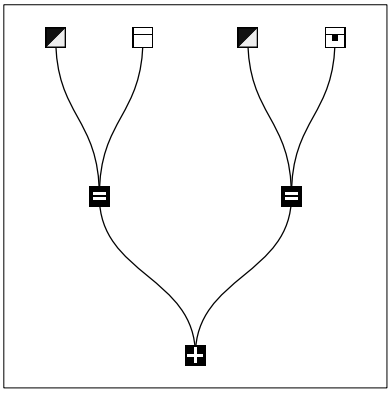


Figure 9.
Utomata GOL algorithm visualised as a binary expression tree.

First, note that this implementation uses a mixed neighbourhood by referring to two built-in neighbourhood variables: V9 and V8, which signify the Moore neighbourhood with — and without — the state of the transitioning cell. The `eql()` function acts as a boolean operator, returning the value `vec(1.0)` if its two parameters are equal and otherwise `vec(0.0)`. Moreover, as noted in the previous section, Utomata operators cast all values to 4D vectors so the use of the constant value 3 is actually interpreted by Utomata as `vec(3.0)`, or more accurately: `vec(3.0,3.0,3.0,3.0)`. Making sense of nested functional statements can be challenging. It is often useful to break them apart from the bottom-up as follows:

```
eql(V9, 3) // Expression A

eql(V8, 3) // Expression B

add(A, B) // OR operation
```

Expression A tests whether or not the inclusive Moore neighbourhood is equal to 3.0. This covers two scenarios: a “live” cell (with a state of 1.0) with two live neighbours, and a “dead” cell (with a state of 0.0) with three. According to GOL’s rules, if the equality holds true, the transitioning cell should be alive in the next time step. Expression B tests whether or not the non-inclusive Moore neighbourhood is equal to 3.0. This also covers two scenarios: a live cell with three live neighbours and a dead cell with three live neighbours. Again, both cases should result in a live cell in the next step. Note that the second scenario for expression B is actually already covered by expression A. Overall, these three scenarios cover exactly all cases that result in a live cell in the next step.

Since the `eql()` operator can only return either `vec(0.0)` or `vec(1.0)`, the sum of both expressions can have three possible outcomes: `vec(0.0)` if both expressions are false, `vec(1.0)` if only one is true, and `vec(2.0)` if both are true. Since all state values in Utomata are clamped to 1.0, the two latter scenarios simply return `vec(1.0)` and otherwise `vec(0.0)`, corresponding exactly to the cell’s desired state in the next time step.

Configuration

In order for any patterns to arise in GOL in the first place, an initial configuration must be applied, which consists of some mixture of ‘living’ and ‘dead’ cells. Otherwise the system will consist of only dead cells. The simplest way to create a non-uniform configuration is to use Utomata’s built-in `rand()` function, which returns a random number between zero and one for each cell in the system.

```
rand() // return a pseudorandom value between 0 and 1.0
```

However, since GOL is a binary state CA, it is up to the configuration function to round this value up or down.

```
rnd(rand()) // a random value rounded to either 0.0 or 1.0
```

While the above configuration is indeed valid, it generates an even distribution of roughly half living and half dead cells across the grid. This ratio can be

adjusted to offer more control over the distribution of living vs. dead cells by comparing the value of `rand()` to some static value. Because boolean operators in Utomata, such as `lrg()` and `eql()` return `vec(1.0)` for true and `vec(0.0)` for false, the even distribution of `rand()` will result in values larger than 0.9 in approximately 10% of cells. This particular distribution appears to provide an optimal configuration for persistent structures to emerge in GOL. Of course, this constant value can be further adjusted as need be.

```
lrg(rand(), 0.9) // return 1.0 if value is larger than 0.9,
                 // otherwise return 0.0
```

3.4.2 Elementary Cellular Automata

Elementary Cellular Automata (ECA) are a class of 1D, outer totalistic, binary state CA, originally devised by Stephan Wolfram [48]. ECA are most commonly represented as rule tables where each of the 256 possible algorithms is identified by an 8 digit binary number which corresponds to the possible configurations of a cell's neighbourhood. These binary numbers determine the cell's state in the next time step, based on its current state and the states of its two adjacent cells. Each cell has access to exactly three values: the state of the cell on its left, itself and the one on its right. Thus, there are exactly 8 ways to position these 3 values, which correspond to counting from 0 to 7 in binary:

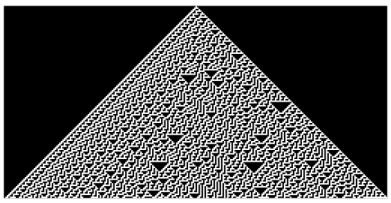


Figure 10.
Elementary Cellular Automata - Rule 30.

```
000 | 001 | 010 | 011 | 100 | 101 | 110 | 111
```

For each of these configurations, a single binary digit can be assigned to signify the new state of the transitioning cell in the next time step. Overall this results in a binary number with 8 digits that capture how a given cell may react to any possible (outer totalistic) neighbourhood configuration. Since there are exactly 256 possible values for an 8 digit binary number, this is also the number of unique ECA algorithms.

Alongside this numbering system Wolfram also provided a set of alternative notations of ECA [73], which include all 256 rules expressed as formulae. This notation happens to be highly suitable for implementing ECA as algebraic statements in Utomata as they feature only addition and multiplication of cell states. As noted, these expressions are outer totalistic, so each cell can access three neighbouring states: it own and the states of its two adjacent neighbours. In addition, these expressions make use of the inverse of each state. Which can be expressed as `(1.0 - V)`.

While it is possible to implement 1D CA in Utomata simply by creating a grid whose height is equal to 1, it is highly preferable to create a 2D grid that iterates one row at a time. This can be done by having each cell address the three adjacent cells above it instead of itself and its left and right neighbours. For this exact purpose, Utomata features the built-in variables: `U1`, `U2` and `U3` and `NU1`, `NU2`, and `NU3`. These provide convenient access points to expressing 1D outer-totalistic CA. Consider the following the following Utomata implementation of Rule 30 ECA: [4]

```
update = set(0.5, 0.0) +

vec((U1 * NU2 * NU3) + (NU1 * U2) + (NU1 * U3))
```

The above notation, though longer than an 8-digit binary number, provides two notable benefits over the more widely accepted use of rule tables. First, it is not an encoding but an explicit implementation that can be easily adjusted, explored or combined with other algorithms to produce different outputs

***4** This implementation uses infix rather than prefix notation in order to simplify and shorten the expression. However, implementing ternary operators such as those used here are nonetheless possible via a composition of two binary operators as follows:* `add(a, add(b, c))`

without altering the underlying framework. Second, this form of expression adheres to the same structure and rules as other CA algorithms in Utomata, thus enabling potential comparisons between algorithms that are commonly seen as too dissimilar to evaluate side by side.

3.4.3 Reaction Diffusion

As discussed in Chapter 2, reaction-diffusion (RD) is a highly influential algorithm, largely inspired by Alan Turing's seminal work on morphogenesis [23]. RD is sometimes regarded as a separate computational model from CA due to its use of a real-valued 2D state. Implementations of RD often require specialised software [133], as many CA frameworks only handle discrete states. However, Utomata's ability to support up to 4D real-valued vector states makes it well-suited for implementing RD and RD-like algorithms. Below is a complete RD implementation in Utomata:

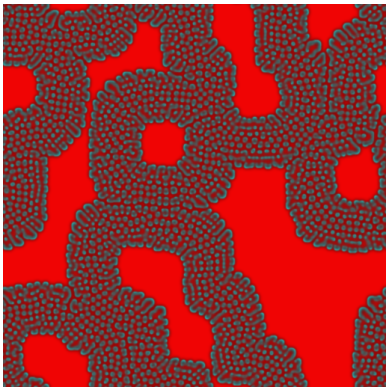


Figure 11.
Grey-Scott Reaction-Diffusion.

```
vec4 N = sub(V4, mlt(V, 4.0)); // weighted sum
float NR = mlt(0.22, N.r); // weighted diffusion for R
float NG = mlt(0.05, N.g); // weighted diffusion for G

float RGG = V.r*V.g*V.g;

float K = 0.062;
float F = 0.036;

update = add(
   frc(V),
   vec(
      add(sub(NR, RGG), mlt(F, sub(1.0, V.r))),
      sub(add(NG, RGG), mlt(V.g, add(F, K))),
       0.0,
       0.0
   )
);

update += set(crsr.xy) * crsr.z; // Cursor interaction
```

This implementation builds upon Karl Sims' online RD tutorial[5] [57] and incorporates advanced strategies in Utomata, such as custom neighbourhoods, kernel approximations, swizzling, vector operations, and real-time interaction. The algorithm simulates interactions between two chemical substances, represented by the red and green channels, denoted here as R and G. The simulation evolves by iteratively updating each cell in the grid with a compound vector that expresses the local reaction and diffusion dynamics.

Laplacian approximation

Custom variables N, NR, and NG calculate the Laplacian approximation, which measures how the value of a cell differs from the average value of its neighbours. This operator, commonly used in RD systems, models the diffusion of substances from areas of high to low concentration. Here, the Laplacian is approximated using a convolution kernel [6] with weighted contributions from adjacent neighbours (0.22) and diagonal neighbours (0.05).

5 *The tutorial presumably relates specifically to Grey-Scott Reaction-Diffusion, though this is not explicitly stated.*

6 *This setup reflects a Laplacian kernel optimized for the Gray-Scott reaction-diffusion model:*

$$\text{Kernel} = \begin{matrix} 0.05 & 0.22 & 0.05 \\ 0.22 & -4.0 & 0.22 \\ 0.05 & 0.22 & 0.05 \end{matrix}$$

These variables are computed as follows:

- `N` calculates the weighted sum of the Von Neumann neighbourhood by subtracting the weighted value of the current cell from its neighbours, effectively applying the kernel's central weight (-4).
- `NR` represents the weighted diffusion contribution for substance `R` from the neighbourhood.
- `NG` represents the weighted diffusion contribution for substance `G` from the diagonal neighbours.
- `RGG` drives the interaction between the two substances, where `R` is consumed to produce `G`. The quadratic dependence on `G` ensures non-linear behaviour, forming the foundation for emergent patterns.

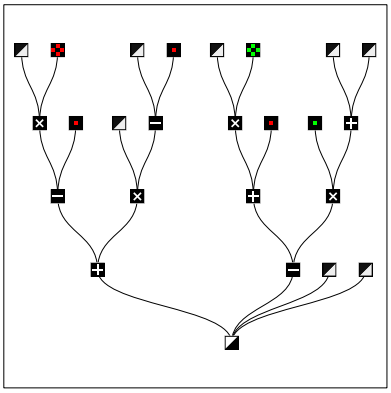


Figure 12.
Grey-Scott Reaction-Diffusion as binary expression tree. Note that this expression uses explicit values, effectively unraveling the custom variables.

Reaction and Diffusion

The term `RGG = R * G^2` drives the reaction where `R` is consumed to produce `G`. The quadratic dependence on `G` introduces nonlinear behaviour, enabling emergent patterns. The parameters `F` (feed rate) and `K` (kill rate) regulate these dynamics:

- `F` controls the replenishment of `R`. Higher values increase `R` availability, potentially overwhelming `G`, while lower values can starve the reaction.
- `K` controls the decay of `G`. Higher values lead to faster decay and less persistent patterns, while lower values support more stable formations.

Update Rule

The term `add(frc(V), vec(R', G', 0, 0))`, computes the transition from the current state of the cell to its new state.

- `frc(V)` caps state values between `0.0` and `1.0`, ensuring stability.
- `vec(R', G', 0, 0)` embeds the equations for `R` and `G` in the red and green channels. The blue and alpha channels remain unused and are set to `0`.

The formulas for `R'` and `G'` operate as follows:

- `R' = (NR - RGG) + (F* (1 - R))`: `R` decreases through reaction with G^2 and increases via diffusion and feeding.
- `G'= (NG + RGG) - (G * (F + K))`: `G` increases through reactions and diffusion but decreases due to decay, which combines the feed and kill rates.

Real-time Interaction

Real-time user interaction is enabled through a cursor-based perturbation mechanism added to the update rule. The function `set(crsr.xy)` returns `vec(1.0)` for the cell at the cursor's current position. This value is multiplied by `crsr.z`, which is `1.0` only when the mouse button is pressed and `0.0` otherwise. This allows users to dynamically inject substance concentrations into the system in order to trigger new patterns or disturb existing ones.

3.5. Summary

This chapter introduces *Utomata*: a novel computational framework for open-ended exploration and study of CA algorithms. Utomata aims to fill the methodological gap between general purpose graphics programming frameworks, which do not offer specific support for CA, and specialised frameworks, which may introduce significant constraints on open-ended exploration or creative applications.

Through its custom operators, variables and functional syntax, Utomata breaks away from traditional approaches to CA programming and allows implementation of an exceptionally wide range of CA. It aims to make CA research more accessible to creative practitioners who may not possess extensive backgrounds in computer or natural sciences. Practical use of Utomata is demonstrated through the implementation of three well-established CA algorithms. These case studies showcase Utomata's versatility and its ability to adapt traditional and distinctly different CA models into a single unified framework.

The next chapter demonstrates how Utomata can be used to conduct an exploratory study of a new family of CA algorithms, called Type-C. The chapter employs direct (low-level) manipulation of algorithmic expressions in Utomata, as well as introduces a number of tools and methods for interacting with CA algorithms. These practices facilitate the discovery of a number of novel algorithms and behaviours, which are presented and discussed. Chapter 5 introduces a high-level exploration method, based on Utomata's functional syntax. Along with its accompanying software implementation, this method allows real-time exploration and visual examination of large combinatorial spaces of CA algorithms.

4.Low-level Exploration

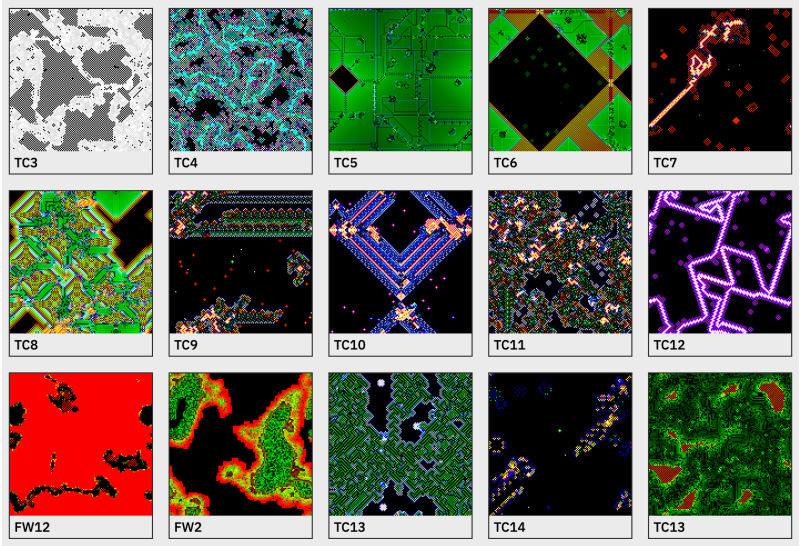


Figure 13.
Novel CA algorithms in Utomata Lib, discovered through low-level exploration.

This chapter details an exploratory study of novel CA algorithms using Utomata. Its findings, which include a number of novel algorithms and qualitative accounts of their behaviour, demonstrate the framework's effectiveness in open-ended exploration of CA. From a computational arts perspective, these studies also represent novel emergent structures which stand on their own merit as contributions offered by this thesis.

Exploration of CA can be understood through two interconnected properties: genotypic and phenotypic. The former pertains to a algorithm's structure — its "genomic" design — while the latter refers to the range of observable behaviours exhibited by its instances. These terms are further clarified in the following sections, and expanded upon in Chapters 5 and 6.

Low-level exploration of CA algorithms, in this context, refers to direct manipulation of their genotypic properties — their code — in order to induce changes to their phenotypic properties — their behaviour — in real time. By manually changing parameters, operators, or the structure of an algorithm, one can essentially navigate the high-dimensional space of its variations to reveal its immediate neighbours, or "sibling" algorithms. This approach is especially useful for uncovering close variants of a given algorithm, combining two different algorithms, or for isolating behaviours of interest through a process of visual examination, combined with vigorous trial and error.

4.1. Rationale

The primary goal of Utomata is to contribute to the diversification of known emergent structures in creative media, artistic experimentation and procedural content generation. As such, it is designed to favour synthesis over analysis and to emphasise a process of improvisation over goal directed research. In its pure functional form, Utomata's syntax does not easily lend itself to analytical comprehension, especially in longer algorithmic expressions. This chapter contends that this is not merely an acceptable trade-off, but rather a useful tool towards achieving the above goal.

The intricate mechanics of nonlinear dynamical systems, including CA, are notoriously hard to fully comprehend, even when using tools specifically designed towards analysis. A notable example is a neural network, where the weights of the underlying graph are not explicitly programmed, nor are their nuanced dynamics ever fully captured by the programmer. Moreover, In creative media it is often the case that once a behaviour is implemented, a tedious process of parameter tweaking starts. [1] This process is not always driven by analytical reasoning but rather by intuition, aesthetic preference and, most importantly, trial and error.

1 During a 2016 talk at Goldsmiths College, Memo Akten, a prominent creative technologist, referred to himself as a "professional number tweaker", stating that roughly 80% of his time developing software is spent on finding the "right" parameters for an algorithm.

While the notion of relinquishing any or all analytical comprehension of an algorithm, as proposed in this chapter, can be criticised for taking this approach too far, this comes with a noteworthy benefit. A non-analytical approach can empower programmers to apply their intuition onto more than just an algorithm's numerical parameters — but also to its logical operators and even its structure. By relinquishing efforts to understand every single element of the code, one is rewarded with creative freedom that would otherwise be difficult to obtain using a traditional programming approach. In this sense, an analytical understanding of CA transition functions may not only be unnecessary for some modes of exploration, but may sometimes hinder the creative process by strengthening the programmer's bias towards behaviours they are already familiar with or can more easily comprehend.

Of course, it is still possible to make sense of any Utomata algorithm. The GOL implementation described in the previous chapter is not only easy to understand but is arguably clearer and more concise, compared to traditional implementations. Finally, a non-analytical programming paradigm can effectively make exploration of emergent structures more accessible to creative practitioners who do not possess a background in computer or natural sciences. By shifting the focus away from established knowledge to acquired intuition, exploration of novel CA algorithms and dynamics could potentially consist of extremely simple workflows, which rely on minimal prerequisite knowledge. Consider the following Scheme:

1. Run a valid Utomata algorithm
2. Modify any aspect of it:
    - Parameter
    - Operator
    - Algorithmic structure
3. Visually examine the changes caused to the system
4. Undo or repeat

The underlying idea behind this scheme is that any change to an algorithm is akin to invoking a different one. A useful metaphor for this process is to consider these two algorithms as genetic siblings, where a functional algorithmic expression is thought of as the genotype and its resulting output is the phenotype. Minor genetic changes, such as tweaking a numerical constant, a neighbourhood type, or a single operator typically result in slightly varied phenotypic behaviour. Such variations often perserve a surpiring amount of phenotypic similarity. However, as more genotypic variations are compounded, phenotypic differences will typically stagger. In that sense, making pronounced changes to the structure of the algorithm can be thought of as invoking increasingly distant cousins, which feature widely different phenotypic characteristics.

4.2. A study of Type-C

One of most pivotal precursors for this research project is a software experiment conducted as part of the author's MA thesis project in Computational Arts [199]. The experiment featured a hardware accelerated implementation of the Abelian Sandpile algorithm [11] — a simulation in which virtual "grains of sand" are continuously dropped upon a lattice to form "piles" . Once a stack of four grains is formed in any cell, it would "collapse" and disperse among the cell's four adjacent neighbours. In turn, this may result in further collapses and ultimately form large scale avalanches and intricate structures.

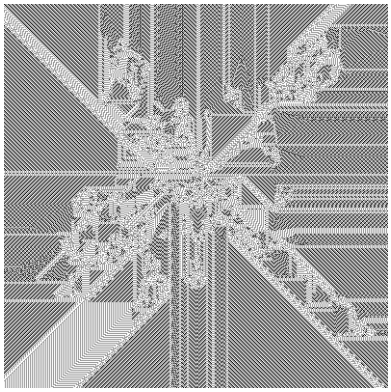


Figure 14.
Original Type-C algorithm.

This experiment had originally set out to implement the Sandpile algorithm as a continuous state CA, whereby instead of consisting of discrete entities (sand grains), continuous quantities would be added to the system in real time using the cursor. While this was initially made as a technical decision, allowing the use of openGL fragment shaders, the experiment ultimately introduced what would later become core attributes of Utomata: the use of normalised state vectors to describe CA dynamics, the use of real-time interaction as a means for CA exploration, and an explicit departure from physical metaphor.

Adapting the sandpile algorithm to use a continuous normalised state can be done by dividing all states values by four. According to the original algorithm, which consisted of a discrete four state CA, a grain would be a value of 1 and thus each cell's state may be either 0, 1, 2 or 3. [2] In a continuous normalised state implementation, grains would have a value of 0.25 and thus cells could have a value of 0.0, 0.25, 0.5 and 0.75. However, as this new algorithm now featured continuous state values, it immediately became apparent that these quantities can be arbitrarily changed to be any numerical value since they no longer represent discrete grains of sand. Similarly, the quantity which disperses between a cell's neighbours upon its collapse is also arbitrary. In fact, as physical modelling was not the goal, this quantity can even be different from the current value of the collapsing cell. In other words, the notion of conservation of matter, which is hard coded into the original Sandpile algorithm, could now be relinquished. The resulting new algorithm was named *Type-C* (shown in Figure 14). It featured the following two open variables, which could now be tweaked in real-time to induce a range of emergent behaviours and structures.

- **a**: The quantity inserted to the system by the mouse cursor.
- **b**: The quantity dispersed between a cell's four adjacent neighbours upon its collapse.

2 Once a value of 4 is reached, the cell "collapses" and its state returns to 0. Each adjacent neighbour is supplemented by 1, and so on.

This new variation was implemented as a fragment shader in the processing programming environment and was able to produce surprisingly complex structural patterns which appeared to emerge from extremely minimal input. This algorithm was later converted to use a totalistic neighbourhood, as well as significantly reduced to a single functional algebraic expression:

```
update = add(frc(V), mlt(stp(0.1, mod(V4, 0.6)), 0.6))
```

In a later experiment, a variation of this algorithm had yielded distinct meandering patterns that, under certain conditions, were able to produce second-order structures — distinct organisations of meandering patterns towards a self-similar, larger scale meandering pattern — shown in Figure 15. This variation was derived through a process of trial and error, whereby various changes to the algorithm were induced in an effort to expose novel features. Due to the experimental and intuitive nature of this process, by that point any analytical understanding by the author of the inner workings of the algorithm had significantly eroded.

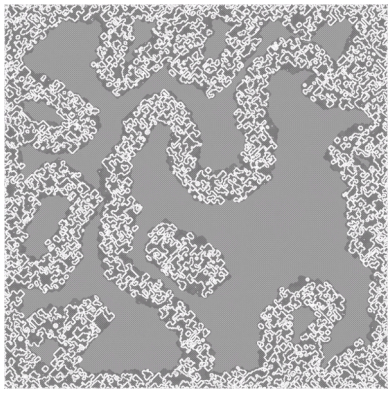


Figure 15.
Second-order meandering patterns in Type-C.

Subsequent attempts at simplifying this particular version in order to regain some level of analytical insight over its underlying process had rendered it sterile. Thus the only remaining course of action was to reduce it to a more concise form and attempt to isolate the self-similar patterns via a somewhat tedious process of elimination. This yielded an entirely new algorithm, albeit one in which the meandering patterns were indeed isolated.

Digger-Dagger

```
update = mlt(add(frc(V.r), mlt(sgn(stp(0.1, mod(V4.r,
vec(0.875)))), sub(add(vec(0.89), mlt(sub(V4.r, V4.g),
vec(9.0, 8.0, 10.0))), frc(V.r)))), vec(1.0, 0.99, 1.0))
```

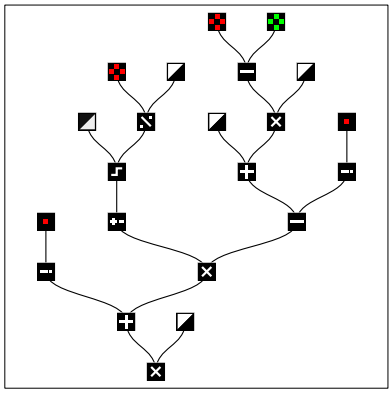


Figure 16.
Digger-Dagger algorithm visualised as a binary expression tree.

This new algorithm features a highly resilient worm-like pattern that, once invoked using the cursor, progressively consumes most available black regions of the grid. This algorithm is quite robust to parameter changes, with minor tweaks giving rise to “sibling” behaviours that present similar phenotypic traits. It is important to emphasise that the above expression was not obtained by gaining any analytical understanding of the algorithm’s behaviour, but through a process of reducing its logical statements to a single functional algebraic expression, combined with careful real-time tweaking of its numerical parameters. This approach takes advantage of the real-time capabilities of working with openGL fragment shaders, which require no compilation. Any change to the algorithm is immediately apparent and can thus be quickly built upon or undone. Likewise, any new algorithm can, once again, be used as a starting point for the discovery of even more algorithms.

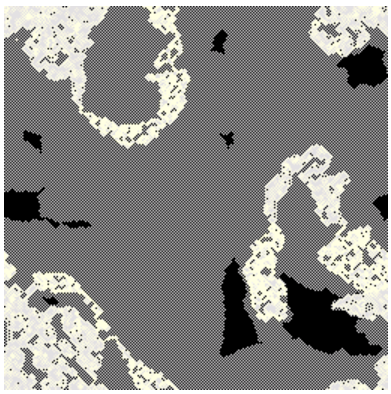


Figure 17.
Digger-Dagger.

4.3. Field Parameterisation

The non-analytical programming method detailed in the previous section describes a low-level approach to CA exploration, coupled with a computational framework that allows real-time programming and interactivity. While this methodology can indeed yield novel and unique findings, it is admittedly tedious. Most notably, while this low-level approach to exploration makes it easy to meander around the space of variations of any given algorithm, it can only account for just one algorithm at a time. To address this limitation, a higher-level tool can be applied to facilitate the concurrent observation of numerous parametric variations of a given CA algorithm. This tool can empower examination of a spectrum of algorithmic behaviours, continuous or discrete, allowing one to explore specific areas of interest. This can be done by integrating the x and y coordinates of each cell in the grid into the algorithmic expression, thereby allowing every cell to potentially execute a slightly distinct transition.

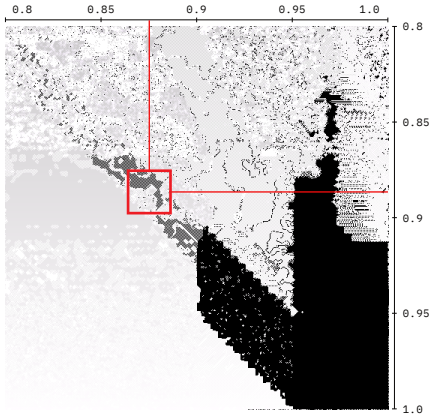


Figure 18.
Continuous field Parameterisation of Digger-Dagger. The highlighted region marks the selected parameters of 0.875 and 0.89.

This concept is referred to as field parameterisation. It was initially applied in the context of CA by Pearson [10], [82] to map a varying field of Reaction-Diffusion systems, wherein the constants `k` and `f` of the algorithm were varied along the x and y axes respectively. This method yields a two dimensional field of diverse phenotypic expressions of an algorithm, as each cell is computed using a unique pair of constants. This method has been incorporated into Utomata and generalised so that it can be applied to any algorithm. By exposing the cell’s position as a normalised 2D vector: `cell.xy`, an algorithm can be modulated to allow concurrent observation a wide range of behavioural variations. This not only makes it easier to locate behaviours of interest, but also allows for a more detailed investigation of specific areas by making it possible to “zoom in” to regions of particular interest. [3] This can be done by scaling the `cell.x` or `cell.y` variables, thus spanning larger or smaller regions of the parameter space. For example, the expression `(0.4 + cell.x * 0.2)` yields

3 *Additionally, the cursor position may be used, as well as any number of external inputs such as range sliders or sensors, potentially allowing higher level software development for more targeted explorations or applications.*

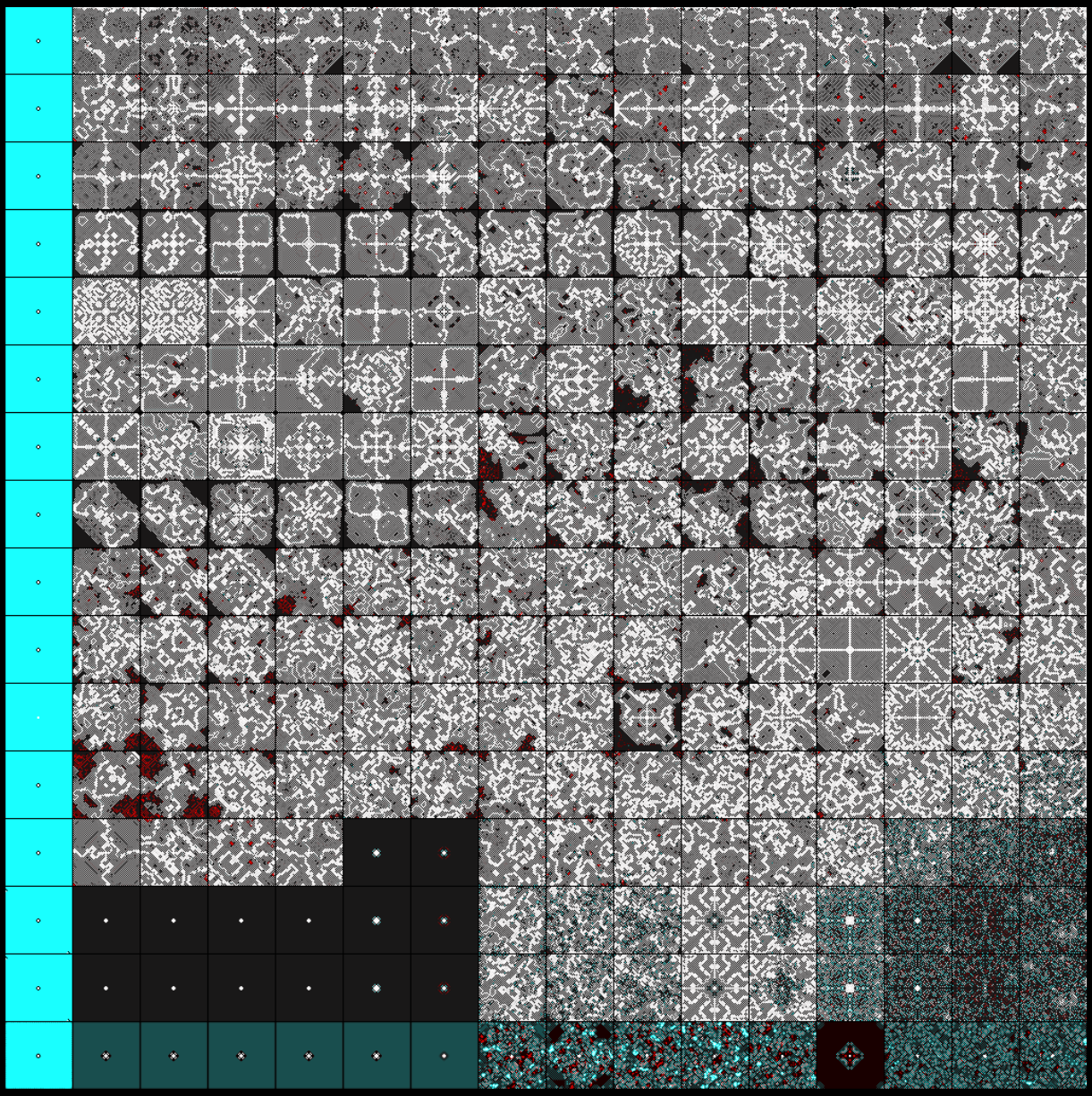


Figure 19.
Tiled field parameterisation of variations to Digger-Dagger.
Utomata supports a built-in tiling system that allows any number of rows and columns. Each tile is isolated to have its own toroidal bounds. A tiled field parameterisation addresses the problem of continuous field parameterisations, whereby the transition function differs between each and every cell, potentially resulting in inconsistent behaviour. Utomata'a built-in tile variable, which returns a 2D vector signifying the tile of the transitioning cell, can be used to map various parameters on a discrete field, allowing each tile to run a consistent and unique algorithm.

a 5X magnification towards the centre of a field. This generalisation of the field parameterisation technique can be leveraged to widen the scope of exploration of the multidimensional parameter space of any given algorithm in Utomata. By varying parameters across the grid. This higher-level view can provide a better understanding of the interplay between different parameters and their collective impact on the dynamics of an algorithm. In the next chapter, this approach is extended significantly, forming vast fields that span not only parameters, but also operators and structural variations of algorithms.

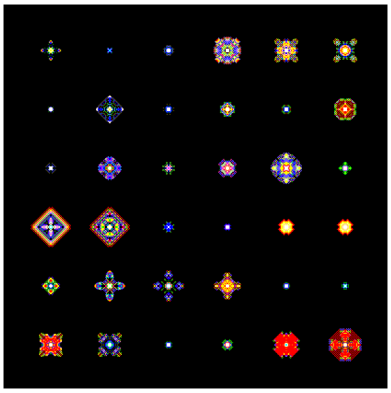


Figure 20.
Tiled field parameterisation of Type-C variants with a centralised configuration pattern, which yields radially symmetric and unique pattern in each tile.

4.4. Case Studies

This section presents a curated collection of novel algorithms, obtained through a process of low-level exploration and field parameterisation in Utomata. The genotypic differences between this particular set of algorithms are quite minimal, yet they present a surprising range of phenotypic patterns, while retaining a noticeable familial resemblance. As with the original Type-C algorithm, from which they all essentially stem from, all variants use a black — `vec(0.0)` — configuration and are typically invoked by setting small local regions of cells to `vec(1.0)` using a a configuration pattern or the cursor.

Red Nose Hexagliders

```
update = mlt(add(frc(V), mlt(sgn(stp(vec(0.1), mod(V4.rrrr,
vec(0.352,0.94,1.0)))), sub(add(vec(0.72,0.129,0.0),
mlt((sub(V4.rrgg, V4.ggrg)), vec(7.,2.0,1.0))), frc(V.
rrbr)))), vec(1.0,0.988,1.0))
```

Red Nose Hexagliders (RNX) is a continuous state, full RGB, Type-C variant which features the emergence of an unusually diverse range of stable, higher-order moving formations. These appear to be various compounds of a basal, rhombus shaped, glider-like pattern, spearheaded by a single red cell. Basal gliders span four by four cells diagonally, and traverse the grid along the x or y axes at a constant rate of one cell per time step. This traversal is a result of a two-step oscillation, visualised in Figure 21.

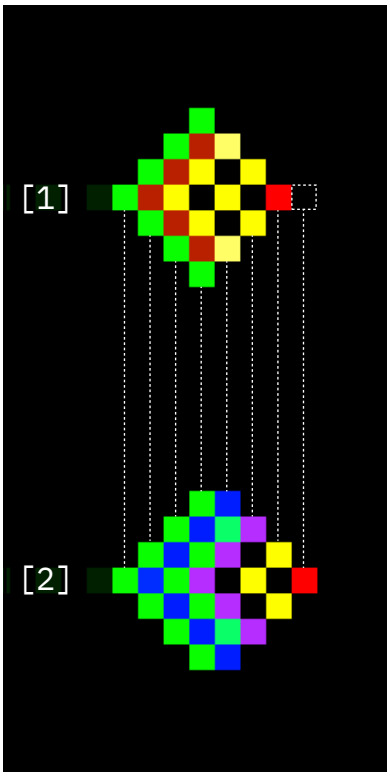


Figure 21.
Basal Glider pattern in RNX. The two-step oscillating pattern between [1] and [2] results in traversal of the grid along either of its axes at a constant rate of one cell per time step.

While relatively simple on their own, basal gliders appear to be resilient to overlapping with other gliders moving in the same direction. This resilience allows higher-order structures of a surprising variety to emerge and persist, as different compounds of gliders travel alongside each other. Figure 25 showcases a taxonomic study of observable formations in RNX.

Basal gliders can be triggered relatively easily by setting small, uneven groups of cells to random or white values. This typically results in the formation of stable globular structures (shown in Figure 25 [49, 50, 57]) which, if triggered again, can spawn one or more basal gliders. Once a glider is formed, it will retain consistent form and constant movement, unless interrupted by other patterns. In most cases, collisions between patterns traveling in opposite or perpendicular directions result in a termination of both. This often leaves a residual pattern, which may be static or feature simple oscillation. However, in some cases collisions may themselves spawn new gliders.

Many types of gliders and higher-order moving formations in RNX leave a unique residual linear pattern behind them as they traverse the grid. Residual patterns from basal and lower-order formations typically consist of simple, static and stable arrangements of cells (Figure 25 [62 - 67]), while higher-order formations may leave behind more compound oscillating patterns (Figure 25 [62 - 67]). The residual patterns of higher-order formations may themselves feature more complex activity, which are sometimes potent enough to continuously

spawn new gliders and glider formations (Figure 25 [39 - 48]) which are typically perpendicular to the direction of their spawning formation. Such active residual formations often result in exponential growth that rapidly fills up most dark regions of the grid within just a few minutes. However, Since the majority of residual patterns are static, the result of this exponential growth is an unavoidable end of activity due to a lack of space. This dynamic, where black regions constitute a depletable resource, is shared by a number of Type-C variants, including Digger-Dagger.

The algorithmic expression of RNX (visualised in Figure 24) includes references to the transitioning cell `(V)` and the Von Neumann neighbourhood `(V4)`. It also features five numerical constant vectors. The expression bares close genotypic resemblance to Digger-Dagger, as shown in Figure 22. This resemblance is quite surprising considering the two's highly distinct phenotypic dissimilarities.

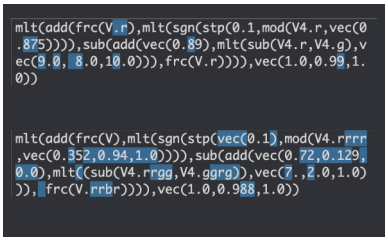


Figure 22.
Difference comparison between Digger-Dagger (top) and RNX (bottom).

Overall, RNX presents a notable and clear example of spontaneous emergence of higher-order structures in a continuous state CA. Figure 25 [1 - 48] showcases a collection of diverse formations of increasingly higher-order, observed and captured in RNX throughout this study. It is important to emphasise that this merely constitutes a preliminary study of this algorithm and the range of phenomena it can support. It is difficult to ascertain at this time exactly what other kinds of formations are possible in RNX. This is largely due to the resilient nature of glider patterns and their sensitivity to initial conditions.

Vital Signs (2021)

Following the discovery of RNX in 2020, a short video essay titled: "*Vital Signs - Red Nose Hexagliders",* was produced and released online [203], as part of the practice based portion of this thesis. The film consists entirely of screen recordings, captured during initial explorations of the algorithm. Narrated and scored by the author, the film adopts the stylistic and narrative framework of the nature documentary genre. This is a deliberate creative and didactic choice, aimed towards broadening the appeal of CA exploration towards a general audience. By taking on familiar cues from this popular genre, the film evokes a sense that the type of phenomena it features constitutes instances of real living creatures, yet without having to explicitly state this as fact. It presents a carefully crafted narrative around captured materials in order to make it accessible to its audience. This, too, in accordance with the nature documentary genre.

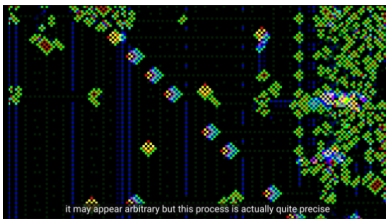


Figure 23.
Screenshot from: VITAL SIGNS - Red Nose Hexagliders.

Field Parameterisations of RNX

There are, in fact, numerous algorithmic variations of RNX. Minor changes to many of the numerical parameters in the algorithm appear to have little to no effect. However, a deeper investigation of parametric siblings reveals a wide range of highly potent dynamics. Figure 27 features four different examples of field parameterisations of a succinct version of RNX. In each one, the parameter configuration of the original algorithm is highlighted and the parameterisation is listed below. This emphasises a key concept regarding the methodology presented in this chapter, and in this thesis overall:

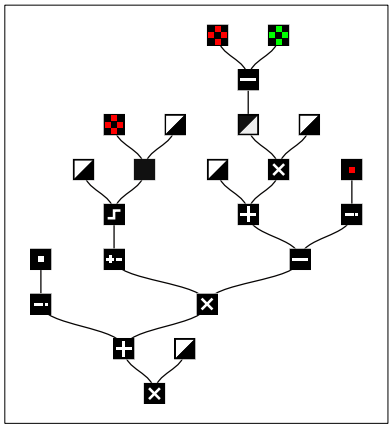
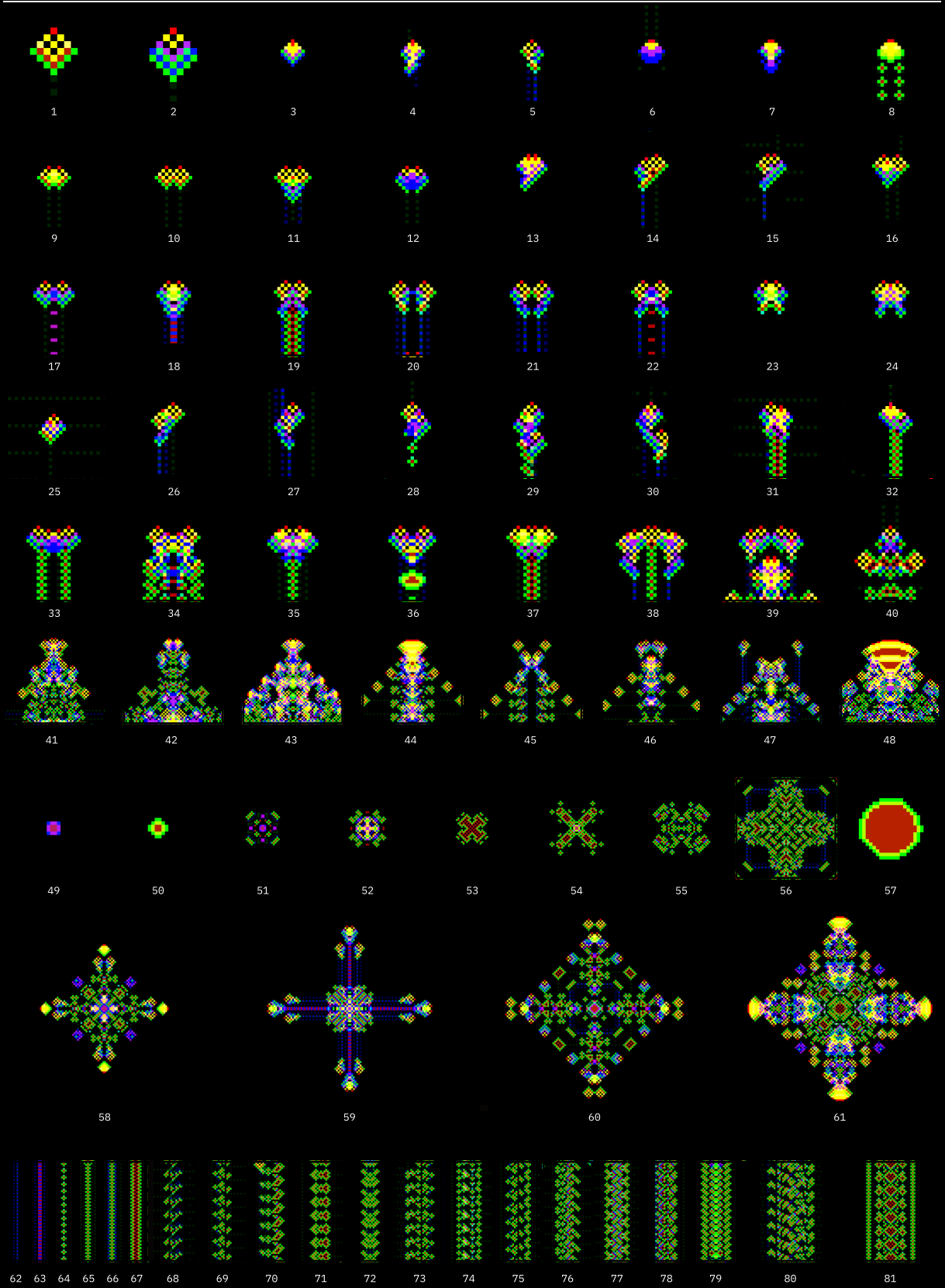


Figure 24.
Red Nose Hexagliders algorithm visualised as Binary expression tree.

Every conceivable CA algorithm can be seen as representing a given dynamical system that is embedded within a vast space of other — similar — systems. The ability to assemble and traverse this high dimensional space in order to explore such systems can benefit from the development of tools and methods that allow observation of such systems within the continuum in which they can be made to exist.

The fields plotted in figure 27 are akin to "zooming out" from RNX to reveal some of its parametric siblings. In each instance, exactly two parameters are plotted



[1 - 2] Basal glider states (2X magnification)
[3 - 5] Basal glider with appendage
[6 - 8] Wide nose variants
[9 - 12] Basal twin formations
[13 - 16] Asymmetric basal twin formations
[17 - 24] 2nd order symmetric twin formations
[25 - 32] 2nd order asymmetric twin formations
[33 - 37] Quadruplets with passive residual patterns
[38 - 40] Compound gliders with active residuals
[41 - 48] High-order formations with active residuals
[49 - 57] Sterile radial activation pattens
[58 - 61] Fertile radial activation patterns
[62 - 67] 1st order residual pattern formations
[68 - 81] Higher order residual pattern formations

Figure 25.
Taxonomy of observed structural formations in Red Nose Hexagliders.

along the x and y axes to reveal a unique set of surrounding behaviours. Due to the high dimensional nature of these expressions, there are numerous fields which can be plotted from a single algorithm, each plotting a range of combinations of two of its parameters at a time. While these examples constitute a continuous parameterisation, they nonetheless reveal distinct regions of coherent behaviour. This aligns with the previous observation about RNX's apparent robustness to minor parametric variations. This property is surprisingly common among other Type-C variants and continuous state CA overall. Phenotypic changes along a parametric field may often result from a "collapse" of a single floating point parameter towards a different value in the overall calculation of the expression. Less common though, is the fact that some algorithms, including RNX, appear to span multiple and separate regions of a field parameterisation. This may indicate that some types of phenomena, such as RNX basal gliders, are more resilient to parameter changes than others.

From a technical standpoint, field parameterisations in Utomata merely involve replacing one or more numerical parameters with an expression that contains `cell.x` or `cell.y` (or both). While these values can be scaled up or down to zoom in or out of a given behaviour, this approach can only be considered comprehensive for up to two parameters at a time. This is not the case for algorithmic expressions such as RNX, which feature at least 12 numerical parameters, as they represent a staggering number of different possible parametric combinations. Obtaining a field parameterisation that yields meaningful variants is thus more of an acquired skill. It involves trial and error, intuition and applying aesthetic judgement towards subject matter. This practice is inherently subjective and difficult to reproduce, for which it is firmly situated by this thesis within the domain of computational arts, rather than scientific research. Nonetheless, it is important to reiterate that a lack of precise analytical understanding of algorithmic behaviour does not appear to be a barrier for this form of exploration to yield meaningful novel findings.

TC8 (Wildfur)

```
update = mlt(add(frc(V), mlt(stp(vec(0.1,0.1,1.0), mod(V4.
rrrr, vec(0.352, 0.94,0.0))), sub(add(vec(0.72, 0.129, 1.),
mlt(sub(V4.rrgg, V4.ggrg),vec(10.0,0.1,1.0))), frc(V.
rrbr)))), vec(1.0,0.988,1.0))
```

This variant is a noteworthy sibling of RNX. It differs from it by only two numerical parameters while featuring a fundamentally different behaviour. Though both share a number of properties such as colour and overall pattern velocity, this algorithm features a much more volatile and continuous dynamical process. Invoking a single cell typically forms a gradient pattern that traverses the grid in four directions as a continuous rhombus shaped pattern. Slightly larger local invocation patterns can lead to the emergence of textured formations of unusual complexity. When invoked by a symmetric pattern and as long as it is uninterrupted, this algorithm appears to retain global symmetry indefinitely. This potentially means that the algorithm is resilient to floating point error, unlike many other continuous state CA. However, any subsequent perturbation rapidly compounds to introduce chaotic formations that take over the entire system.

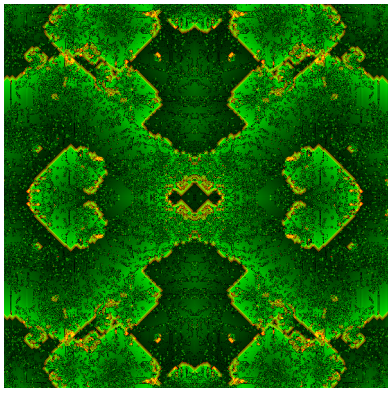


Figure 26.
TC8 (Wildfur).

A notable feature of TC8 is that, unlike RNX and Digger-Dagger, it appears to be capable of replenishing its non-black regions, allowing it to retain a potent dynamical process over a potentially indefinite number of steps. The yellow-orange waves that traverse the grid diagonally seem to be resilient to impact with other formations. A region that has been traversed by such a wave will become washed by a continuous bright green value that gradually diminishes,

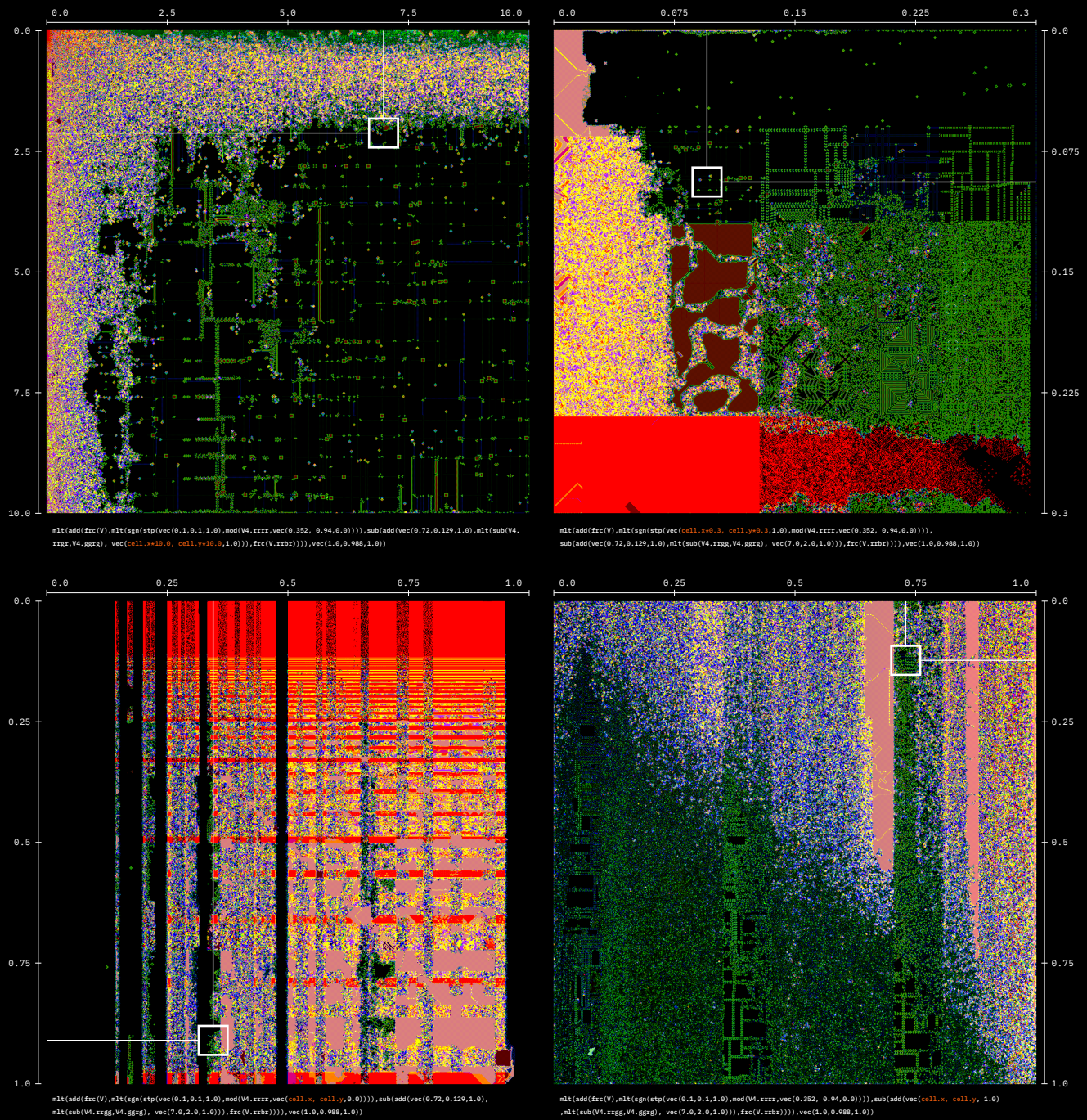


Figure 27.
Continuous field parameterisations of Red Nose Hexagliders.
The parameters of an algorithm can be explored in various ways. In these examples, four different numerical parameter pairs are selected and spread along the x an y axes using the cell.x and cell.y variables. Note that in some cases, the full normalised range can be scaled up or down, as well as appended with a constant value. This allows focusing on particular regions of interest. In many of these fields, the original RNX algorithm can be found in one or more region, among a continuous range of its sibling algorithms.

leaving a local arrangement of small, oscillating cell regions. These oscillators appear to “spontaneously combust” in sparse intervals, forming other wave patterns. The question of whether or not this process is indeed perpetual is, of course, undecidable. However this property makes it stand out among other Type-C variants, constituting a delicate and potentially perpetual equilibrium between volatile and stagnant dynamics.

TC5 (Infinitron)

This variant was discovered as part of an attempt to locate a middle ground between the discrete and robust dynamics of RNX, and the seemingly perpetual ability for regeneration apparent in TC8. While it is neither as diverse as the former, nor as regenerative as the latter, it demonstrates that particular behaviours of interest can indeed be cultivated through intuition based manipulation of algorithms. In this case, the ability to yield discrete formations, as well as replenish non-zero regions was located through a careful process of exploring field parameterisations, consisting of various combinations of its two sibling algorithms.

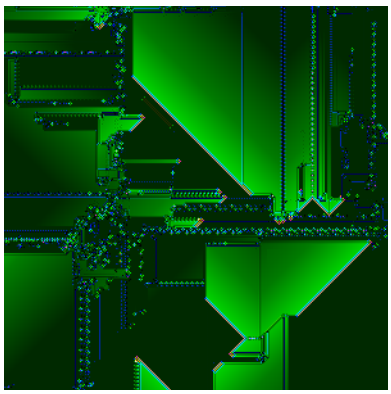


Figure 28.
TC5 (Infinitron).

RNX2

Another notable variation of RNX is a subsequent attempt to find an algorithm that supports the emergence of discrete formations, while still having the capacity for replenishing the non-zero regions of the space. This variant features distinct glider patterns that bare a familial resemblance to RNX basal gliders, while also presenting notable differences. Here, basal gliders consist of an orthogonal 3 by 3 formation. While it does support a number of glider types, they present much less diversity compared to RNX. The residual patterns left by gliders are also more uniform, with most patterns appearing as a smooth gradient green line. Perhaps most notably, the two step oscillation, which drives all RNX basal gliders is not present here, hinting that this oscillation may not be the prime instigator of motion in this lineage, but rather a by-product of it.

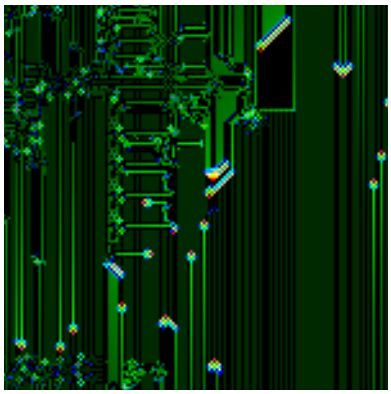


Figure 29.
RNX2.

FireWorm

The distinct worm-like patterns of Digger-dagger are visually reminiscent of a number of existing algorithms, including neural worms [194], a number of MNCA algorithms [192], as well as certain variants of RD. This apparent similarity is particularly curious considering none of these algorithms share any of their code. Subsequent attempts to isolate the worm-like formations featured in Digger-Dagger yield a highly robust and surprisingly succinct algorithm. This variation stems from an iterative process of combining field parameterisations with a gradual reduction and simplification of the algorithmic expression. With each iteration some part of the expression is removed, new variations are plotted across a 2D field and noteworthy variants are selected for their phenotypic traits.

```
update = vec(sub(div(V.g, V24.r),0.01), frc(V24.r), 0.0)
```

Unlike Digger-Dagger, this variant does not leave a residual pattern and thus appears capable of retaining activity indefinitely. Its single numerical parameter can be assigned values between 0.01 and 0.2, allowing some attenuation of the pattern’s scale and stability. These worm-like formations feature a 2 step oscillation in which the red edges alternate sides. They also demonstrate a remarkable robustness to collisions. In perpendicular impacts the open edge cam perform a full U-turn and continue uninterrupted. More shallow angle impacts can often disrupt the other side to form a new connection.

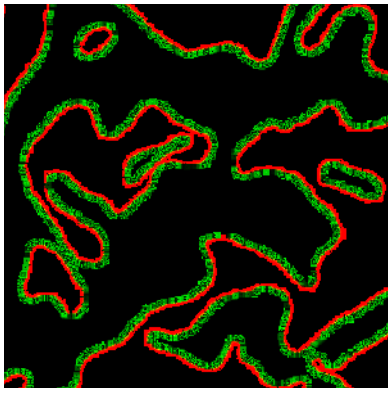


Figure 30.
FireWorm.

RNX4

Another RNX variant exhibits a two phase dynamic. In the initial phase, a stepped random configuration invokes a small population of simple glider patterns with short gradient residual formations that follow them but do not appear to subside. The second phase arises as a result of glider collisions which invoke localised rectangular regions on a 45 degree angle. These rectangular regions possess a feedback mechanism that causes them to grow exponentially in distinct pulses. As they rapidly fill the entire grid, they form a medium that acts as a perfect conduit for the pulsating patterns to persist in. The result is a highly dynamic pattern that constantly traverses through the stagnant, fractal-like formations of the now overlapping rectangular regions.

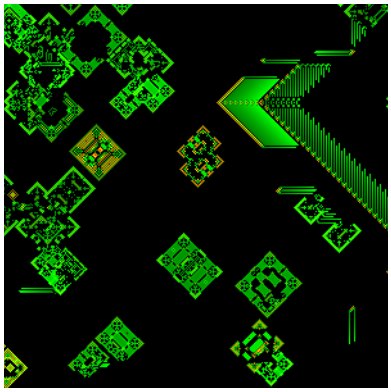


Figure 31.
RNX4.

TC15 (Forst)

This variant features unusual higher-order formations which present as two primary region types. The brighter one consists of high contrast, diverse arrangements of colours. When viewed up close, these regions appear highly chaotic, however, zooming out exposes a strict, repetitive structure which spans hundreds of cells across. This region type appears to emerge as a bi-product of an underlying, more rigid fractal formation in the form of a faint dark green, overall less diverse, and slightly faster process. This dynamic exhibits formations of varying scales and types but typically all are formed as symmetric trails that advance at a 45 degree angle.

The symbiotic relationship between the two region types enables a linear growth process which, like Digger-Dagger and RNX, eventually engulfs most available regions of the grid. However, the residual patterns appear to feature a more intricate oscillating dynamic compared to other Type-C variants, appearing as large scale invocations of blinking cells. It is unclear whether or not these invocations are a result of high order interactions that somehow manage to traverse the grid through the static residual patterns or a result of local cell interactions that statistically happen to induce blinking patterns in different regions at a time.

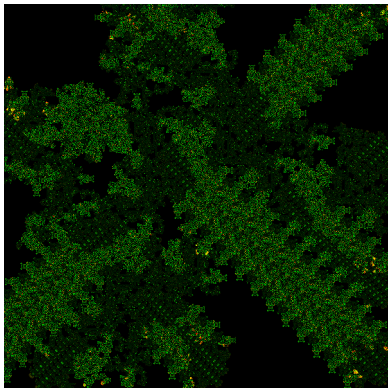


Figure 32.
TC15 (Forst).

TC12 (Neon Tubing)

This variant features resilient tube-like formations that traverse the grid in unusual angle increments of 22.5 degrees. The resulting diagonals are highly prone to produce oblong Sierpinski-like triangle patterns. The resilience of the tube formations allows them to easily “bounce” off of each other upon impact, thus often closing off large triangular regions across the grid. Though Sierpinski triangle fractal formations are apparent in many CA, they are typically very fragile, with any protuberance typically destroying further development. However, here the resilience of these patterns induces a “healing” dynamic in which they quickly regroup and move in alternate directions or even create tangent fractal formations.

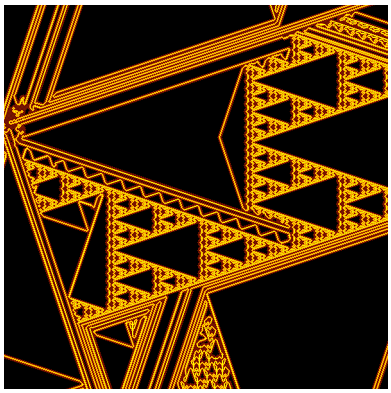


Figure 33.
TC12 (Neon Tubing).

TC9 (Citymakr)

This variant features a relatively diverse range of gliders and semi stable residual patterns. In this case, the residual patterns left behind by gliders appear to be an exact imprint of their values. This makes them appear as “borrows” of various sizes and compositions. This algorithmic expression is closely related to Digger-Dagger, as evident by the worm-like formations that form inside active regions. Since they are essentially enclosed, they cannot develop and retain a stagnant, yet potent form. However, when they do manage to break out of these regions, they can cause further invocations of a range of patterns.

4.5. Discussion

Low-level exploration of CA algorithms in Utomata essentially consists of applying a functional programming approach to CA transition functions. While this, in itself, is not necessarily a novel concept, this chapter demonstrates the effectiveness of this approach towards exploration of novel algorithms. The proposed methodology involves relinquishing an analytical understanding of CA dynamics, and instead relying on intuition, visual examination, and trial and error. It is important to emphasise that the case studies showcased in this chapter constitute only a single investigation of a handful of variants to a particular algorithm that, while showing initial promise, is merely one of countless others.

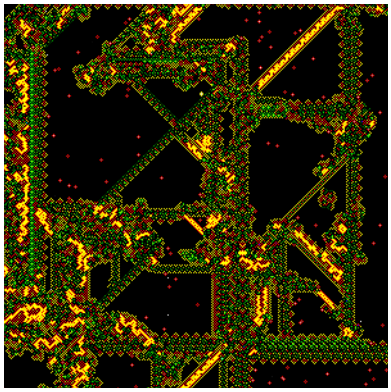


Figure 34.
TC9 (Citymakr).

The hidden potential of as-yet-undiscovered CA algorithms that present emergent properties is extremely difficult to assess. Regardless of its potential applications, this form of non-analytical programming is not without flaws. It is innately less suited for closed-ended tasks such as simulation, physical modelling or goal directed pattern generation. Moreover, the process of converting an algorithm from a general purpose language to a functional Utomata expression and vice versa, can require a moderate to high level of programming proficiency. Another notable drawback of this approach is the fact that it is unavoidably time consuming. Despite its real-time capabilities and tools for parallelisation, this kind of exploration in Utomata can only ever account for a single algorithm at a time. While clever use of the field parameterisation method is shown to offer a somewhat extended view of larger parametric spaces, effective as it may be, it simply cannot capture the vast landscape of algorithmic variations.

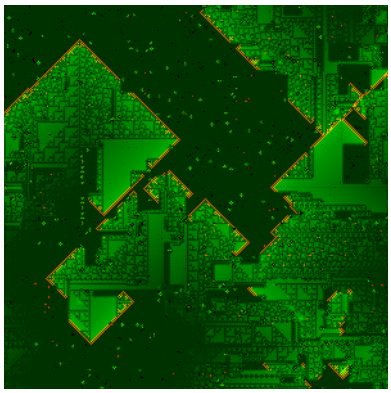


Figure 35.
TC8B.

Utomata expressions of a moderate length, such as the case studies presented here feature up to 20 numerical parameters. It is reasonable to assume that scattered across these 20 dimensional spaces are pockets of remarkable emergent behaviours, waiting to be discovered. Non-automated efforts to uncover, cultivate, isolate and study these particular behaviours of interest involve a tedious process that is perhaps more akin to farming than programming; a methodical craft that involves mutation, observation, selection and cross breeding.

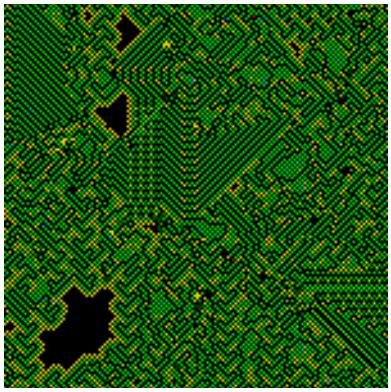


Figure 36.
TC13.

While undeniably tedious, this process can also be incredibly satisfying. It is an acquired skill that is more of an art than a science. As such, its appeal to a wider target audience may represent an untapped potential. An intuition-based method for CA exploration could offer more than just allowing artists to incorporate CA into their creative practice. Extending CA research's appeal to new audiences and contexts can potentially introduce entirely new paradigms to the study of CA, contributing to a mass diversification of known emergent phenomena. The next chapter introduces a higher-level approach to CA exploration. While notably more complex, this approach enables exploration of exponentially larger spaces of algorithmic variants and does not rely on any programming skills.

Both the low-level method presented in this chapter and the high-level method presented in the next chapter should best be regarded as different tools through which to observe and experiment with CA. These methods are not interchangeable, nor do they necessarily rely on each other. However, they may indeed complement one another in certain contexts. For example, a low-level study can be conducted initially to locate a particular algorithm of interest, which can then be investigated further using a high-level approach to uncover novel variations of it. A low-level approach can then, once again, be used to conduct more in-depth studies of these variants, and so on.

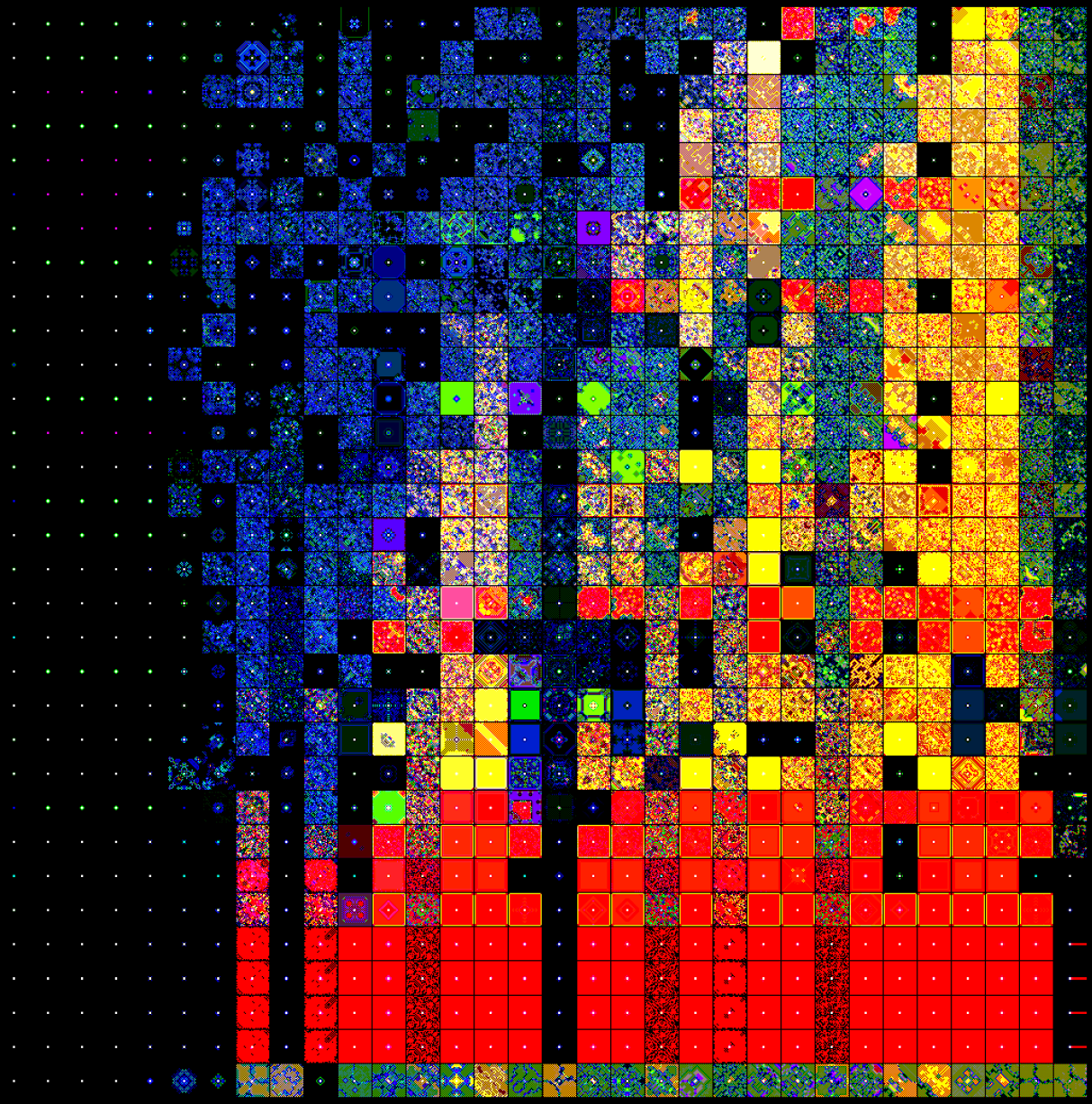


Figure 37.
Tiled field parameterisation of Type-C variants.
While this method can only account for two parameters at a time, mindful use of it can help expose the hidden range of behaviours that lies between them.

5.Spatial Mapping

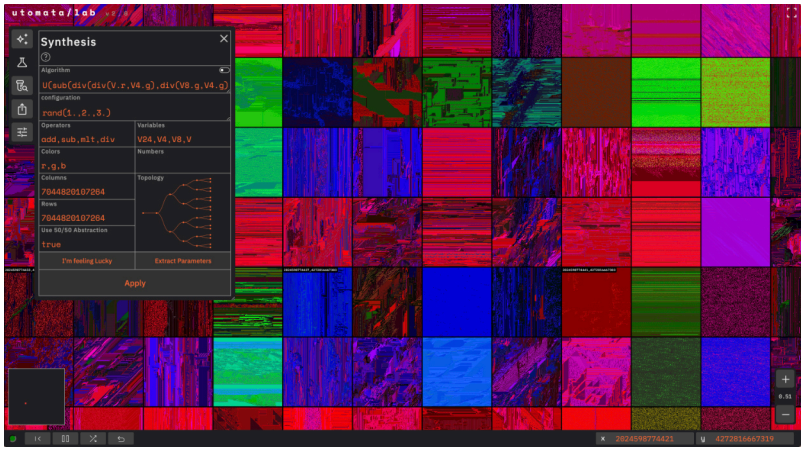


Figure 38.
Screenshot from Utomata Lab [204]

This chapter introduces a novel method for high-level exploration of CA algorithms. It involves procedural generation of nested Utomata expressions and a technique for mapping them onto a two-dimensional field. This effectively yields a map consisting of all possible variations of a given algorithm, allowing simultaneous interactive visual examination of an exceptionally large range of phenomena. What sets this method apart is its ability to produce a somewhat continuous space of phenotypic behaviour, whereby algorithms that share similar traits can be situated in geographic proximity to each other within the vast combinatorial space they occupy. There are, however, a number of notable limitations to this method, which are discussed throughout this chapter and in Chapter 8.

Spatial Mapping can be considered as a meta-programming approach to CA exploration. It enables one to literally scroll through a space of possible algorithms, as if browsing a vast online map. This has a potentially profound effect on the act of exploration; it can strengthen the notion that novel emergent structures are not created arbitrarily, but occupy their own unique domain, waiting to be discovered. In that sense, the method described in this chapter and its accompanying software implementation [204] constitute a literal embodiment of a “Laboratory of Babel”: they account for how a space of all possible CA algorithms of a given format may be constructed and explored.

Sections 5.1 introduces this concept and provides a step by step description on how this method is implemented. Section 5.2 offers a selection of case studies for utilising this method towards exploring combinatorial spaces around previously discussed algorithms. The next chapter demonstrates how this method can be used as a basis for a comprehensive study of a novel family of CA algorithms, called *Type-U*. Future expansions and implications of this method for the study of CA, as well as other types of virtual structures are outlined in Chapter 8.

5.1. Formal definition

The previous chapter demonstrated how low-level manipulation of algorithmic expressions in Utomata can be used to explore and study novel CA algorithms. It was suggested that this form of non-analytical programming is akin to navigating a combinatorial space of algorithms, such that any valid algorithmic expression constitutes a unique coordinate in that space. In that regard, any

modification, such as changing one of its numerical constants, is akin to traversing this high dimensional space, namely, along the dimension that corresponds to the parameter that was modified.

This way of thinking about CA algorithms evokes a sense that they are not just abstract entities that are generated from scratch in a computer program, but pointers to instances of unique dynamical systems. From a low-level coding perspective, this merely signifies a conceptual framework which aims to stimulate free-form thinking about open-ended exploration. However, this chapter goes beyond the conceptual by proposing a practical method for the construction of such spaces in software, allowing interactive exploration of all conceivable variations to any given CA algorithm.

This method involves several formidable challenges, such as uncovering the precise dimensionality and domain of these vast combinatorial spaces, as well as making sure they are consistent, complete and distinct. Furthermore, a mere enumeration of algorithms is insufficient; an organised and consistent framework for their placement is crucial for effective exploration. The proposed methodology includes the following steps:

1. Articulate an algorithmic expression and compile a set of symbols.
2. Define the domain and dimension of the algorithmic space.
3. Devise an isomorphic mapping onto every algorithm in the space.
4. Establish a method for spatially distributing algorithms.

A successful implementation of this method necessitates the development of a specialised software framework designed to facilitate not only the generation of large algorithmic spaces, but also seamless navigation of the various spaces it can make accessible. This software implementation, called *Utomata Lab* [204], is a key contribution made by this thesis. It acts as both a proof of concept for the method presented in this chapter, as well as a key instrument for the discovery and study of many of the algorithms documented in this thesis.

5.1.1 Algorithmic expression

The first step is to define and limit the scope of algorithmic expressions to be included in the combinatorial space in question. To illustrate the concepts of dimension and domain, it may be useful to first consider the following simple — yet flawed — approach for their definition:

- **Dimension:** Consider algorithmic expressions as sequences of characters up to a given maximum length. This length is defined as the dimension of the combinatorial space.
- **Domain:** The domain refers to a set of unique symbols available for constructing an expression. In this example, this refers to the total number of ASCII characters which may appear in a valid Utomata expression, which happens to be 41. [1]

For example, consider any Utomata expression consisting of up to 100 characters. This expression exists as a single point in a 100-dimensional space. This space contains precisely 41^100 unique expressions. While this method appears straightforward and is indeed capable of generating a space that contains all Utomata expressions of to 100 characters in length, this method is critically flawed. It describes a space that consists almost entirely of expressions that are decidedly NOT valid Utomata algorithms. Not only is the astronomical size of this space far too large to even suggest it can be explored in practice, it is so full of irrelevant combinations that it would be highly improbable to come

1 This includes letters a-z, digits 0-9, and the following characters: "(", ")", " ,", "." and " ". Note that while the whitespace character is not strictly a part of an algorithm, it allows inclusion of algorithms that are shorter than the maximal length.

across a single valid Utomata algorithm, let alone one of interest. Hence, it is evident that merely creating a comprehensive set of expressions is insufficient. A viable candidate set must also possess properties of completeness, exclusivity and distinctiveness. [2] In simpler terms, this space must posses the following properties:

1. **Completeness:** It must encompass a complete set of finite Utomata expressions.
2. **Exclusivity:** It must consist of only of valid Utomata expressions.
3. **Distinctiveness:** It must contain exactly one of each.

In order to construct such a specific and exclusive set, certain restrictions must be placed on the algorithmic expressions to be considered. The previous chapter demonstrated the advantages of utilising nested functional expressions for low-level manipulation of algorithms in Utomata. This format allows modification or substitution of any operator, variable or numerical constant, all while retaining the validity of the algorithm. Since Utomata exclusively uses unary and binary operators, algorithms adhering to this structure can be represented as binary expression trees. [3] This approach differs from the flawed approach of the previous example, which imposed no restrictions on symbol selection. A method for generating expressions, which relies on binary expression trees can indeed yield a complete, exclusive, and distinct set of algorithms. To this aim, a nested Utomata algorithm can be characterised by the following sets:

- **Topology (T):** A labelled binary/unary tree with a finite number of nodes, denoted as T.
- **Unary operators (U):** A set of operators used to label all nodes in the tree which have exactly one child. (two edges)
- **Binary operators (B):** A set of operators used to label nodes in the tree which have exactly two children. (three edges).
- **Variables (V):** Variables used within the algorithm (ie: V4, U9). These can only label leafs in the tree. (one edge)
- **Numerical Constants (N):** A collection of numerical constants used in the algorithm. These can also only label leafs in the tree.

The dimension and domain of the algorithm can thus be defined as follows:

- **Dimension:** For a given tree topology T, the dimension of the combinatorial space, encompassing all algorithmic expressions equivalent in topology to T, equals the number of nodes in T.
- **Domain:** Unlike in the earlier example, where all symbols were drawn from a single set of 41 characters, this combinatorial space features a *heterogeneous domain*. This arises from the varying sizes of the different sets containing variables, constants, unary operators and binary operators.

For example, consider the combinatorial space of the following expression:

```
add(frc(U2), 0.01)
```

This expression represents a valid Utomata algorithm, showcasing a combination of binary and unary operators, a variable, and a numerical constant. As such, it aligns with the structure of a binary expression tree, shown on the right, where each node falls into a distinct domain. This algorithm can be abstracted as follows:

```
B(U(V), N)
```

2 The requirement of distinctiveness only pertains to the genotype. There may be any number of algorithms that describe the exact same phenotypic behaviour. This property is discussed at length in the next chapter.

3 This discussion uses the term binary tree to include topologies that consist of both unary and binary operators.

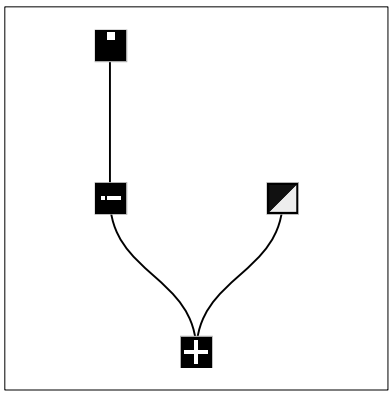


Figure 39.
Example of a simple binary/unary tree topology.

Any alteration of a symbol in this expression, that is, replacing it with a different symbol from its designated domain, yields an equally valid Utomata expression. The collection of all such substitutions defines a combinatorial space encompassing every conceivable permutation of expressions that adhere to the given topology T. It is important to emphasise that for this given topology, this set is both complete and exclusive: it encompasses all potential permutations of the algorithm, while containing nothing but valid algorithms. Crucially, this only holds true provided that all substitutions are drawn from their respective symbol domains.

The sheer magnitude of the combinatorial space described above should be considered with respect to the use of real-valued numerical constants. While this method can technically be applied to any finite set of symbols, the use of anything more than a handful of numerical constants would quickly stagger to form unreasonably large result spaces. For many algorithms, this would, in turn, result in flooding the space with overwhelming redundancy in the form of expressions that differ only by minute decimal fractions, producing large amounts of virtually indistinguishable outcomes. Therefore, the goal of achieving a complete space is hereby relaxed in order for a reasonably distinctive space to be formulated. This can be regarded as fine-tuning the resolution of the combinatorial space; a process that can be adjusted by adding or removing symbols from their respective domains. In this way, a dramatic reduction of the result space can be achieved, simply by restricting the domains to small subsets of symbols.

5.1.2 Enumeration

Continuing the example expression from the previous subsection, a finite combinatorial space of a manageable size can be constructed by defining the domain for each symbol set, as listed below. The number of unique algorithms in the space can then be calculated by multiplying the domain size for each symbol in the expression. Note that in this example, the set of topologies and the set of unary operators each consists of just one symbol. [4]

- `T: B(U(V), N )`
- `B: add, sub, mlt, div`
- `U: frc`
- `V: U1, U2, U3`
- `N: -0.02, -0.01, 0.0, 0.01, 0.02`

With the domains, dimension, and the total number of algorithms in the space now established, the next step involves defining a function capable of formulating every possible algorithmic expression. This is achieved by enumerating all potential configurations of available symbols. The number of possible expressions that conform to the above set can easily be calculated by multiplying all domain sizes together. However, enumerating all configurations in order to generate these algorithms is not as trivial.

Each unique configuration of symbols can be mapped to a series of numbers, which conform to a selection of the algorithm's symbols from their respective domains in order. For example, The series [1, 0, 0, 2] can be mapped to the expression `sub(frc(U2),0.0)`. Since each of these domains is of a different size, the enumeration must takes this into account. For example, the series [1, **1**, 0, 2] cannot be mapped to any configuration because the U domain contains only one symbol. This problem can be solved by using a *mixed radix system* for

***4** In this example there are 4 binary operators, 1 unary operator, 3 variables and 5 numerical constants. Thus amounting to exactly 60 permutations as follows:*

```
B( U(V), N)
4  1 3   5
4 * 1 * 3 * 5  = 60
```

enumeration. Unlike in many conventional number systems, where all digits conform to the same base, the digits of a mixed radix system can each be counted using a different base. This provides a consistent method to enumerate all possible expressions where each symbol is selected from a domain of a different size.

For example, in a decimal numbering system, all digits are in base-10. Incrementing from 0 involves advancing each digit up to 9, then resetting to 0 and advancing the next digit. Other base systems operate similarly, only with more or fewer digits available. For instance, base-2 systems can only use 0 or 1, while base-16 systems typically employ symbols from 0-9 and then A-F. In contrast, mixed radix systems employ a *heterogeneous base* so that each digit can count up to a value that may differ from other digits. Though less common and less intuitive, mixed radix systems are useful in specific contexts, such as time notation (hh:mm:ss and dd:mm:yyyy) and the imperial unit system.

As stated above, this mapping must be defined so that every valid input number is mapped to exactly one unique Utomata expression. This is known as an isomorphic mapping. Where it not isomorphic, the one-to-one relationship between input numbers and expressions would not be kept, resulting in the combinatorial space containing invalid expressions, missing expressions or duplicate expressions.

```
X = 0000
b = 4135
```

The input number for the mapping, denoted as **X**, can now be defined. The number of digits in X corresponds to the number of nodes in the algorithm's tree topology, which also conforms to the dimension of the combinatorial space. Additionally, a string of digits, denoted as **b**, is defined to indicate the base for each digit of X. These bases directly correspond to the domain sizes for each of the symbols in the expression in order. To establish a connection between an input number and a Utomata expression, each digit effectively acts as an index within its respective symbol domain. Consequently, when a digit assumes the value of 0, it selects the first symbol. Similarly, subsequent increments, in turn, access subsequent symbols.

```
0000: add(frc(U1),-0.02)  0001: add(frc(U1),-0.01)  0002: add(frc(U1),0.0)
0003: add(frc(U1),0.01)   0004: add(frc(U1),0.02)   0010: add(frc(U2),-0.02)
0011: add(frc(U2),-0.01)  0012: add(frc(U2),0.0)    0013: add(frc(U2),0.01)
0014: add(frc(U2),0.02)   0020: add(frc(U3),-0.02)  0021: add(frc(U3),-0.01)
0022: add(frc(U3),0.0)    0023: add(frc(U3),0.01)   0024: add(frc(U3),0.02)
1000: sub(frc(U1),-0.02)  1001: sub(frc(U1),-0.01)  1002: sub(frc(U1),0.0)
1003: sub(frc(U1),0.01)   1004: sub(frc(U1),0.02)   1010: sub(frc(U2),-0.02)
1011: sub(frc(U2),-0.01)  1012: sub(frc(U2),0.0)    1013: sub(frc(U2),0.01)
1014: sub(frc(U2),0.02)   1020: sub(frc(U3),-0.02)  1021: sub(frc(U3),-0.01)
1022: sub(frc(U3),0.0)    1023: sub(frc(U3),0.01)   1024: sub(frc(U3),0.02)
2000: mlt(frc(U1),-0.02)  2001: mlt(frc(U1),-0.01)  2002: mlt(frc(U1),0.0)
2003: mlt(frc(U1),0.01)   2004: mlt(frc(U1),0.02)   2010: mlt(frc(U2),-0.02)
2011: mlt(frc(U2),-0.01)  2012: mlt(frc(U2),0.0)    2013: mlt(frc(U2),0.01)
2014: mlt(frc(U2),0.02)   2020: mlt(frc(U3),-0.02)  2021: mlt(frc(U3),-0.01)
2022: mlt(frc(U3),0.0)    2023: mlt(frc(U3),0.01)   2024: mlt(frc(U3),0.02)
3000: div(frc(U1),-0.02)  3001: div(frc(U1),-0.01)  3002: div(frc(U1),0.0)
3003: div(frc(U1),0.01)   3004: div(frc(U1),0.02)   3010: div(frc(U2),-0.02)
3011: div(frc(U2),-0.01)  3012: div(frc(U2),0.0)    3013: div(frc(U2),0.01)
3014: div(frc(U2),0.02)   3020: div(frc(U3),-0.02)  3021: div(frc(U3),-0.01)
3022: div(frc(U3),0.0)    3023: div(frc(U3),0.01)   3024: div(frc(U3),0.02)
```

A comprehensive list of all variations of the example algorithm: ***add(frc(U2), 0.01)***

Upon reaching its maximum base value (which is its corresponding digit in **b**), every digit cycles back to 0, and the next digit in X is advanced, just as it would in a homogeneous base system. This iterative process ensures enumeration of all possible symbol combinations, as X is iteratively incremented — one digit at a time. Once X has reached a value of 3024, the process will have generated a set of algorithms that is guaranteed to consist of exactly one instance of every possible combination from the given set of symbols.

5.1.3 Spatial Arrangement

The previous subsection detailed the process of obtaining a complete set of permutations of a given algorithm. This process is based on a topology T and four distinct symbol sets: B, U, V and N. It relies on an isomorphic mapping that establishes a correlation between algorithms and a mixed radix system, denoted as X. The numerical assignment generates an ordered list of algorithms, ready for implementation in Utomata. However, as it may be common to explore relatively longer algorithms, as well as larger symbol sets, the number of possible permutations increases exponentially. Combinatorial spaces of this kind can easily encompass trillions of algorithms, making it impractical for a software implementation to run, or even attempt to store all of them.

It should also be emphasised that this method constitutes an isomorphic mapping from a high-dimensional space to a one-dimensional list. This step is crucial so that all permutations can be enumerated. However, extending this mapping to two (or possibly more) dimensions can provide significant benefits to exploration by allowing various spatial arrangements to be formed. Having some level of control in positioning algorithms along a two dimensional field is almost like applying a sorting mechanism that can significantly increase the continuity of the space by positioning similar algorithms in proximity to each other. If carefully applied, this may even result in some clustering of similar phenotypic behaviour, where algorithms of particular interest may end up situated in pockets of similar, perhaps equally interesting, variants.

If the mixed radix number X can be used to generate an ordered one-dimensional list of algorithms, then in order to generate a 2D space of algorithms, the same approach can be applied by creating a second mixed radix number, denoted as Y. By explicitly assigning each symbol in the expression to either X or Y, every algorithm would now be mapped to two numbers instead of one. This assignment can be denoted by an additional string **d**. The number of characters in d conform to the number of symbols in the expression. For a 2D distribution, each character in d would either be x or y. While there are 2^4 unique configurations of d, most of these configurations will result in equivalent spatial distributions. For example, “xxxx” and “yyyy” both result in the same distributions along a horizontal or vertical layout.

Note that this process can be extended to any number of dimensions,, up to the number of symbols in the expression, which, again, is the dimension of the combinatorial space itself. In practice, this process involves supplementing each symbol in the abstracted version of the algorithm with a corresponding indicator of its designated axis in the mapping as follows: [5]

```
Bx(Ux(Vy), Ny) // d = xxyy
```

Different 2D mappings can yield significantly different spatial arrangements. One of the primary goals of this method is to maximise the continuity of Spatial Mappings so that similar algorithmic expressions are situated in proximity to each other in their encompassing result space. The example above features a 2D mapping where all operators are assigned to the x axis and all variables and

5 *In the context of this study, only 2D mappings are considered. However, the prospects of using higher dimensions for spatial arrangements, as well as enumerating binary tree topologies are discussed in Chapter 8.*

numerical constants to the y axis. The resulting space might therefore feature a gradient of behaviours along each of its axes. This is because each two horizontal neighbouring algorithms would differ by exactly one operator, and each two vertical neighbours by a single parameter. This approach may be appropriate for certain algorithms, but not others. For example, some algorithms, such as the one presented in the next chapter, feature distinct axial symmetry. In that case, assigning axes in accordance to the unique structure of the algorithm is shown to be highly preferable. It should also be emphasised that while this method does achieve a high level of genotypic continuity, this does not necessarily ensure phenotypic continuity.

While a noticeable correlation between genotypic and phenotypic similarity has been observed for some of the algorithms discussed here, this is by no means assumed to be consistent or universal to CA in general. However, as demonstrated in the next chapter, it does seem apparent that larger combinatorial spaces feature greater phenotypic continuity than smaller ones. This is presumably due to the fact that the relative impact of some permutations is diminished in proportion to the length of the expression. The final step involves translating the mixed radix number systems, X and Y, to decimal coordinates. While not strictly necessary, this step allows for significantly more intuitive navigation of the result space.

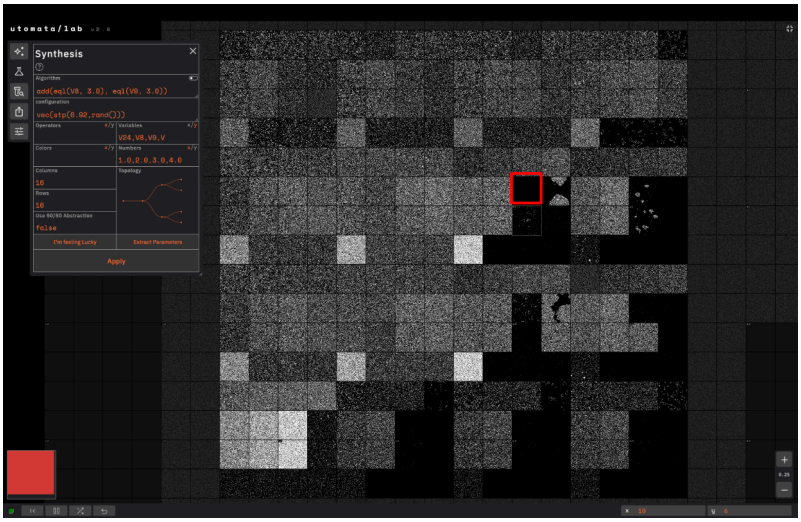


Figure 40.
Screenshot of Utomata Lab, featuring variations to GOL. (source algorithm is highlighted)

5.1.4 Implementation

Utomata Lab [204] is hereby presented as an integral part of the practice based portion of this research. It features a robust online implementation of the Spatial Mapping method described in this chapter, and has been used extensively for the study presented in the next chapter. This software implementation uses Utomata's built-in tiling feature, which allows up to 64 unique algorithms to run on a single HTML canvas node. This feature is supplemented with a bespoke rendering and layout engine, allowing it to run up to 16 HTML canvases simultaneously. Hence, running on sufficiently powerful hardware, Utomata Lab currently supports the simultaneous execution of 1024 unique algorithms.

Further, as users navigate through a space of algorithmic variations, new canvas instances are generated on the fly and hot-swapped to allow a seamless scrolling experience. Hidden Utomata instances are paused and their canvas object is replaced with a static png image of their updated state. Once an area is returned to, it automatically resumes running from its previous last step. The software can also be used to perform batch runs of any size in order to systematically cover larger regions of a given space. In this mode, each tile is set to run up to a

given number of steps, optionally performing a custom image analysis before halting to allow new tiles to be rendered. Moreover, It features the ability to export each individual tile, allowing entire regions to be graphed for subsequent offline study and further analysis.

5.2. Case studies

The example algorithm presented in the previous section was deliberately selected for clarity and simplicity, even though it does not yield particularly interesting behaviour. In this section, a series of case studies is presented, applying Spatial Mapping to the algorithms discussed in previous chapters. These case studies serve two primary objectives. Firstly, they provide practical demonstrations, showcasing how Spatial Mappings can be applied to existing algorithms. This serves to further clarify the process of symbol selection, spatial arrangement, and to demonstrate interactive exploration. Secondly, these case studies function as a proof of concept for the method itself by showcasing numerous variations of well-established algorithms.

5.2.1 Game of Life

As discussed in Section 2.2.2 and in Chapter 3, Conway's Game of Life stands as one of the most widely recognised CA algorithms, particularly beyond the realm of academic research. Its popularity has spurred the development of numerous derivative algorithms over the years [67], [69]. Many of these derivations have emerged from the creative endeavours of practitioners and enthusiasts who have conducted low-level explorations, employing methods akin to those described in Chapter 4.

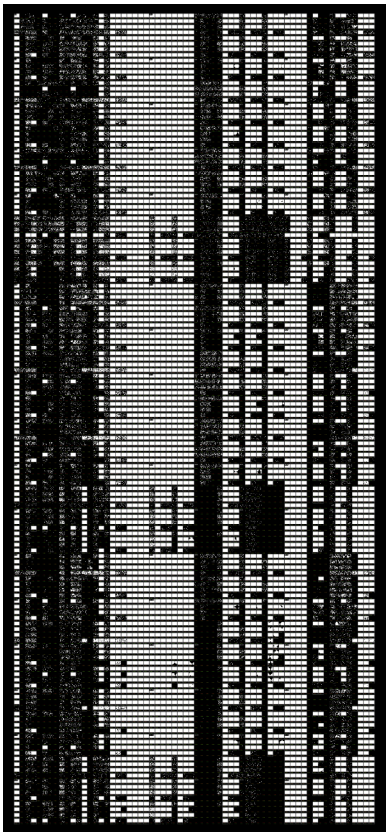


Figure 41.
GOL variations in Utomata Lab.

Efforts to discover novel variants of GOL have also included the creation of the "Life-Like" set of algorithms [71], which employs a specialised notation to describe algorithmic behaviour. In the Life-Like notation, the fundamental concept of GOL is hard-wired. Thus, while there are numerous possible Life-Like variants, they only differ in the number of live neighbours it takes for a birth, death or survival to occur. In contrast, Utomata offers a much more flexible approach to GOL variant exploration. Its use of functional expressions can transcend beyond the above restrictions to also include algorithms that, explicitly or implicitly, do not adhere to this life-death metaphor.

Through the use of Utomata Lab, the act of obtaining novel variations to GOL has been surprisingly straightforward. Recall the following implementation of GOL from Chapter 3. This expression consists of three binary operators (add, eql, eql), two variables (V8, V9) and two numerical constants (3.0, 3.0). A Spatial Mapping of GOL can be generated by the method described in the previous section. This involves defining a set of symbols [B, V, N], and assigning x and y coordinates to each symbol in the expression. Consider the following mapping parameters:

```
Source:      add(eql(V8, 3.0), eql(V9, 3.0))
Abstraction: Bx (Bx (Vy, Ny ), Bx (Vy, Ny ))
```

```
B: [add, mlt, eql, lrg, sml]
V: [V, V4, V8, V9, V24, V25]
N: [0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0]
```

These mapping parameters result in the creation of a relatively large combinatorial space of variations to GOL. This particular axial distribution assigns all three binary operators (B) to the x axis and all variables and numbers

to the y axis. The set of binary operators include the `add` and `eql` operators, which are present in the source expression, as well as three additional operators: `mlt`, `lrg` and `sml`. These were selected for their compatibility with boolean operations in Utomata, thereby ensuring all transition functions still return integer values so that the result space would, in turn, feature only binary state algorithms. The set of variables (V) includes five totalistic neighbourhood variables, as well as the value of the transitioning cell. Numbers (N) include whole numbers 0.0 through 9.0. [6]

The resulting space consists of 450,000 unique variations, spanning 125 columns and 3600 rows. In this particular configuration, the source algorithm is positioned at (x:12, y:2023). While the order of the symbols in their respective domain sets does not change the composition of algorithms in the space, it can significantly impact the order in which they appear. For instance, switching the `mlt` and `eql` operators would position the source algorithm at (x:6, y:2023) instead. Along with adjusting the axial distribution, adjusting the order of symbols in their symbol domain sets can be a useful strategy for organising and clustering certain phenotypic behaviours in a result space.

This particular mapping is advantageous because it ensures that every two neighbouring cells differ from each other by only one symbol. Moreover, it makes navigating the space slightly more intuitive, as travelling along the x axis means changing operators and travelling along the y axis means changing parameters and variables. This is evident in space's tendency to feature similar behaviours along its columns, more so than along its rows. There are, of course numerous other ways to create a combinatorial space of GOL variants. Different Spatial Mappings and distributions may offer more or less continuous spaces, which feature more or less interesting variations.

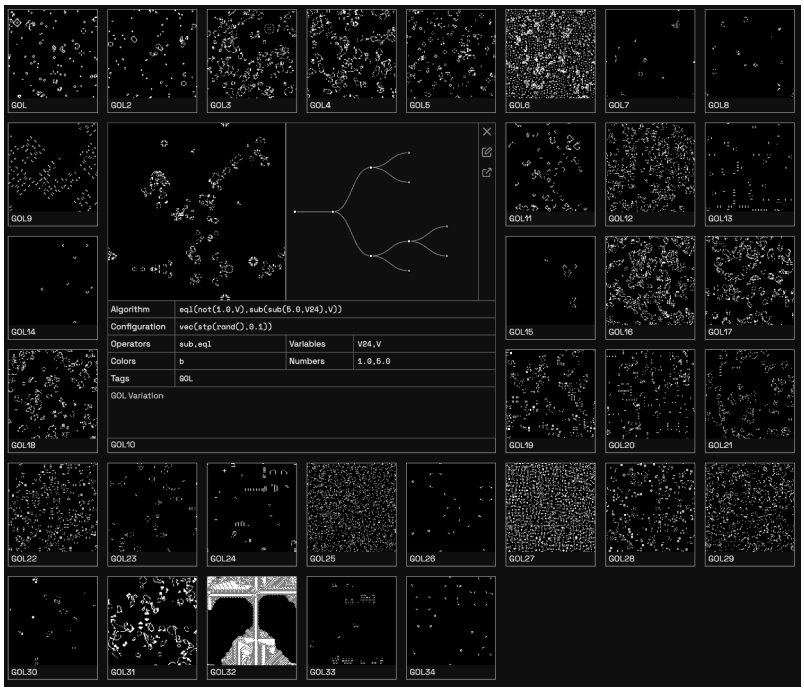


Figure 42.
Variations to GOL featured in Utomata Lib. Most of these algorithm were discovered by browsing through Spatial Mappings of GOL.

For example, a different axial distribution may involve plotting all operators and variables on the x axis and all numerical constants on the y axis. This approach yields a space with 4500 columns and 100 rows. The immediate surroundings of the source expression, now situated at (x:280, y:33), consists of numerous other algorithms which feature the use of the same numerical values [3.0, 3.0]. This is because one would have to travel a significant distance along the y axis before

6 *Of course, these choices can be easily adjusted later on. For example, by adding the div and mod operators to the domain, some algorithms are likely to return fractions, resulting in the incorporation of continuous state CA in the result space.*

reaching variants that involve other constants. This may be beneficial for certain types of algorithms, as it may help to uncover particular variants of interest.

The presence of perfect duplicate phenotypic behaviour within a combinatorial space appears to be quite common. In this case, this can occur by flipping between the V8 and V9 variables. Two algorithms which feature identical behaviour — but not an identical algorithm — can be referred to as *phenotypic twins*. In this case, a number of such twins may exist in the space due the fact that both the `eql` and `add` operators are symmetric. Some level of phenotypic redundancy in Spatial Mappings is often unavoidable. This topic is elaborated upon in the next case study, as well as further discussed in the next chapter.

Though it can appear to make exploration somewhat harder by unnecessarily increasing the size of the combinatorial space, some level of redundancy also entails a noteworthy benefit. Since there are many different ways to articulate the behaviour of GOL in Utomata, there are many different combinatorial spaces that can be created that encompass it.

```
update = add(add(mlt(eql(V8, 2.0),eql(V, 1.0)),mlt(eql(V8,
3.0),eql(V, 1.0))),mlt( eql(V8, 3.0), eql(V, 0.0)))
```

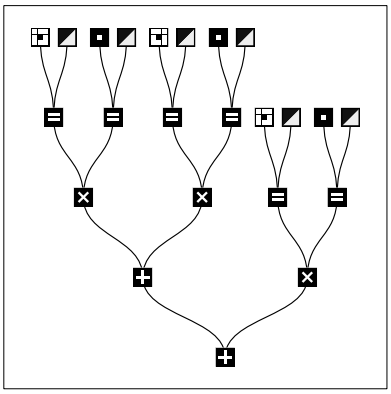


Figure 43.
Visualisation of extended GOL algorithm.

The above expression (also visualised in Figure 43) is an implementation of GOL which features a more explicit translation of its original rules. While it is noticeably less elegant, compared to the previous example, it presents an opportunity to explore a much larger combinatorial space of GOL variants due its sizeable length. Using the same symbol domain sets as before, a Spatial Mapping of this algorithm would consists of 48,828,125 columns and 46,656,000,000 rows. The source algorithm, residing at (x:882056, x:25551473630) is now surrounded by a surprising number of phenotypic twins. This aligns with the previously mentioned hypothesis that larger combinatorial spaces may inherently feature greater phenotypic continuity because the impact of any single permutation in an expression is inverse to its length.

```
Source: add(eql(V8, 3.0), eql(V9, 3.0))
Abstraction: add(eql(Vx, Ny), eql(Vx, Ny))
```

```
V: [V, V4, V8, V9, V24]
N: [0.0, 1.0, 2.0, 3.0, 4.0]
```

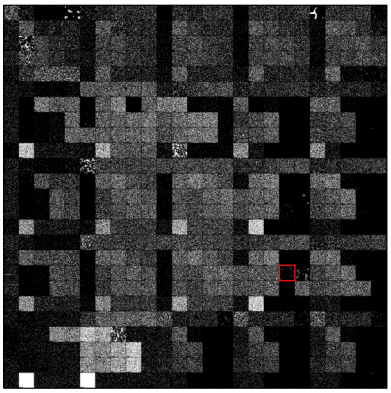


Figure 44.
A restrained 25x25 combinatorial space of GOL. (source algorithm is highlighted)

Similarly, the size of a space can be significantly decreased by either using smaller symbol sets, or only selecting certain symbols to permeate. With the above mapping parameters, all algorithms in the space retain the add and eql operators in place. The axial distribution alternates 5 variables along the x axis and 5 numbers along the y axis. This creates a highly explorable space consisting of 25 columns and 25 rows, shown in its entirety in figure 44. The source algorithm is positioned at (x:18, y:7) and it appears to have exactly one phenotypic twin at (x:18, y:11). Though significantly smaller than in previous examples, this space still features at least 8 other variants that, just like GOL, have been observed to enable the emergence of persistent higher order structures. Figure 42 showcases a number of GOL variants, discovered through various Spatial Mappings.

5.2.2 Elementary Automata

Elementary Cellular Automata (ECA) have been discussed and implemented in previous chapters. The methodical approach applied by Wolfram to define their combinatorial spaces makes it a noteworthy precursor for this research project.

However, due to their innate properties, the combinatorial space of ECA is comparatively very small, consisting of exactly 256 unique algorithms. A Spatial Mapping of ECA can use the formulaic expression devised by Wolfram [73], which was shown to be highly suitable for Utomata in Chapter 4. This approach yields expressions that can easily be adapted for Spatial Mapping as follows:

```
Source:
set(0.5, 0.0) +
vec((U1*NU2*NU3)+(NU1*U2*1.0)+(NU1*U3*1.0))

V: [0.0, 1.0, U1, U2, U3, NU1, NU2, NU3]
```

The above set results in a space that spans 32,768 columns and 4,096 rows. The source algorithm, rule 30 ,is situated at (x: 22294, y:697). Non-exhaustive excursions of this result space indicate it does contain any novel phenotypic behaviour beyond the known 256 variants. This is due to the overwhelming amount of genotypic redundancy, which is partly attributed to operator symmetry. For this particular algorithmic structure, which relies on just addition and multiplication, the number of mathematically identical transition functions is indeed staggering.

However, a crucial component of redundancy stems from the inclusion of the numeral constants 0.0 and 1.0 in the variables domain. This mixed use is necessary for expressing the variability of the source algorithms, which feature multiplications of zero, one, two or three symbols at a time. While the inclusion of numerical constants in the variables domain is technically possible in Utomata Lab, it should be used with caution. In this case, 0.0 and 1.0 serve as null and identity operators respectively. For instance, according to Wolfram's formula, Rule-128 is denoted simply as `vec(U2*U3)`. Therefore the only way for it to be included in the Spatial Mapping is through an expression such as the one below, of which there are numerous variations which are mathematically equivalent.

```
(U2*U3*1.0) + (0.0*0.0*0.0) + (0.0*0.0*0.0)
```

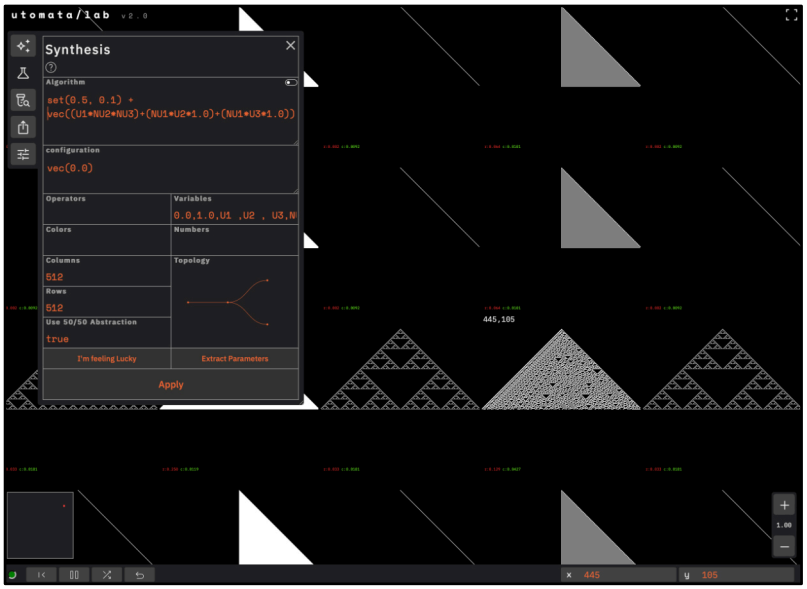


Figure 45.
Algorithmic variations of ECA in Utomata Lab.

5.2.3 Reaction-Diffusion

The effectiveness of any Spatial Mapping depends, in large part, on the structure and composition of the base algorithm in question. Longer algorithms which contain more and deeper nested symbols potentially result in gargantuan combinatorial spaces, which are generally far too vast to explore manually. Notably, this is not to say that they are too large for exhaustive searches — which is the case for almost all Spatial Mappings — but that they are, in fact, so large that coming across anything resembling coherent behaviour becomes highly improbable. Moreover, as algorithms become longer there are many more ways in which their combinatorial spaces may be formed. Thus, a careful consideration of which symbols should be up for substitution and which ones should remain constant can have a significant impact by reducing the size of the result space. This may require considerable trial and error, which involves selecting, isolating and exploring small sets of symbols at a time, as well as experimenting with different symbol domain sets.

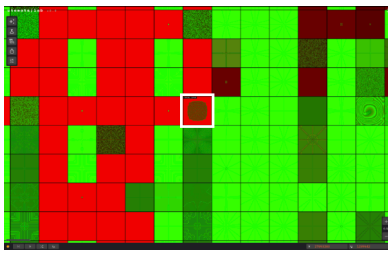


Figure 46.
Grey-Scott Reaction Diffusion and its immediate siblings. (Source algorithm is highlighted)

Reaction-Diffusion (RD) represents a significantly more complicated algorithm, compared to GOL and ECA. It features many more numerical constants, as well as a deeper nested expression. As such it potentially corresponds to much a larger space, which may be formed in numerous different ways, some more effective than others. For example, a maximalist approach may involve selecting every single operator and numerical constant in an RD expression. This, in conjunction with comprehensive parameter domain sets, comprising of all Utomata operators, as well as a large list of numerical constants. Such an approach would create a result space of an unfathomable size. While it is very likely to contain instances of remarkable behaviours, the sheer magnitude of such a space makes it terrible candidate for exploration.

On the other hand, a minimalist approach may involve selecting just one or two operators for substitution, combined with symbol domain sets consisting of one or two parameters. This would yield very small result spaces, which can more easily be explored and examined. However, such minimal spaces are also highly unlikely to feature sufficiently diverse or significant findings. A moderate approach would, for example, involve selecting a handful of operators and constants for substitution or, alternatively, to leave all operators intact and explore only substitutions of neighbourhood variables. It is also worth noting that substitutions and symbol domain sets are not strictly limited to single symbols. For example, one could select the custom neighbourhood expression of RD in its entirety: `sub(V4, mlt(V, 4.0)` and explore any number of variations of this subexpression, or parts of it.

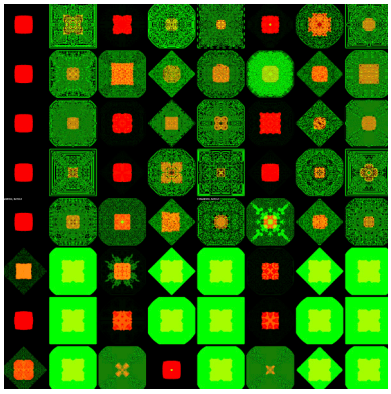


Figure 47.
Algorithmic variations of RD in Utomata Lab.

As previously noted, real-valued numerical constants represent an additional layer of complexity to Spatial Mapping since they cannot be enumerated. Here, too, there are a number of ways to address this complication, each with notable trade-offs. One approach would be to avoid substitutions of numerical constants altogether. While this would indeed help in reducing the size of the space, it is also very likely to impose arbitrary limitations on supported behaviour; it seems unreasonable to expect that a given set of numerical constants would result in noteworthy behaviour across numerous permutations of an algorithm.

A different approach may involve testing different numerical constants, one at a time, along with a stepped set of values (for instance, at increments of 0.25). This approach may be suitable for more targeted studies of a very particular variants. Lastly, a more effective approach involves combining the field parameterisation method with Spatial Mapping by incorporating the cell.x and cell.y values into the source expression, thus allowing each tile in the space to span a continuous range of behaviours in and of itself.

```
Source:
add(
   frc(V),
   vec(
      add(
  sub(
     mlt(0.22,sub(V4, mlt(V, 4.0)).r),
     mlt(V.r,mlt(V.g, V.g))
         ),
         mlt( 0.036, sub(1.0, V.r))
      ),
     sub(
        add(
           mlt(0.05,sub(V4, mlt(V, 4.0)).g),
           mlt(V.r,mlt(V.g, V.g))
         ),
        mlt( 0.097, V.g)
      ),
      0.0
   )
)

Configuration: vec(sml(dst(cell.xy, vec2(0.5)),0.02));
Operators: add, sub, mlt
Variables: V, V4, V8
```

As for the axial distribution, while there are a number of approaches to choose from, note that the RD transition function consists of two closely related expressions, situated in the red and the green channels of a single vector. The distinct symmetry between the two stems from the algorithm's original goal of modelling the process of diffusion between two chemical substances. Therefore, it would make sense to map symbols of the the red channel onto the on one axis and symbols of the green channel to the other. When applied, this approach appears to position the source algorithm in an environment that consists of many similar looking variants, which may suggest that, with this axial distribution, other behaviours of interest might also be embedded in pockets of similar behaviours.

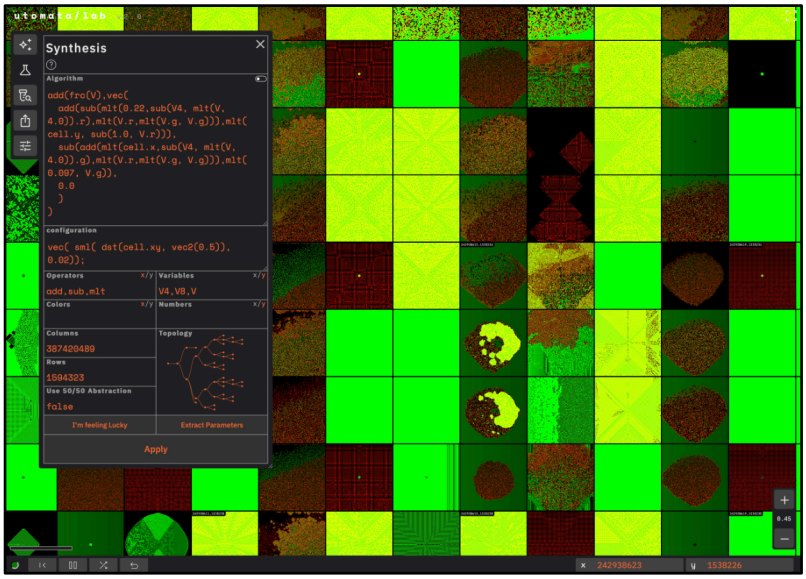


Figure 48.
Spatial Mapping of RD, combined with Field Parametrisation exhibits a continuous range of behaviours for each variant.

5.3. Summary

This chapter introduces *Spatial Mapping,* a novel method for high-level exploration of CA algorithms. It uses a sophisticated mapping technique that accepts a Utomata algorithm and a set of symbols for substitution, and creates a space consisting of all possible variations of the algorithm. This space is demonstrated to be complete, exclusive and distinctive. Utomata Lab is a software implementation of this technique, allowing real-time exploration of any Spatial Mapping. The chapter details the theoretical foundations of this method, discusses its implementation, and demonstrates its use through a series of case studies.

Spatial Mapping is unique in its ability to allow real-time interactive navigation of the combinatorial spaces of any CA algorithm, potentially offering new insights into their behaviour. However, this method also has significant limitations, which this chapter also discusses at length. This method has profound implications for exploring emergent virtual structures, providing a unique lens on CA dynamics that traditional approaches may overlook. The next chapter details the use of this method towards formulation, exploration and examination of a newly identified family of CA algorithms, called *Type-U*. Chapter 8 offers further discussion on the limitations of this approach, as well as its implications for the broader study of emergent virtual phenomena.

6.A study of Type-U

This chapter demonstrates the use of the Spatial Mapping method detailed in the previous chapter towards an exploratory study of a novel family of CA algorithms, called *Type-U*. While this marks the first formal introduction of this algorithm and the first methodical exploration of its countless variations, there have been instances of similar algorithms used in non-academic work, most notably in the work of Jazer Giles [150]. A similar aesthetic is also apparent in the works of Kermi Safa [151], Kim Asendorf [152], Julian Hespenheide [153] and Andreas Gysin [154]. However, these practitioners arrive from live performance, generative, Glitch, and ASCII art disciplines, rather than CA research, and no formal account of their work has been made available at the time of writing.

Section 6.1 provides a formal definition of Type-U, which can technically be classified as continuous-state, outer-totalistic CA. Type-U expressions are characterised by a rigid, recursively defined genotypic structure. Their transition functions are constructed as symmetric binary trees of varying depths, with the root node always being Utomata's `U()` function. Given two parameters, X and Y, this function returns the unaltered state of a single neighbouring cell in relative coordinates. This rigidly defined structure makes Type-U algorithms particularly suited for exploration through Spatial Mapping. Their explicit and symmetric X and Y components allow for an intuitive axial distribution, wherein variations to the left-hand and right-hand expressions correspond directly to traversals of the combinatorial space along the X and Y axes, respectively.

Moreover, Type-U combinatorial spaces exhibit an unusually high ratio of "fruitful" to "sterile" variations. In many other CA algorithms, including those explored in the previous chapter, the vast majority of variants tend to either be sterile — exhibiting no discernible behaviour — or overly chaotic, where excessive activity prevents the formation of emergent structures. In contrast, the rigid structure of Type-U algorithms ensures that a majority of variants generate discernible patterns of behaviour, as demonstrated throughout this chapter.

Sections 6.2 and 6.3 evaluate the unique genotypic and phenotypic characteristics of Type-U algorithms and their encompassing combinatorial spaces. Section 6.4 proposes a qualitative metric for categorising Type-U algorithms, aimed at facilitating further exploration of Type-U spaces. Together, these sections offer an initial investigation into the spatial, genotypic, and phenotypic properties of Type-U algorithms: Spatial properties pertain to the arrangement and composition of algorithms within a given combinatorial space, genotypic properties relate to the structural attributes of Type-U algorithmic expressions, and phenotypic properties describe the observable structural and behavioural qualities of individual algorithmic instances.

It is important to note that the observations presented here are derived from interactive exploration and visual examination, offering only an initial glimpse into this vast family of algorithms and their properties. Nonetheless, it is suggested that this exploratory approach constitutes a valuable tool for gaining insights, developing informed intuition, and establishing expectations about the properties and behavioural boundaries of a given CA algorithmic space.

6.1. Formal Definition

As discussed in Chapter 3, Utomata features a number of built-in variables and functions for accessing common totalistic neighbourhoods:

```
V  // self - the transitioning cell
V3 // three upper adjacent cells (for totalistic 1D CA)
V4 // Von Neumann neighbourhood
V8 // Moore neighbourhood
V9 // Moore neighbourhood and self
```

These built-in variables are themselves defined by directly accessing individual neighbour states using Utomata's built-in `U(dx, dy)` function and adding these values together as follows:

```
V  = U(0, 0)
V3 = U(-1, -1) + U(-1, 0) +U(-1, 1)
V4 = U(0, -1) + U(1, 0) + U(0, 1) + U(-1, 0)
V8 = V4 + U(-1, -1) + U(1, -1) + U(1, 1) + U(-1, 1.)
V9 = V8 + V
```

The U function returns the current state of any cell by accepting two integer parameters: deltaX and deltaY, which signify the target cell's position relative to the transitioning cell. [1]

```
setup = vec(cell.x);
update = U( 1., 0.);
```

Since the U function returns the current state of a particular neighbouring cell, it can be used to induce continuous motion in a system that features toroidal bounds. In the example above, the configured horizontal gradient pattern moves to the left at a rate of one cell per step. This global motion happens as each cell in the system selects the state of its adjacent right side neighbour as its new state at each time step. The system's toroidal bounds ensure that this motion continues indefinitely, since there are effectively no edges on either axis.

Higher local delta values generally result in higher velocity and the ratio between deltaX and deltaY determines the angle of motion. Any deltaX and deltaY parameters can be used, including negative values or even values larger than the size of the grid. For example, `U(1.0, 0.0)` and `U(1.0 + grid.x, 0.0)` both point to the same coordinate because of the grid's toroidal bounds. However, since the grid is discrete, this motion is always stepped. Any variation of this transition function that just uses numerical constants will necessarily result in homogeneous and regular motion. Homogeneity stems from the fact that every cell in the system is given the same two parameters and regularity stems from the fact that these parameters never change.

```
// configuration as a red-green gradient
setup =  vec(cell.x, cell.y, 0.0);

update =
// regular and homogeneous
U( 1., 2.) * tile(0., 0.) +
// regular and heterogeneous
U( cell.x*32., cell.y*32.) * tile(0., 1.) +
// irregular and homogeneous
U( time, time) * tile(0., 2.) +
```

1 *Despite being represented as floating point values, these parameters are, in effect, treated as integers, as they signify relative coordinates on a discrete grid. Therefore, any decimal value provided to the U function is rounded.*

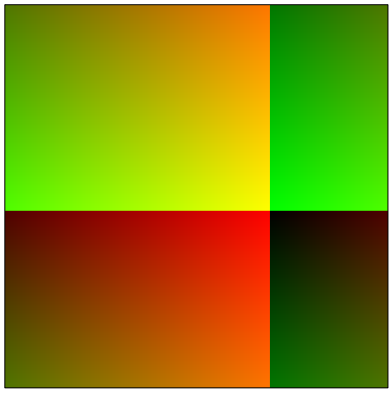


Figure 49.
Regular and homogeneous motion. Here the configuration pattern is preserved but shifted.

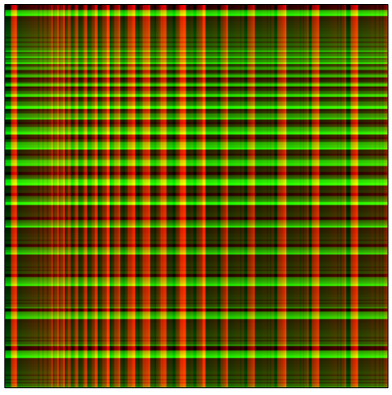


Figure 50.
Regular and heterogeneous motion. Different cells point to different neighbours

```
// irregular and heterogeneous
U( rand(1.)*2. -1., rand(2.)*2. -1.) * tile(0., 3.);
```

In order to achieve other types of motion in the system, different x and y parameters can be used. For example, heterogeneous, but regular motion can be invoked using the `cell.xy` variable, so that it calculates a slightly different result for each cell. Similarly, irregular and homogeneous motion can be achieved using the current time step, which is shared by all cells — but itself is different at each step. This results in a cyclic velocity, as the current time step cycles through multiples of the grid size. Lastly, irregular and heterogeneous motion can also be achieved using the `rand()` function to generate a unique value for each cell at each time step. [2]

The gradual disruption of the configured gradient pattern through irregular and heterogeneous motion, using the `rand()` function, reveals intriguing patterns, as seen in Figure 51. Yet, it is important to note that this behaviour's reliance on pseudorandom values categorises it as a stochastic, null-neighbourhood CA. Though it initially appears capable of intriguing behaviour, this algorithm is a poor candidate for exploration for a number of reasons. Firstly, it is extremely unlikely for higher-order structures to emerge because this process is ultimately very uniform. Even if this would occur by chance, it would be virtually impossible to recreate. More importantly, this algorithm effectively has no meaningful variations to explore because it contains no operators, variables or constants; instead, it "outsources" its entire decision tree to the `rand()` function. Lastly, this algorithm uses pseudo-randomness, which sooner or later is bound to yield repetitive patterns. Luckily, irregular and heterogeneous motion can indeed be achieved through other means.

2 Here the range includes negative numbers in order to prevent a bias towards only the positive direction.

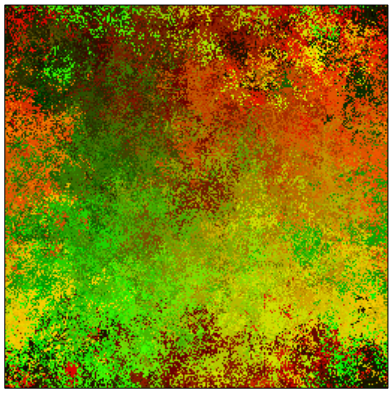


Figure 51.
Irregular and heterogeneous pattern using the `rand()` function

First order Type-U

```
setup = rand(1.,2.,3.);
update = U(V8.g, V4.b);
```

In the above Utomata program, the configuration consists of assigning a random value to each cell and the transition uses every cell's current value to determine its next state. It should be emphasised that the final result of this transition function can only ever be the current — unmodified — state value of some cell in the system (possibly the transitioning cell itself). Hence all possible state values in this, and in all Type-U algorithms discussed in this study, are confined to those created by the configuration function. This is partly the reason for employing a "fully random" configuration in this study, which is further discussed in the next section.

Figure 52.
TU0 - a first-Order Type-U.

This algorithm is deceptively simple — signifying a notable strategy for heterogeneity and irregularity: the incorporation of state into the deltaX and deltaY parameters of the U function acts as the sole driver of behaviour. This strategy turns out to be a surprisingly powerful generator of emergent structures, due to its innate capacity for feedback loops. It constitutes a strong interdependence between state, neighbourhood and transition. While the use of state in determining the result of the transition function is one of the defining characteristics of CA algorithms, what sets this method apart is the fact that the states of the cell and its neighbourhood(s) are used to define a unique neighbourhood, comprising of exactly one cell — for each cell.

In its basic form, this Type-U algorithm does not seem to induce particularly interesting behaviour. However, this strategy for harnessing state towards determining the neighbourhood can easily be extended to create arbitrarily

more compound variations of this expression. By applying the Spatial Mapping method introduced in the previous chapter, these variations can now be explored in-mass. Consider the following mapping parameter sets:

```
Source: U(V8.g, V4.b)
Abstraction: U(Vx.Cx, Vy.Cy)
Variables: V24, V8, V4, V
Colours: r, g, b
```

These mapping parameters include four variables and three colours. The axial distribution is set in accordance with the structure of the U function, so that values that determine deltaX are mapped along the X axis and values that determine deltaY are mapped along the Y axis. The result space thus consists of 144 unique algorithms, organised as a 12 by 12 grid. Figure 53 shows a complete Spatial mapping of first-order Type-U in Utomata Lab.

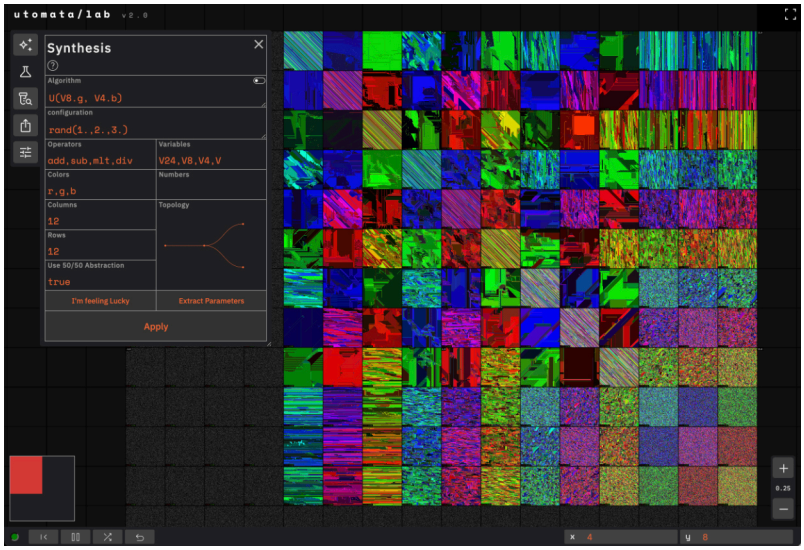


Figure 53.
Complete Spatial Mapping of first-order Type-U.

Second-Order Type-U

A much larger space of Type-U algorithms can be constructed by adding a layer of arithmetic operations for each axis, in the form of binary operators. This allows incorporation of more neighbourhoods and colour variations into the expression and, most importantly, many more combinations of them.

```
Source: U(sub(V4.r, V8.g), sub(V4.b, V.b))
Abstraction: U(Fx(Vx.Cx, Vx.Cx), Fy(Vy.Cy, Vy.Cy))
Operators: add, sub, mlt, div
Variables: V, V4, V8
Colours: r, g, b
```

The above mapping parameters denote a *second-order* Type-U Spatial Mapping. For this mapping, four basic arithmetic operators are used: addition, subtraction, multiplication, and division. Together, they introduce novel ways in which the totalistic neighbourhoods and colour channels can be combined to produce a range of more intricate results for any given expression.

```
U(Fx(Vx.Cx, Vx.Cx), Fy(Vy.Cy, Vy.Cy)) // expression
  4  4  3   4  3    4  4  3   4  3    // combinations
deltaX: 4 * 4 * 3 * 4 * 3 = 576
deltaY: 4 * 4 * 3 * 4 * 3 = 576
deltaX * deltaX = 331776              // unique expressions
```

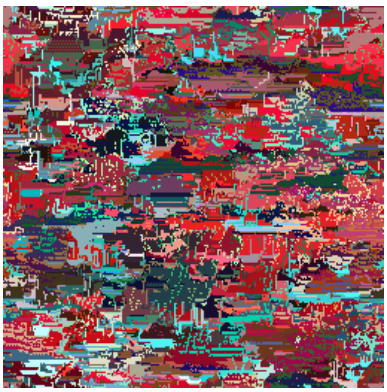


Figure 54.
TU72 is an example of a second order Type-U algorithm. This algorithm is documented in the next chapter.

The number of unique second-order Type-U algorithms is higher by an order of magnitude. While the first-order space consists of four operators and three colours, giving 144 unique algorithmic expressions in total, with the introduction of two binary operators for each axis, the second-order space consists of 331,776 unique expressions. Notably, because this new space consists of exponentially more types of local interactions between cells, it features a plethora of CA behaviours which are potentially not present in the lower-order space. Unlike the first-order space, which is captured in its entirety in figure 53, this space is much harder to capture and explore — though it is still possible. Extended exploration of second-order Type-U in Utomata Lab can offer a reasonably comprehensive view of the range of behaviour it encompasses.

Higher-Orders

The transition from first-order to second-order Type-U described above essentially involved replacing the variable and colour symbol (V.C) with a binary operator that includes two variables (with their symbols) for both parameters of the U function. This substitution rule can be abstracted to form a recursive definition of higher-order Type-U spaces, not unlike performing iterations on an L-System string. The substitution can be defined as replacing every occurrence of a symbol with a binary operator as follows:

```
V.C >> F(V.C, V.C)
```

As such, an Nth order Type-U expression can be constructed by applying this rule on a first order Type-U expression — N times. Beyond differing significantly in size, each order also features several unique properties, which may or may not be present in other orders — lower or higher. These are discussed in the following sections. Appendix A denotes Type-U orders one through six.

6.1.1 Scope

There can be numerous ways to define Type-U algorithms, as well as to construct their combinatorial spaces. However, in the context of this chapter, which aims to serve as an introductory exploratory study of Type-U, the scope of algorithms considered has been limited to include the use of a particular configuration, mapping parameters and axial distribution method. It is important to note that these particular choices are not put forth as a definitive or even an optimal form of neither Type-U algorithms or their mappings. Rather, their selection serves to facilitate a robust examination and demonstration of the capabilities which may be inherent to these algorithms. While they are beyond the scope of this thesis, alternative Type-U mappings and comparative studies of different symbol domain sets may very well present more interesting outcomes and should be considered worthy candidates for future studies, examples of which are suggested in Chapter 8.

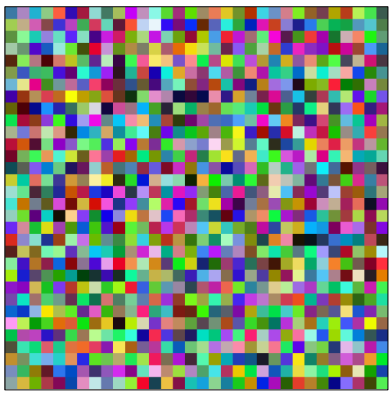


Figure 55.
*Fully random configuration on a 32*32 grid using rand(1.0, 2.0, 3.0).*

This study relies on Utomata's built-in GLSL approximation of a random number generator. However, a comparative study showed no significant differences from a configuration based on pseudorandom or true random numbers.

Configuration

The configuration in CA determines the system's initial conditions and thus plays a significant role in determining its behaviour, which is typically highly chaotic. In Spatial Mapping, the use of an appropriate configuration function is crucial, as the advantages of observing large numbers of algorithmic variations can easily be nullified by a configuration that fails to induce noteworthy dynamics to the system. This type of exploration leans heavily on an ability to visually discern structures of interest among a large set of similar, but not identical, patterns.

In Type-U algorithms the configuration has an even more dramatic effect. This is due to the inherently reductive nature of Type-U dynamics, whereby the transition function is strictly incapable of introducing new state values to the system. It is therefore up to the configuration function to include as many different state values as possible in order to maximise the number of possible structures that may subsequently emerge.

It may sometimes be useful to consider Type-U algorithms as “unguided sorting algorithms”, where the behaviour of the sorting mechanism is determined by the data (state values) in real-time, acting differently on different parts of it. This process of sorting and a-sorting constantly adapts as the system evolves and changes its composition. This feedback mechanism relates to the aforementioned web of interdependencies between state, neighbourhood and transition. Through this process, the rapid elimination of state values, often shortly after configuration, can easily render a system sterile. Therefore, a configuration that is insufficiently diverse can profoundly constrain the range of phenomena in a Type-U algorithm. For this reason, a fully random, maximally inclusive configuration function has been deemed optimal for this initial, exploratory study. That said, the use of more constrained configurations may be still useful in certain cases. For example, different configuration patterns can be tested in order to evaluate the limits of behaviours which can emerge in a particular algorithm of interest.

Mapping parameters

The symbol domain sets from which a Spatial Mapping is formed also have a significant impact. Apart from determining the size of a space, they directly affect its composition, organisation, as well as the range of possible supported phenotypic behaviour. These are regarded here as the mapping parameters. For example, adding a single operator to the operators set will result in an exponentially larger space, as it is now supplemented with all of its existing combinations with this new operator. Unless otherwise stated, the studies and experiments presented in this chapter refer to the following minimal mapping parameters:

```
Operators: add, sub, mlt, div
Variables: V24, V8, V4, V
Colours: r, g, b
```

These have been selected in an effort to achieve a balance between minimising the size of the combinatorial spaces explored, while maximising the range of phenomena they encompass. The set of variables includes the state of the transitioning cell, as well as three well-established totalistic neighbourhood types. This allows a range of cell interactions to be driven by both low and high state values. The incorporation of V24 in particular, results in a significant increase of dynamic behaviour in a space. This is presumably because it allows algorithms to include references to neighbours that are further away, in turn, potentially increasing the density and variability of the networks of local cell interactions.

Likewise, utilising just four fundamental arithmetic operators appears to strike a careful balance between facilitating a diverse range of behaviours and maintaining mappings of a tractable size, particularly in the context of lower-order spaces. From this minimal set, other operators can be assumed to reside somewhere in the space through compounding. For Example, exponentiation can be excluded from the mapping parameters under the assertion that it can be achieved through multiplication. Likewise, scaling by large numbers can be achieved through division over small fractions.

The `div()` operator introduces the most distinct dynamics of all four. Its inclusion in some cases presents extremely volatile behaviour, whereby large regions of the grid may suddenly flip to a uniform colour. This is presumably because division has the capacity to return arbitrarily high values, though further testing is required. Unlike in many other CA transition functions in Utomata, high values are not strictly clamped by Type-U transition functions because the calculation is always used to produce the deltaX and deltaY coordinates of the U function. As previously mentioned, the use of toroidal bounds in Type-U also means that all possible values return a valid coordinate.

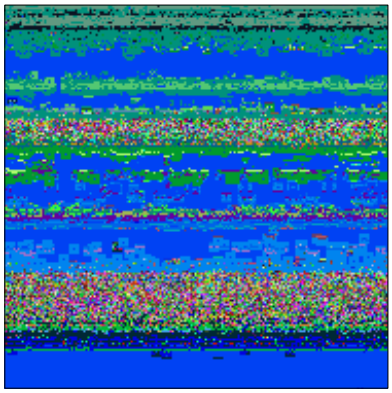


Figure 56.
TU81. The `div()` operator often causes extremely volatile dynamics.

Initial tests suggest that the use of Utomata's boolean operators — `sml()`, `lrg()` and `eql()` — offer no substantial contribution to the percieved phenotypic diversity in a space. This likely stems from the boolean (0.0 or 1.0) return value of these operators, which often drives the U function to return the state of the transitioning cell itself, thereby creating a strong bias towards static behaviours.

The incorporation of RGB colour channels is essential in Type-U because the U function expects two integer values, rather than 4D vectors. However, beyond this structural requirement, the incorporation of colour channels also acts as an important feedback mechanism between horizontal and vertical dynamics.

```
update = U(sub(V9.b,V9.g),div(V4.g,V9.g))
```

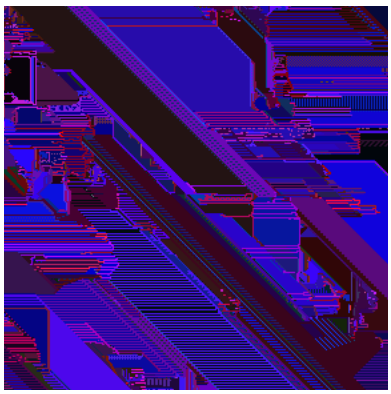


Figure 57.
TU82 does not share colour values across both axes and therefor displays monotonous behaviour.

For example, in the above algorithm, depicted in Figure 54, the deltaX component refers to the blue and green values of the Moore neighbourhood, while the deltaY component refers to the green values of the Von Neumann and Moore neighbourhoods. The fact the the green channel is shared by both axes means that there is necessarily some level of interaction and feedback in the process of determining the horizontal and vertical transposition of cells. The importance of this property is often evident in algorithms where this is not the case, such as the one below (seen in Figure 57). In these cases, the values of the deltaX and deltaY components are determined independently of each other, subsequently presenting a more limited range of state interactions. This could suggest a further explanation as to why higher-order spaces appear to contain much richer phenomena: longer expressions feature a higher probability for the same colour channel to be used, at least somewhere, on both axes.

```
update = U(sub(V.r,V4.r),add(V.g,V4.g))
```

Axial distribution

Selecting an appropriate Spatial Mapping method ideally involves a careful consideration of the type of investigation (i.e: exploratory, analytical, result oriented), as well as properties of the algorithms in question. In the case of Type-U, the fact that the U function is ultimately driven by explicit X and Y parameters are best taken into account. This innate property of the algorithm, whereby each cell selects some other cell using a 2D coordinate, strongly suggests that an XY based mapping, such as the one described in Section 5.1.1, shown below, is indeed appropriate. As such, it potentially makes traversal of the space more intuitive, since moving along each axis corresponds with a permutation of the algorithm on that same axis.

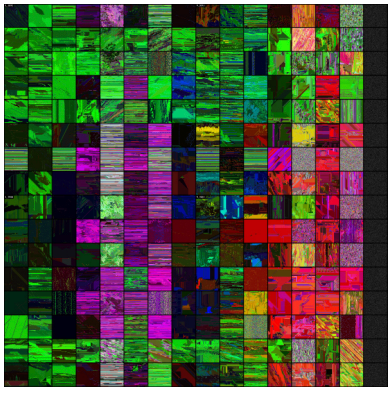


Figure 58.
Alternative axial distribution where operators are plotted on the X axis and variables and colours on the Y.

```
U(Fx(Vx.Cx, Vx.Cx), Fy(Vy.Cy, Vy.Cy)) // XY distribution
```

A number of alternative axial distributions have also been considered. For instance, plotting operators on the X axis and variables and colours on the Y axis (shown below) would yield a space consisting of the exact same set of algorithms, yet distributed very differently. This arrangement highlights different

relationships between neighbouring algorithms in a space, which may or may not be beneficial, depending on context. Notably, unlike the symmetric axial distribution method above, which creates spaces consisting of an equal number of rows and columns, this mapping method results in spaces that are highly skewed. For example, the mapping below conforms to a space with 16 columns and 20,736 rows. Notably, traversing columns and rows in such a space no longer represents permeating the X and Y parameters of the U function, but rather its operators and variables.

```
U(Fx(Vy.Cy, Vy.Cy), Fx(Vy.Cy, Vy.Cy))
```

There are many other ways in which permutations can be plotted along a two dimensional field. Different mappings result in a different arrangement of algorithms along that field, thereby having a direct effect on exploration. For example, a mapping created by randomly generating a Type-U algorithm for each cell in the field would result in a highly irregular space in terms of its genotype, where proximity or adjacency of algorithms have no meaning whatsoever. Such a mapping would therefore serve as a very poor candidate for a study that aims to explore direct siblings — close permutations of a given algorithm. Moreover, unlike the mappings described in the previous section, such a mapping is not guaranteed to contain all possible permutations of the algorithm in question and may also feature high levels of redundancy in the form of algorithms that happened to have been randomly created more than once.

One may therefor assume that a random mapping would also yield a highly irregular space in terms of its phenotype for this same reason. However, this actually depends on the algorithm in question and the size of the space plotted. For example, almost all permutations of sixth-order Type-U algorithms rarely yield any discernible structures whatsoever. Therefore a random mapping of sixth-order Type-U is actually more likely to feature regularity in terms of phenotypic properties, with occasional isolated instances of discernible structures, scattered randomly across the entire space. However, while they are indeed poor candidates for exploratory studies, random mappings are by no means useless. They can serve as a reliable tool for statistical analysis of a given combinatorial space and for quantitative evaluation of the range of behaviours apparent in the algorithms it encompasses.

6.2. Spatial Properties

The study presented in this chapter serves a dual purpose: it aims to exemplify the effectiveness of the Spatial Mapping method in regards to open-ended exploration of novel CA, while also serving as a formal introduction to Type-U algorithms. This creates a potential clash between the need to discuss general properties which may be inherent to Spatial Mappings and the need to exposit specific properties of the family of Type-U CA. While the following two sections discuss spatial and phenotypic properties of Type-U respectively, it should be noted that these two properties are fundamentally linked. Nonetheless, an exploratory and qualitative approach to CA research, such as the one presented in this chapter firmly aligns with both stated goals.

Type-U algorithms feature a number of distinctive traits that make them exceptionally suitable for exploration through Spatial Mapping. As previously noted, their consistent algorithmic structure is characterised by explicit X and Y components, which enhances the effectiveness of exploring them in a 2D space. Type-U algorithms also lack numerical constants, ensuring that their combinatorial spaces can be comprehensively plotted. Their scalability allows for the comparison of algorithms across different orders while preserving most

of their shared characteristics. Lastly, the (deceptive) simplicity of their transition function, essentially culminating in the selection of a single cell, results in the majority of Type-U algorithms yielding at least some discernible behaviour. These properties stand in contrast to many other CA families which have been examined in previous chapters; most of the algorithms discussed so far appear to be *"an almost miraculous exception"* in a space consisting almost entirely of chaotic or uniform behaviour.

This section identifies a number of properties and characteristics of Type-U spaces, while the next section deals with phenotypic properties of the algorithms themselves. Together, they offer a broad view of Type-U dynamics, attempting to characterise their behaviour and limits, laying out a foundation for more targeted future explorations of Type-U and their countless variations.

6.2.1 Redundancy

Any comprehensive enumeration of CA algorithms necessarily contains redundancy — the presence of identical, or nearly identical behaviours. This may stem from a number of reasons, which may be highly codependent and may vary significantly between different algorithms. Generally, they can be divided into two categories: *genotypic redundancy*, which stems from apparent similarities between algorithmic expressions and the properties of their operators, and *phenotypic redundancy*, in which dissimilar algorithmic expressions result in similar behaviour. Assessing the level of genotypic redundancy may be complex, but can be achieved with a relatively high degree of accuracy through analytical methods. However, phenotypic redundancy is far more difficult to ascertain.

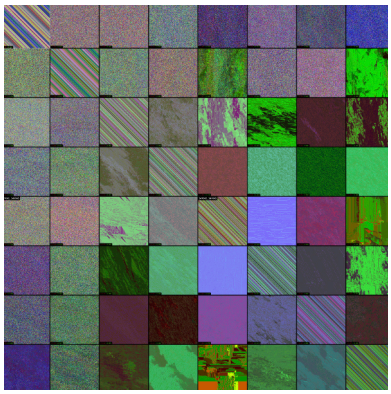


Figure 59.
Diagonal Symmetry in second-order Type-U.

Characterising the various types and levels of redundancy in a Spatial Mapping can be crucial in facilitating effective exploration. This is true even in cases where redundancy is never actually mitigated, but instead simply taken into account. It is also important to note that redundancy, in and of itself, is not necessarily detrimental to exploration. In a reasonably well-understood spatial arrangement, the presence of multiple identical variations may even assist in locating particular nuanced behaviours of interest. Likewise, obtaining a better understanding of redundancy may assist in identifying regions of a space that are of no consequence in order to isolate and expose regions of interest.

Diagonal Symmetry

The first-order Type-U space described in section 6.1.1. consists of 144 algorithms, arranged as a 12 by 12 grid. Upon visual examination, phenotypic behaviour in the space appears to feature a distinct diagonal line from top left to bottom right. The algorithms along this diagonal feature highly constrained structures and motion, all going in parallel to it. In every such instance, the deltaX and deltaY parameters are identical, which explains why they exclusively feature such limited capability: whatever value is returned by their totalistic neighbourhood variables, any cell can only ever select the state of a cell situated along its own diagonal.

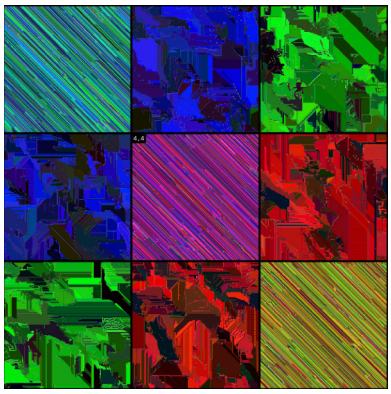


Figure 60.
Twin algorithms along the diagonal in first-order Type-U.

The deltaX and deltaY parameters of all first-order Type-U algorithms both consist of a single variable symbol, followed by a colour channel symbol; they exhibit perfect symmetry and are therefore treated in the same way by the mapping, each along their corresponding axis. This means that for every single algorithm in this space — `U(v1.c1,v2.c2)` — there exists a twin-algorithm — `U(v2.c2,v1.c1)`. The phenotypic properties of these two algorithms are thus identical — but rotated by 90 degrees. For example, the algorithms located at (5, 7) and (7, 5), which correspond to `U(V4.b,V8.g)` and `U(V8.g,V4.b)`

respectively, can be regarded as twin algorithms because they effectively induce the exact same behaviour. This symmetry holds true for every Type-U space created using this axial distribution, regardless of its order.

One of the distinct advantages of using this axial distribution method is that it neatly divides the space into two distinct regions — above and below the diagonal. Algorithms along the diagonal itself still conform to this pattern, however, since their x and y components are identical they have no twins. Thus, due to this inherent symmetry, the number of unique phenotypic behaviours in any Type-U space is actually only about half of its total number of algorithms, consisting of all algorithms above or below the diagonal, as well as those along it. For example, out of the 144 algorithms in the first order space, there are actually no more than 78 unique phenotypic behaviours. [3]

Operator Symmetry

Another apparent example of genotypic redundancy relates to operator symmetry. For example, addition and multiplication, which together are featured in exactly half of all algorithms in the space, are effectively symmetric, such that (A*B == B*A) and (A+B == B+A). This means that any mapping that features symmetric operators in its domain will necessarily contain multiple copies of identical phenotypic behaviours for every single occurrence of a symmetric operator. This, of course, is not unique to Type-U algorithms and likely affects many other mappings. [4]

Assessing the level of redundancy due to operator symmetry requires taking into account the number of occurrences of operators in the expression and the composition of operators in the domain. For example, in the second-order mapping, each expression contains two operators and the operator domain set consists of two symmetric (add, mlt) and two asymmetric (sub, div) operators. Therefor it can be deduced that exactly half of all algorithms in the space feature two symmetric operators and another 0.25 contain exactly one. Thus each algorithm in the former group would have three twin algorithms and each one in the latter group would have one.

Unlike diagonal symmetry, which can simply be taken into account in the process of exploration, redundancy due to operator symmetry is slightly harder to locate using the mapping method in its current form. While a more sophisticated mapping function can indeed be devised to avoid operator symmetry altogether, this may come at the cost of compromising the structural integrity and continuity of the spatial arrangement by forming irregular gaps between neighbouring algorithms.

Phenotypic Similarity

Through a combination of diagonal and operator symmetry alone, it can roughly be estimated that no more than 15%-20% of Type-U algorithms in their respective spaces are actually unique. However, while this kind of redundancy can be precisely calculated and, to an extent, even mitigated for a given space, there exists a far more illusive form of redundancy which may effectively be unknowable. This can be regarded as phenotypic redundancy, as it stems from the possibility of two distinctly dissimilar algorithmic expressions yielding indistinguishable phenotypic behaviours. As is the case with genotypic redundancy, this form of redundancy is not strictly unique to Type-U algorithms, yet they do appear to be particularly vulnerable to its effects.

Since all Type-U transition functions ultimately select a single neighbour, they feature a relatively limited set of outcomes, compared to other continuous state CA. Moreover, phenotypic redundancy also appears to be highly dependent on

3 *The exact number of unique algorithms, k, in a symmetric space with n rows and n columns can be expressed as the following arithmetic progression:*

$$k = ((n * n) - n) / 2 + n$$

4 *The Spatial Mapping of ECA algorithms, as presented in the previous chapter is a good example of staggering levels of redundancy due to operator symmetry. The mapping results in a space consisting of 262144 unique algorithms, while the number of unique phenotypes is known to be exactly 256. Moreover, when accounting for black&white symmetry the number is actually 128.*

the order of the space. Since higher-order spaces, by definition, consist of more deeply nested expressions, it is safe to assume that they feature far more phenotypic redundancy than lower order ones. In an expression which consists of several nested operators, the impact of any given change to the expression diminishes in proportion to its depth — its distance from the root operator. For example, changing the operator at the first level (above the U function) from addition to division is likely to have a dramatic effect on the outcome. In contrast, the impact of the same substitution at the fifth layer of a sixth order algorithm may have very little or even no effect on phenotypic behaviour.

The fact that deeper nested differences (closer to the leafs in the expression tree) between algorithms have a diminished effect on their phenotypic differences suggests that higher-order spaces feature increasingly more phenotypic redundancy as they contain exponentially more instances of increasingly minute differences between behaviours. Still, as previously noted, this kind of redundancy should not necessarily be considered as a side effect or obstacle for exploration. High levels of phenotypic redundancy in a space also mean that particular desired behaviours of interest may be recovered at an arbitrary level of precision. A more robust assessment of phenotypic redundancy requires a more robust classification method of Type-U phenotypic behaviours, which is beyond the scope of this thesis.

6.2.2 Texture

The exact method by which algorithms are distributed in a Spatial Mapping can have a significant impact on its exploration and study. Ideally, algorithms would be organised in a way that expresses the relationships between them. Continuity refers to the level of difference between neighbouring algorithms in a space and contiguity refers to the tendency for similar behaviours to be grouped together. Generally, spaces that feature a continuous and contiguous texture offer significant advantages.

Continuity

A space can be said to possess a smooth, or continuous texture if the difference between adjacent algorithms is minimal. For example, a spatial arrangement where each algorithm is randomly positioned would typically present a rough texture. This can be likened to a geographical terrain, where a height map of the entire space can be plotted by evaluating the difference between adjacent algorithms. Rough textures indicate a weak correlation between adjacency and similarity, while smooth textures indicate a strong correlation.

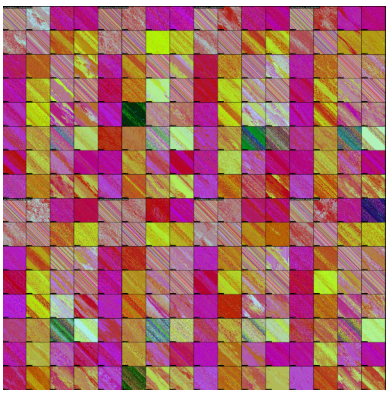


Figure 61.
Smooth textured area in fifth-order space. Any single permutation typically has a minimal effect on the output.

An evaluation of the texture of a space may relate to any measurable property of its algorithms. Hence, a space can be evaluated as having any number of textures. For example, a measurement of the overall level of brightness in an algorithm can be evaluated to give a unique score for each algorithm in a space (or a subspace). This score can then be plotted to reveal the texture of the space in regards to that particular heuristic function. A continuous space would be said to feature a smooth gradient of brightness values and a non-continuous space would feature roughness in the form of adjacent algorithms in which measured levels of brightness differed substantially. A space can thus be characterised as continuous in one respect and non-continuous in another.

The correlations between different properties of algorithms in a space, and therefore between their evaluated textures, can be complex and hard to evaluate. Some properties, such as average velocity and colour diversity, may be linearly and positively correlated, while other properties may feature inverse correlations, non-linear correlations or none at all. Preliminary visual

examinations suggest that Type-U algorithms generally possess a relatively strong correlation between their genotype and phenotype. Since genotypic texture can be considered smooth in Type-U mappings, due to their effective distribution method, they can generally be considered as also possessing smooth phenotypic textures as well. Phenotypic continuity is also strongly related to phenotypic redundancy. In high-order Type-U mappings (third-order and above), the massive increase in nearly identical phenotypic behaviour also results in a significant increase in the overall levels of continuity in the space.

Of course, there are bound to be notable and, most likely, unavoidable ridges and sharp features in any space. For example, a substitution from multiplication to division is typically more likely to have a dramatic effect, compared to a substitution from addition to subtraction. Moreover, as noted in the previous section, the depth of a given change is also of great significance, whereby deeper nested changes are more likely to have a diminished effect.

Distinct behaviours may often cluster along rows and columns, as observed in the GOL Spatial Mapping in the previous chapter. Clustering of phenotypic behaviours may take various forms. For example, the first-order Type-U mapping appears to be divided into distinct sets of triplets. This stems from the relatively coarse granularity of the space, where progressing along each axis translates to cycling through the colour channels of a single variable at a time. This clustering is rather unique to the first-order mapping. Higher-order spaces refer to more than just one colour per axis, thereby diminishing the impact of any single colour channel permutation.

Contiguity

A closely related spatial property refers to the concentration of similar algorithms within self-contained regions. A Spatial Mapping can be said to be contiguous if it features a correlation between spatial proximity and phenotypic similarity. This can manifest as the tendency of a space to group related behaviours together. Having distinct spatial arrangements, or “pockets” of similar behaviours is generally advantageous for exploration as it means that in order to discover permutations of a given algorithm, one only needs to locate it in a space and observe its immediate surroundings.

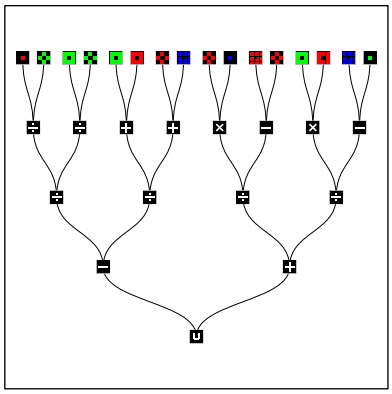


Figure 62.
Binary tree of fourth-order Type-U.

A contiguous space is also likely to feature a correlation between genotypic and phenotypic texture, assuming an ordered mapping method was used. This indeed appears to be the case in Type-U spaces. This property is by no means trivial and is an example of what makes certain Spatial Mappings more useful than others. Contiguous spaces that feature correlation between genotypic and phenotypic regularity are easier to explore because they are organised in a way which can be better made sense of, making manual explorations more intuitive, as well as making higher-level evaluations of a space, such as sparse sampling, more viable.

The contiguity of Type-U mappings is also closely related to their order. First-order and second-order Type-U spaces seem to have less distinct regions. This is presumably because of their relatively shallow depth, in which substitutions between symbols of adjacent algorithms have more impact. Third and fourth order Type-U spaces appear to be more contiguous, perhaps striking a better balance between algorithmic depth and spatial regularity. Fifth and higher order spaces feature increasingly smoother terrain in the form of large regions of similarity sometimes spanning thousands or even millions of algorithms. These spaces are so large that they effectively become too smooth to feature what may be considered distinct pockets of behaviour.

6.2.3 Composition

As a general rule, the larger a combinatorial space becomes, the more likely it is to contain a wide range of behaviours. This, of course, comes with the heavy cost of unique behaviours being scattered across exponentially larger fields. Thus, one of the biggest challenges involved in exploration via Spatial Mapping is minimising the size of the space while maximising the diversity of the algorithms it encompasses.

Along with assessing the redundancy and texture of a given space, assessing its composition, and specifically its range of behaviours, plays a key role. Striking a good balance between having a tractable space to explore and making sure it contains sufficiently diverse phenomena is, in large part, determined when formulating the source algorithm and mapping parameters.

For example, an exhaustive exploration of the space of Reaction-diffusion (RD) algorithms, as demonstrated in the previous chapter, is by all accounts a non-tractable task. This is due to a number of properties of the algorithm, including its reliance on real-valued numerical constants, as well as its sizeable length. Hence, while it is reasonable to assume that RD spaces do indeed contain novel and potentially noteworthy RD-like phenomena, it is highly unlikely for manual exploratory studies to uncover any of it. However, larger spaces do not necessarily feature larger phenotypic diversity. For example, The Spatial Mapping of ECA, also presented in the previous chapter, is said to contain more than a quarter million unique algorithms, yet presenting only 256 unique behaviours.

Type-U algorithms feature a relatively narrow range of phenotypic behaviour, which appears to have distinct characteristics and limits. [5] This is a result of several unique properties, including their rigidly defined algorithmic structure, limited set of symbols, lack of numerical parameters and the fact that their root operator is always the U function. While a limited range is not typically ideal for open-ended exploration, it offers two distinct advantages. Type-U combinatorial spaces are relatively small, compared to many other CA families, and they feature unusually high occurances of uinque discernible behaviours.

5 For example, structures that exhibit radial symmetry, which are quite common in other CA models, are highly unlikely to emerge in Type-U due to their distinct use of outer-totalistic neighbourhoods.

The recursive definition of Type-U algorithms in ascending orders, presented in Section 6.1 is an explicit attempt to gain some level of control over the composition of Type-U spaces. It allows for adjusting the algorithm itself towards the type of exploration to be conducted. Lower-order spaces, which contain fewer algorithms and, accordingly, notably less diversity, are more suitable for targeted, analytical or general studies of the characteristic behaviours of Type-U phenomena. Higher-orders, on the other hand, are more suitable for broad stroke moonshot expeditions to discover novel Type-U phenomena.

Higher-order Type-U spaces contain profoundly more algorithmic permutations and thus are generally more likely to support behaviours not present in lower-order ones. However, the exact relationship between the order of a Type-U Spatial Mapping and its level of diversity is currently unknown. While it is apparent that third and fourth order mappings do contain behaviours that are not present in the first order, this does not necessarily indicate a general rule.

The relationship between the order of a Type-U space and its level of diversity is also not necessarily linear. Initial excursions of fifth order spaces have, thus far, not yielded findings that are fundamentally different from those which can be found in fourth-order spaces, even though the former contains vastly more permutations. This implies that there may be a sweet spot between the size of a

Type-U space and the range of phenomena it encompasses. Of course, this implication is by no means conclusive, nor can it be assumed to hold true for other CA algorithms. Further comprehensive studies of Type-U spaces and the Spatial Mapping method itself are needed. A quantitative analysis of the phenotypic composition of Type-U spaces is beyond the scope of this thesis. However, as discussed in Chapter 8, such a study can indeed make use of its accompanying software implementation, Utomata Lab, even in its current form.

Initial excursions of Type-U spaces, performed as part of this research nonetheless offer a number of qualitative accounts of characteristic behaviours of Type-U algorithms. Below are a number of properties that are correlated with the order or the size of the space in which the algorithm in question resides. The next section discusses characteristic Type-U behaviours in general.

Directional Bias

Heterogeneous and irregular motion are presented in section 6.1 as a core aspect of Type-U transition functions. The various ways in which different cells in the grid select their new state can be likened to a vector field (illustrated in Figure 64). As such, the angle and length of each arrow in the field are determined by the transition function for each cell. This can be a helpful framework for visualising the overall levels of directional bias in an algorithm. For example, as discussed in Section 6.2, algorithms situated along the diagonal of a space feature identical deltaX and deltaY components. Therefore, the angle of all resulting vectors will necessarily be 45 degrees, limiting all calls to only access cells along their own diagonal.

Moreover, due to their simplicity and lack of operators, first-order Type-U algorithms feature distinct directional and colour biases, not just along the diagonal. The general angle and overall velocity of a first order algorithm can be predicted regardless of its configuration, demonstrating a bias towards the axis in which the larger neighbourhood value lies. For example, the algorithm `U(V24.r, V4.g)` can be predicted to feature overall distinct horizontal motion. Notably, this does not necessarily affect every single cell in the grid, as some cells may have low red values in their extended-Moore neighbourhood and high green values in their Von Neumann neighbourhood.

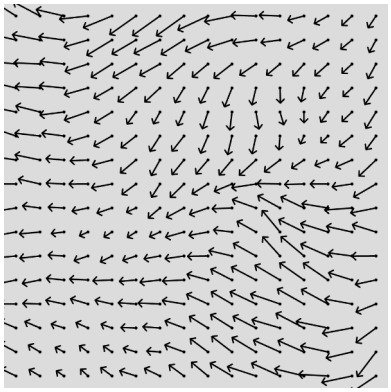


Figure 63.
Vector field (illustration).

Moreover, all algorithms in the first-order space move towards the negative direction on both the X and Y axes. This is a result of the fact that deltaX and deltaY are always greater than or equal to zero. Thus, under the stated mapping parameters, it is impossible for the transition function to return the value of any cell above or to the left. A resolution of this bias, in the form of positively directed motion, is only introduced in second-order Type-U with the inclusion of the `sub()` operator. However, this is also not to say that all second order (or above) algorithms are unbiased. In fact, every single Type-U algorithm which does not contain the `sub()` operator necessarily is. By extension, this holds true for any Type-U space in which the `sub()` operator is not included in the domain. [6]

Neighbourhood and Velocity

The amount of motion in first-order algorithms, while generally heterogeneous, is still dependent on the composition of variables in their respective expressions. For example, the 12 algorithms that exclusively use the V variable feature noticeably more static behaviour compared to the rest. This stems from the limited mobility of state transitions that can select, at most, the adjacent neighbour to their right or bottom. In contrast, algorithms that incorporate the V24 variable can support a velocity of up to 24 cells per step. This property can therefore also predict the general direction of movement. The axis containing the larger neighbourhood value will typically feature overall higher velocity.

6 *There are, of course, other ways to resolve directional bias without using the* `sub()` *operator. For example, one could include negative variables such as* `(-V4)` *in the domain.*

Colour Bias

In all first-order algorithms, a histogram analysis shows a distinct tendency towards one or two of the primary colour channels. Since any first-order expression can, at most, feature two colour symbols, it appears that the value of colours which aren't present in the expression typically follow an even distribution, while the value of colours that are featured tend to gradually be reduced as the system evolves. This is likely the result of an implicit selective process, whereby higher values are more likely to cause the transition function to return the state of a cell that is not the transitioning cell.

For example, consider the transition function: `U(V8.b,V.r)`. Histogram analyses over several runs consistently show an even distribution of values along the green channel, a tendency for lower than 0.5 values along the red channel and distinctly low values along the blue channel. These readings align with how the U function operates: in areas that are rich with blue values, the transition function is very likely to change the state of cells to those up to 8 cells to their right. Likewise, cells whose red channel is higher than 0.5 will select the state of a cell below them. In contrast, the presence of green values does not appear to directly affect the transition process. It can therefore be predicted that the system will retain its green colours (with low reds and low blues) through a gradual process of selection.

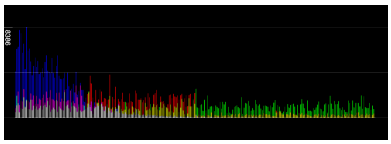


Figure 64.
Histogram of `U(V8.b,V.r)` after 30 steps.

Second order algorithms appear to feature a significant increase in overall colour diversity. This can be attributed to the fact that expressions can now refer to all three colour channels, as well as perform rudimentary calculations in mixing them together. While distinct tendencies towards basal RGB colours are still apparent in second-order Type-U, the increase in algorithmic variations introduces visibly novel colour schemes. In third-order mappings and above, colour bias is dramatically reduced. Generally, this can be attributed to the fact that longer expressions are both more likely to include all three colour channels, as well as more nuanced mixtures through more deeply nested operations.

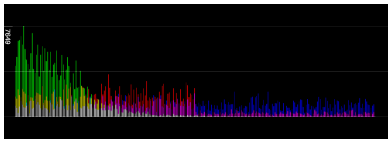


Figure 65.
Histogram of `U(V8.g,V.r)` after 30 steps.

Idiosyncrasy

Some Spatial Mappings may feature characteristic phenotypic behaviour. For example, First and Second order spaces seem to consist of a relatively uniform set of possible behaviours, compared to higher-order spaces. This property is, of course, directly related to diversity. However, idiosyncratic behaviour in a space may indicate more than just a lack of diversity; certain mappings may feature a distinct overall behaviour of their algorithms that may be considered unique.
Third and fourth order spaces contain numerous instances of a dynamic that is distinctly "fluid", featuring semi-stable structures (shown in Figure 66). These behaviours do not appear to be present at all in first or second-order spaces, and quite hard to locate in fifth and higher orders.

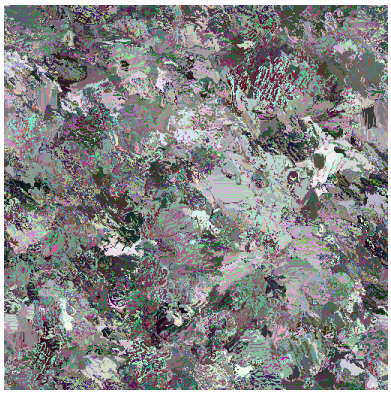


Figure 66.
TU68 exhibits fluid-like dynamics in second order Type-U. This dynamic is most commonly found in third order algorithms.

Containment

The recursive definition for different orders of Type-U raises a question regarding the extent to which low-order mappings are contained within high-order ones. The notion of containment can be referred to as the degree of overlap between different orders, and more generally, between different CA combinatorial spaces. While containment may be easy to assess in regards to the genotype, this does not necessarily translate to the phenotype. Not one of the 144 algorithms in the first-order space can be found in the second order. For instance, the behaviour `U(V4.r,V.b)` simply cannot be formulated as a second-order algorithm since there are no operations of type `F(V.C, V.C)` that return neither `V4.r` nor `V.b`. In fact, the recursive introduction of operators for all subsequent higher-orders precludes explicit genotypic containment. This, however, does not mean that the phenotypic behaviour invoked by `U(V4.r,`

`V.b`) cannot be approximated to an arbitrary degree by a higher-order algorithm.

In any case, lower order containment in Type-U mappings can easily be achieved by simply incorporating two values — 0.0 and 1.0 — into the variables domain. This would result in numerous permutations that contain addition of 0.0 and multiplication by 1.0, which would ensure the presence of all lower-order expressions in all higher-order mappings. Moreover, these algorithms are likely to be featured many times over due to operator symmetry. However, this would also result in the unavoidable inclusion of algorithms that feature addition of 1.0 and multiplication by 0.0, which may or may not be desired. Referring to the example above, the operation of type `F(V.C, V.C)`, which represents a single delta component of a second-order algorithm, would now be able to return `V4.r` in four different ways: `add(V4.r,0.0)`, `add(0.0,V4.r)`, `mlt(V4.r,1.0)` and `mlt(1.0,V4.r)`. While this would indeed ensure containment, this redundancy would quickly stagger, flooding higher-order mappings with numerous instances of redundant behaviour. For this reason, these values were excluded from the mapping parameters explored in this study.

6.2.4 Summary

This section offers an examination of the characteristics of Type-U spaces. It discusses number of unique properties which make them highly suitable for exploration via Spatial Mapping. The section also discusses some of the limitations, complexities and open questions regarding Spatial Mapping of Type-U, as well as CA algorithms in general. It serves a dual function, both as a broad introductory review of Type-U algorithms, as well as a case study for CA exploration through Spatial Mapping.

6.3. Phenotypic Properties

Type-U algorithms present a number of unique characteristics and notable differences from many established CA algorithms. Perhaps the most notable of which is the fact that new state values cannot be created by the transition function. Instead, particular colour values can only persist by shifting between cells as they get selected, or otherwise become removed from the system. The following section offers an initial review of Type-U phenotypic properties, derived from preliminary observations and explorations of various spaces. These are mostly qualitative characterisations and are meant to reflect general properties of Type-U, in line with the scope of this expository study, and more broadly, with the aims of this thesis. Once again, significant further research is needed in order to obtain a more comprehensive understanding of Type-U phenotypic behaviour.

6.3.1 Heterogeneity

The previous section discussed the composition and diversity of various distinctive behaviours of Type-U algorithms, as they arise in their respective combinatorial spaces. However, composition and diversity can also be discussed in the context of individual algorithms. The range of phenomena which can arise from the dynamics of a given algorithm represents an important aspect of its potential value. In Type-U, this can be measured in terms of structural and colour diversity.

The strong correlation between structural formations and colour in Type-U makes it exceedingly difficult to consider them separately. While the

composition of colour in a currently running system can be evaluated via a histogram analysis, there are no precise equivalent heuristics to measure structural patterns. Moreover, the temporal aspect is of key significance, both in terms of evaluating patterns, and comparing between them. In some algorithms, discernible patterns may develop (and dissolve) in only a few dozen steps, while in others this process may take thousands of steps, or may never arise in the first place.

There are some genotypic markers which could help predict whether or not an algorithm will develop certain discernible behaviours. For example, the presence of the `div()` operator and the Extended Moore neighbourhood both appear to induce more erratic overall behaviour. While their presence may sometimes be correlated to volatility, presumably due to the presence of larger delta values, the degree to which this is the case requires further targeted studies.

6.3.2 Convergence

Convergence relates to the range of possible outcomes an algorithm may exhibit from different random configurations. More specifically, its tendency to converge towards a particular outcome. Highly convergent Type-U algorithms can be characterised as being prone to being taken over by the same particular pattern on multiple separate runs, while divergent algorithms are typically characterised by a more balanced dynamic between the various phenomena they exhibit, either throughout their evolution or towards their possible terminal states.

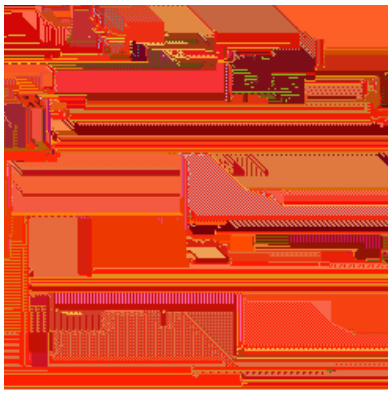


Figure 67.
TU59 typically converges towards the same behaviours, patterns and colours.

This tendency towards particular outcomes in Type-U algorithms can also be regarded as a bias that favours certain emergent patterns or behaviours over others. This should not be confused with deterministic behaviour in regards to configuration, as discussed in relation to the use of pseudorandom values earlier in this chapter. In that regard, all Type-U algorithms are explicitly deterministic in the sense that a given configuration will always evolve towards the exact same output on multiple different runs. Some convergent Type-U algorithms feature such a strong bias towards particular dynamics that certain patterns predictably "take over" the entire system in all, or almost all runs. Universally common properties of such patterns have not been identified at this time. However, this dynamic is speculated to be akin to a process of natural selection, whereby patterns which best fit their environment — the particular dynamic imposed by the algorithm — are the ones who ultimately get to persist.

While there is, of course, some degree of co-dependence between heterogeneity and convergence, they are not necessarily coupled. A Type-U algorithm may, throughout its evolution, feature a relatively diverse set of structures and still present a tendency towards a particular terminal state. Likewise, an algorithm may only be capable of supporting a small set of discernible structures overall, yet still feature a balanced dynamic that does not demonstrate a clear tendency towards any particular terminal outcome.

Figure 68.
Diverse colour regions in TU8.

6.3.3 Levels of Organisation

Structural patterns in Type-U algorithms can be thought of as embodying a symbiotic relationship between different colours within localised regions. The particular composition and arrangement of any given group of colours, combined with the unique dynamics of the underlying transition function is what allows them to maintain their relationships to one another and persist over time. This may involve either moving or static regions of varying sizes and complexities, as well as fluctuating patterns that periodically change their own

structure. It is important to emphasise that all such structures are strictly self-assembled; they emerge through a process of natural selection, whereby structures that cannot maintain a stable internal structure or a sustainable relationship with their environment are forced to make room for ones that can.

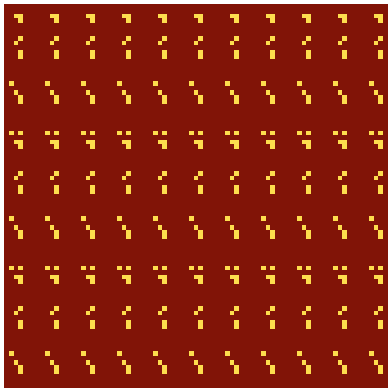


Figure 69.
First order structures in TU37.

Type-U dynamics generally appear to favour arrangements of high contrast; smooth and gradient patterns are uncommon, though this may be, at least in part, a result of the fully random configuration used throughout this study. Structural patterns themselves may feature varying degrees of diversity, with some clearly consisting of only two colours, while others may be formed as a collaboration of five or even more. Distinct structures may form almost instantaneously upon configuration, often as a swift reaction to the initial configuration of random colour values. Alternatively, some structures may arise at a much later stage, either as a result of a clash between previously unrelated formations, or as a result of the demise of a pattern which was holding them back.

Obtaining a clear definition of what a structure actually is in Type-U is not a trivial task. While distinct patterns such as gliders may sometimes emerge (see Figure 72), in most cases Type-U patterns exist along a continuum, mixing into one another, often forming non-distinct boundaries between them. Many patterns in Type-U emerge from highly complex interactions in their surrounding environment and simply cannot be isolated or even considered apart from it. Even when distinct boundaries do appear, this may or may not indicate a symbiotic relationship between patterns at a higher level of organisation.

Figure 70.
Second order structures in TU37.

Type-U algorithms have a distinct dynamic which differs from many other types of CA. This may necessitate a different way of thinking around structure formation and emergence in Type-U. For example, a glider in GOL consists of a persistent, moving arrangement of white cell states within a uniform, static black environment. However, in Type-U it may be the glider which is standing still within a moving environment, which itself is often far from uniform. Moreover, it may very well be that this glider actually emerged as a result of distant minor fluctuations which somehow staggered and permeated across the grid in a manner which cannot be recreated when attempting to isolate it, even within its immediate environment.

To that extent, adopting a holistic, qualitative approach, whereby all Type-U structures are considered as co-evolving alongside each other, may be preferable to a strict classification or taxonomic approach. This also aligns with how many other nonlinear systems are generally viewed, namely, not only as large sets of interacting components but as parts of a larger ecosystem that exhibits multiple levels of organisation and co-dependance. The self-assembled nature of Type-U structures and the reductive nature of their transition functions effectively force an implicit process of natural selection. This process differs from explicit evolutionary algorithms, such as those created by Sims [102], [118], [120], where selection is made by an external, uniform fitness function. Here, a structure's ability to both assemble and persist depends entirely on its ability to maintain stable internal and external interactions.

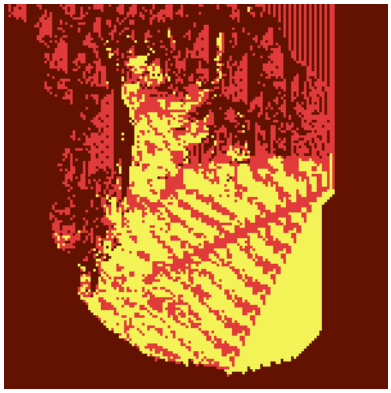


Figure 71.
Third-order structure in TU37.

Nevertheless, Type-U structures do consist of distinct patterns that exhibit distinct behaviours, which can be captured qualitatively. In regards to levels of organisation, they can be categorised as follows. If an individual cell in a CA is regarded as an indivisible universal module, then a first-order structure (exemplified in Figure 69) can be defined as a persistent local arrangement consisting of more than one cell. Notably, such structures may still span large regions of the grid, as it is not their overall size that defines them but the level of

intricacy of their underlying dynamic. A second-order structure (Figure 70) can be defined as a persistent local arrangement consisting of more than one first-order structure. These can be characterised by embodying a do-dependancy between their underlying structures. Second-order structures may exhibit a cyclic or shifting dynamic, stemming from interactions between their constituent parts. Likewise, a third-order structure (Figure 71) can be defined as a persistent arrangement of second-order structures, and so on.

6.4. Qualitative Characterisation

Stephen Wolfram's seminal work on Elementary Cellular Automata (ECA) [48], [73] stands as one of the few existing examples of a comprehensive study of a combinatorial space of CA algorithms. Another notable example is Christopher Langton's paper: Computation at the edge of Chaos [43]. Though the two studies are hard to compare in scope, both represent key precedences for this thesis. Among Wolfram's earlier studies of ECA, he provides the following classification of their phenotypic behaviour:

1. *Evolution leads to a homogeneous state*
2. *Evolution leads to a set of separated simple stable or periodic structures*
3. *Evolution leads to a chaotic pattern*
4. *Evolution leads to complex localised structures, sometimes long-lived*

[43]

As noted in Chapter 2, the above classes were originally presented as a "qualitative characterisation of cellular automaton behaviour". In the lack of subsequent well-established research on classification of CA, this characterisation has since obtained an unofficial status as a commonly accepted general classification of CA algorithms, yet it appears it was not originally formulated as such. Wolfram's characterisation can serve as a useful litmus test for examining preliminary observations of novel CA behaviour and is therefore suitable to serve as a foundational framework for characterising the findings of some of the exploratory studies presented in this chapter. It is both general enough to account for phenomena that can be observed in CA, yet still useful in defining principle differences between its four classes. These may even be used as a system for assigning value to CA algorithms, with class 4 attributed to algorithms of high potential value.

However, it may be useful to devise a qualitative characterisation that is better suited for Type-U phenotypic behaviour, which takes into account the preliminary findings presented in the previous sections. Such an adaptation would attempt to resolve a number of issues and inconsistencies that arise when applying Wolfram's characterisation to Type-U algorithms, which present notably different phenotypic properties from ECA. These are hereby presented, followed by a proposed adaptation of Wolfram's characterisation specifically tailored to Type-U.

Maximalist configuration

The fully random configuration of Type-U, used throughout this chapter, is an explicit attempt to maximise the range of observable emergent phenomena. As previously discussed, this approach addresses the highly subtractive nature of the algorithm. Once configured, the majority of Type-U systems progressively resolve towards a relatively ordered state. This contrasts with many well-established CA algorithms, where configurations are often minimal, such as altering the value of a single cell, most notably in ECA. In many cases, CA practitioners may avoid using random configurations to reduce the system's reliance on pseudorandom values. By doing so, the reliability of emergent

structures generated by consistent rules may be less influenced by randomness, potentially allowing their underlying emergent properties to be observed more clearly. However, it is important to note that the reasoning behind these choices is often context-dependent and may vary between different implementations and studies.

The assumption of a minimalist configuration function appears to be baked into Wolfram's class 3, as evolution towards chaotic behaviour is ranked relatively high. Commonly, CA that develop chaotic patterns from a minimalist configuration are indeed noteworthy, as pointed out by Wolfram. However, in a maximally configured CA such as Type-U, the presence of persistent chaotic patterns more likely indicates an inability to generate any discernible emergent patterns from the original random configuration. If anything, this property should therefore grant it a relatively low rank in the context of Type-U.

Homogeneity

A "homogeneous state" is characterised by class 1 as the lowest-value CA behaviour, as it signifies the evolution of a system towards trivial or "sterile" formations. While this characterisation is appropriate for additive algorithms such as ECA, RD and Abelian Sandpile, a homogeneous global state is not a trivial case in Type-U. It indicates that a single colour has taken over the entire system, a dynamic which may or may not constitute a behaviour of interest. In any case it should be noted that this constitutes the furthest possible state from the system's initial random configuration.

Continuous state

Class 2 characterises CA systems that settle to "a set of separated simple stable or periodic structures". This description does find some equivalence in Type-U. Stable periodic structures do often arise and are especially common in first and second order spaces. However, in regards to non-oscillating patterns, stable structures that traverse the grid at a constant rate should not be considered different from stable structures that are standing still. Constant motion in Type-U is often a result of a directional bias that is inherent in many algorithms. It is thus highly common for algorithms to feature homogeneous, constant motion towards a particular direction. In these cases, the emergence of meaningful patterns can only occur when some regions go against their environment's overall direction.

Type-U algorithms are, in-effect, continuous state CA. They feature fuzzy borders and sensitive symbiotic relationships between structures and colours. Thus "separated" is also not an ideal descriptor of their behaviour, as it again assumes the development of discrete patterns in a uniform environment and likely assumes a discrete state. For example, gliders and distinct moving patterns are comparatively rare in Type-U, though they do sometimes exist. More common are aggregated structures that develop into stable distinct formations, which then traverse the grid at a constant rate. As previously noted, it is typically the internal formation of structures that is of particular interest, rather than their traversal, which can be cancelled out in relation to their local environment.

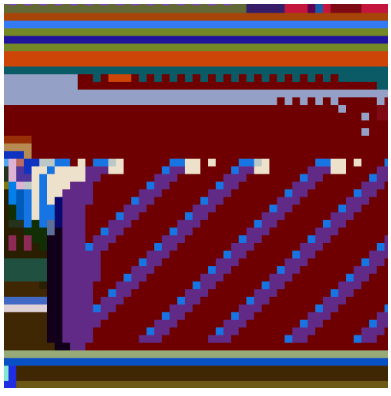


Figure 72.
Glider gun in TU8.

Between stability and chaos

Wolfram's classification effectively outlines a spectrum of CA behaviour, starting from uniform — to simple — to chaotic, and lastly — complex. While this is not explicitly stated, this characterisation implies that class 4 algorithms are of potentially higher value, as it is often assigned to well-established and notable algorithms such as GOL. However, it is clear that Wolfram's implicit assumption

of a minimal configuration and additive dynamics are incongruent with the inherent properties of Type-U. Not only do the configurations and dynamics of Type-U algorithms operate in opposition to this assumption, but the ranking of chaotic behaviour residing between simplicity and complexity simply does not align with observations of Type-U thus far.

The presence of ordered and chaotic patterns in Type-U is directly correlated with the length of the vectors created by the transition function. Algorithms that feature heavy use of division and high-value variables such as V24, appear to feature more chaotic behaviour overall. Similarly, algorithms that distinctly feature low-value variables and lower-impact operators, such as addition and subtraction, tend to exhibit simple, uniform, and static formations.

In any case, the emergence of complex structures does not appear to be correlated with either low or high-impact variables or operators. Instead, it more likely results from a delicate balance between chaotic and stable dynamics. Complex patterns seem to arise when the interplay between cells and cell states is potent enough to allow for the reorganisation of novel structures. Simultaneously, these interactions must be stable enough to prevent these structures from dissipating. Highly reactive, chaotic, and volatile interactions appear incapable of sustaining complex high-order structures, as these likely require some degree of aggregation in order to form and persist. On the other hand, highly rigid, sterile, and stagnant interactions appear to hinder the development of complex structures in the first place. Consequently, complexity in Type-U can more appropriately be characterised as arising from a delicate balance between simplicity and chaos.

The idea of class 4 residing between class 2 and class 3 has been previously put forward by Langton et al. [43, p. 32]. The concept of “edge of chaos” describes a delicate balance whereby emergent behaviour is neither too orderly nor too chaotic, but positioned on the brink of criticality. It proposes that solid and fluid are phases of matter that constitute “two fundamental universality classes of dynamical behaviour”, separated by a phase transition. According to Langton, it is within that transition region that the basic mechanisms for emergent behaviour resides.

While the initial alignment of Langton’s ideas with Type-U dynamics seems promising, it is important to note that applying Langton’s proposition to this study remains speculative at this stage. Moreover, it should be emphasised that this thesis does not engage with the profound implications that Langton’s and Wolfram’s research have for universal aspects of CA, computation, and the origins of life and the universe. Instead, it only advocates for open-ended exploration of CA and offers novel tools for doing so.

1. *Evolution leads to simple stable or periodic structures*
2. *Evolution leads to complex localised structures, sometimes long-lived*
3. *Evolution retains a chaotic pattern*

A proposed adaptation of Wolfram’s classification for Type-U is thus presented above, incorporating the following adjustments: removing the term “set of separated” due to its implicit assumption of additivity and discreteness; replacing “leads to a chaotic pattern” with “retains a chaotic pattern,” as the latter more accurately represents a trivial case in Type-U; eliminating Class 1, as it does not constitute a trivial case; and swapping the ranks of “chaotic” and “complex” to better reflect the observed characteristics of complexity in Type-U. These modifications refine the classification to align more closely with the emergent properties observed in Type-U systems.

Temporal Undecidability

The temporal dimension plays an important role in CA and it is worth noting that the characterisations discussed above leave certain temporal aspects unaddressed. Specifically, the number of time steps required for a system to run before it can be classified is never stated. Wolfram's terminology, particularly the phrase "leads to", keeps this temporal aspect somewhat vague, leaving it up to researchers to determine when exactly in its evolution a system is to be classified. In Type-U this is amplified by the fact that a single algorithm's behaviour can exhibit a dynamic process encompassing all three classes. For instance, it may start with a highly chaotic pattern, gradually evolve to form complex structures, which may then subside and dissipate, giving way to a simple or even uniform state.

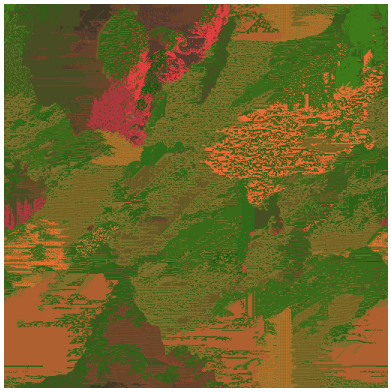


Figure 73.
TU57 is an example of a late stage phase transition in Type-U. It typically takes around 450 steps for it to resolve into its characteristic behaviour

A further challenge lies in the fact that some Type-U algorithms may remain in their initial chaotic state for a significant duration before undergoing a global phase transition to one of the other classes. Consequently, classifying these algorithms becomes a potentially undecidable task. The temporal evolution of Type-U systems therefor introduces an additional layer of complexity and variability, which should be taken into account when classifying their behaviour. More generally, the temporal aspect is particularly impactful in exploratory studies that are based on direct visual examination such as this one, in which late stage phenotypic behaviours are very likely to be missed.

This section discusses qualitative characterisations of Type-U phenotypic behaviour, drawing on the foundational classifications established by Stephen Wolfram. It discusses the relevance and limitations of Wolfram's four classes of CA behaviour when applied to Type-U algorithms, which differ significantly from ECA. The section proposes an adapted classification scheme that is better suited to capture the unique properties of Type-U. Based on Langton's work, this adaptation suggests that the emergence of complex structures in Type-U may arise as a phase transition between chaotic and ordered dynamics.

6.5. summary

This chapter introduces a new family of CA algorithms, called Type-U. It offers an examination of a range of Type-U algorithms, as well as the combinatorial spaces they occupy. The chapter discusses a number of their distinct genotypic and phenotypic properties and provides metrics for qualitative evaluation and categorisation. It demonstrates how the Spatial Mapping method can be used to conduct an in-depth exploratory study of novel algorithms and dynamics. Type-U algorithms posses a number of unique characteristics which make them particularly well suited for exploration using this method. These include their recursive formal definition, strong correlation between genotype and phenotype and their strong tendency to yield discernible behaviour.

This study primarily relies on manual explorations and visual examinations in Utomata Lab, suggesting that such an interactive approach can be useful for gaining insights into the properties and behaviours of novel CA algorithms. This preliminary study can thus serve as a foundation for more targeted future studies of Type-U algorithms, or as a framework for future exploratory studies of other CA algorithms.

The next chapter presents a curated collection of Type-U algorithms which feature noteworthy phenotypic properties. While these, of course, represent only a minuscule fraction of possible Type-U dynamics, they represent the unusually wide range of Type-U patterns and behaviours that have been observed over the course of this study.

7.Case Studies

This chapter presents a curated collection of Type-U algorithms and provides qualitative characterisations of their phenotypic properties. These algorithms were largely discovered through excursions into various Spatial Mappings, and were each selected for their unique properties. Together, they constitute a representative group of the range of Type-U behaviours observed in this study.

7.1. TU8

```
U(
  div(add(V8.g, V4.g),sub(V.r, V24.b)),
  sub(sub(V4.g, V.b),div(V24.g, V24.g))
)
```

TU8 is a third-order, class 2 Type-U. The expression features two layers of operators, which include two additions, three subtractions and two divisions. It contains references to all four neighbourhood symbols (`V, V4, V8` and `V24`), five of which refer to the green channel. Notably, the last operator in the algorithm, `div(V24.g,V24.g)` , always returns a value of 1.0. The algorithm's dynamics feature a rapid initial tendency to form isolated chambers, allowing an unusually diverse range of structures to develop in each one. Its observed types of phenomena include highly stagnant regions, compound moving imprinted formations, shape shifting formations and highly turbulent dynamics. Its tendency for encapsulation also allows sustained activity over large periods of time.

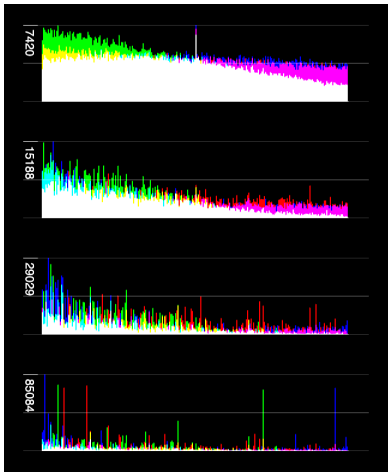


Figure 74.
Histogram Analyses of TU8 at 10, 100, 1000 and 10,000 steps.

Dynamics in TU8 consist of a pronounced east-bound directional bias, combined with a strong tendency for resilient, stagnant formations. Upon configuration, the initial reaction is rapid and consistent throughout the grid, with most of its characteristic structures emerging within a few dozen steps. This initial reaction also produces the vertical columns which, combined with its directional bias, divides the grid into distinct rectangular regions, with some forming rough, yet distinct borders. Since this subdivision occurs shortly after configuration, these chambers act as "incubators" that contain a high amount of colour diversity, still remaining from the initial random configuration. This results in unusual overall structural diversity, as unique dynamics develop in relative isolation inside each chamber. Likewise, colour diversity is relatively high, with an initial bias towards low green and blue values, which typically tapers off after a few hundred steps, eventually settling in to a more balanced spectrum on all three channels.

Some isolated chambers feature highly volatile and turbulent dynamics, with single cell sized strands of as many as ten distinct colours. Other chambers may feature much more stagnant and uniform behaviour, even to the extent of forming large, single colour, stagnant regions. Overall, there appears to be a positive correlation between local colour diversity and the east-bound directional bias. Stagnant and uniform colour regions also appear to be much more resilient than moving, turbulent and diverse structures. This applies to the vertical columns that initially divide the grid, thus accounting for encapsulation that persists over sustained periods of time.

Given enough time, some capsule walls do eventually give in to corrosion, allowing active and volatile patterns to seep out of their incubators. This

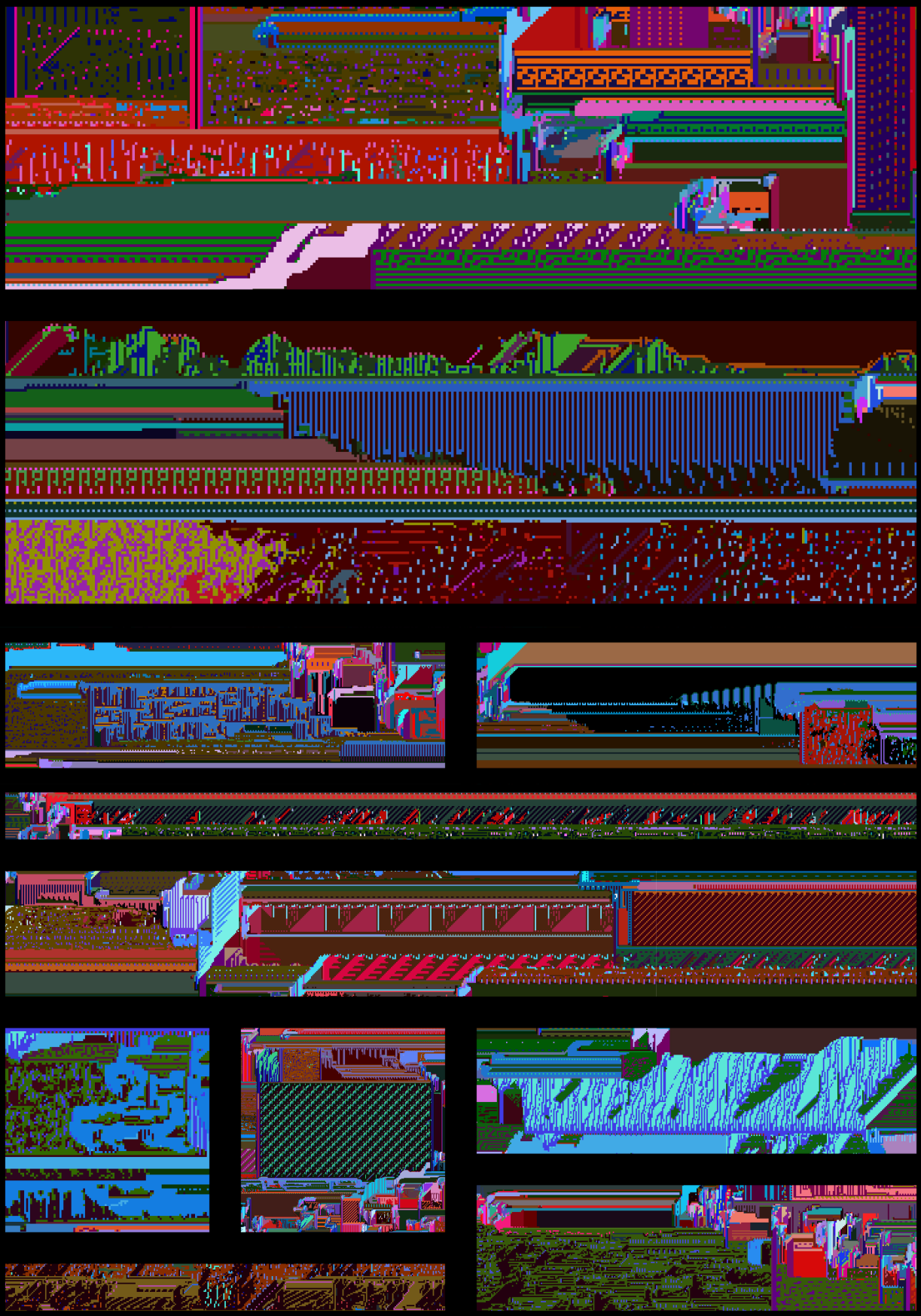


Figure 75.
Specimen collection from TU8.

introduces late stage novel formations into the system, as fully developed, previously unrelated patterns suddenly come into contact and form new, often surprisingly potent reactions. Late stage reactions can become highly corrosive, resulting in major shifts and secondary collapses, in turn, introducing high levels of activity into otherwise stagnant systems at very late stages of their development (upwards of 20,000 steps).

As a notable class 2 algorithm, it features a significant capacity for the emergence of higher-order structures. However, this does not appear to be an overwhelming tendency. Most characteristic active behaviours typically feature bounded, localised cell arrangements of 10-20 cells in two or three distinct colours. These often feature short cyclic behaviours in the form of elongated structures, comb-like formations, various grid formations and thick diagonal columns. Simple repetitive structures can often merge with each other to form chaotic rhythms that overlap, forming second and third-order structures. The directional bias can sometimes take form as a range of local velocities, resulting in higher-order formations and unique symbiotic relationships within the same encapsulated region.

In some cases, small regions appear capable of spawning unusually diverse colours, not present anywhere in their immediate environment. Since new colour values cannot be introduced in Type-U — only moved around — this implies that in particular circumstances the transition function may return unusually high X and Y delta values, pulling in colours from remote regions. This is presumably a result of the presence of the `div()` operator and the `V24` neighbourhood, both of which can potentially return arbitrarily high values. However, this is assumed to be a rare occurrence, since such a dynamic would not allow the apparent encapsulation, that is characteristic to this algorithm, to take place.

Intricate formations sometimes appear to be extruded from “nozzles”. These are small, centralised, high velocity regions that culminate in one or two static cells, thorough which heavy colour variations are rapidly expelled. Combined with the east-bound directional bias, this creates imprints of distinct repetitive formations, reminiscent of objects moving along an assembly line. In more rare cases, more than one such nozzle may be present, which may significantly compound the complexity of the resulting imprints.

Other types of generative and extruding formations have also been observed. “Water fall” formations have a similar capacity for imprinting formations, however these typically span a larger area than nozzles and tend to feature slower dynamics. Other generators are reminiscent of “glider guns” in GOL, extruding small and simple preconfigured arrangements of colours at regular intervals (see Figure 72). The various ways in which generative patters in TU8 may compound are staggering. In some instances, higher-order generators may form. They act as valves that open and close periodically, intertwining their output to produce exceedingly complex patterns.

Despite its high structural and colour diversity, behaviour in TU8 is relatively convergent, with a strong tendency for the same colour distribution and overall measure of directional bias on multiple separate runs (seen in figure 76). The algorithm’s stages of development are also quite predictable. The initial encapsulation occurs within the first 50-100 steps, structural variation usually peaks at around 600-800 steps, followed by a sustained gradual decay, caused by occasional erosions of capsule walls. Unless perturbed by leakage from collapsed neighbouring chambers, regions that have settled into a stable dynamic will persist indefinitely. This allows the algorithm to retain diverse, complex and potent activity over an indefinite timespan.

Figure 76.
64 separate runs of TU8 reveal overall convergent behaviour.
*(128*128, 2000 steps)*

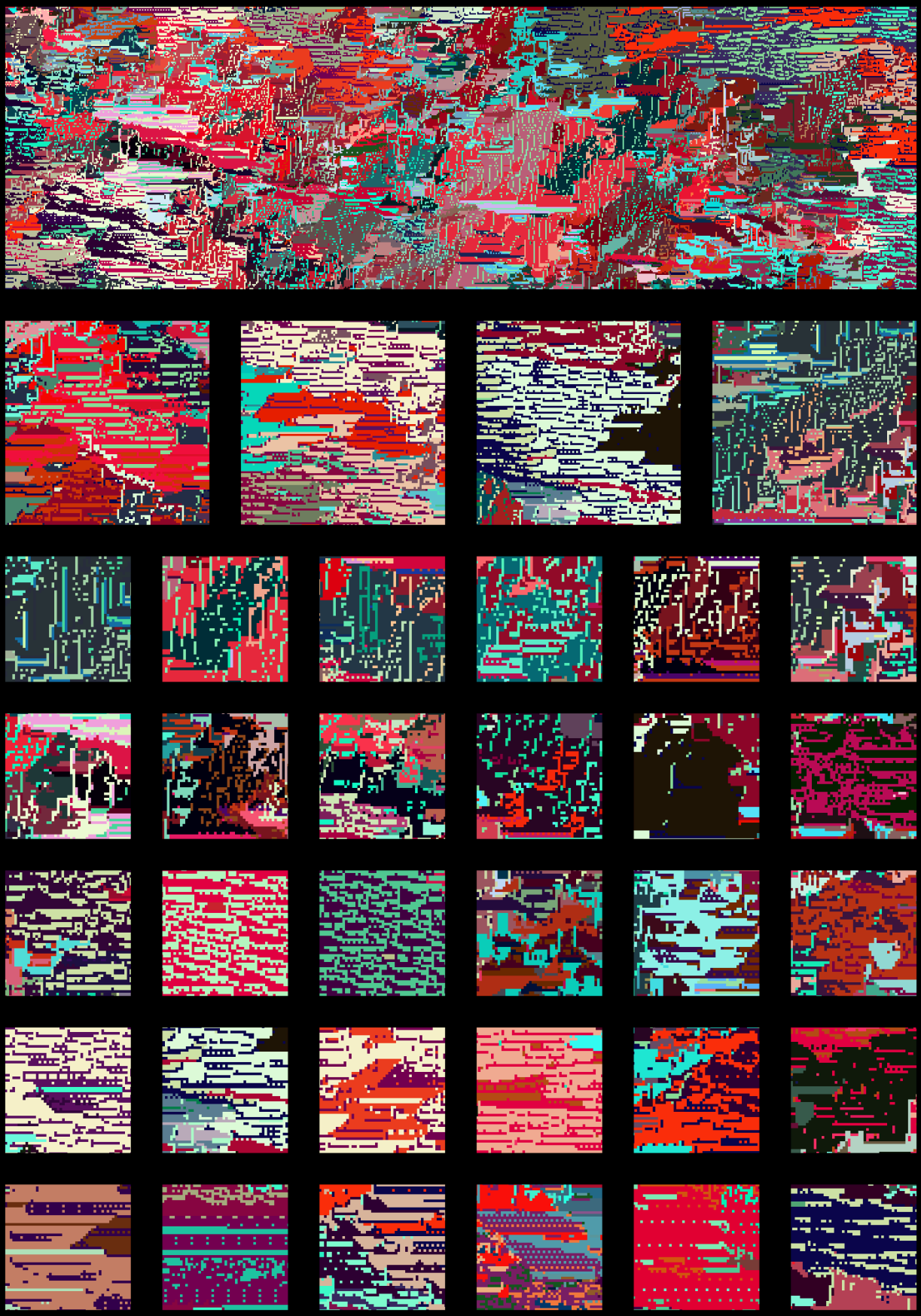


Figure 77.
Specimen collection from TU72.

7.2. TU72

```
U(
  sub(V9.b,V9.g),
  div(V4.g,V9.g)
)
```

TU72 is a second-order algorithm featuring overall uniform, yet highly distinct characteristic meandering patterns. While similar pattern types have been observed in other Type-U algorithms, an overwhelming presence of such types appears to be extremely rare. The algorithmic expression features subtraction and division, and a combination of V9 and V4 neighbourhoods, consisting of the green and blue channels.

Overall dynamics are uniform and stagnant, with a north-north-west directional bias, reminiscent of a “bubbling up” process. After configuration, the system settles into semi-stable formations within a few hundred steps, but continues to percolate indefinitely. The slow and steady flow of north-bound corrosive patterns appears to form at the edges of thin vertical lines, typically manifesting as dotted lines of single-cell formations. They travel through a substrate of mostly stagnant, less resilient formations, thus keeping the system in constant flux. The slow and sustained dynamic eventually reduces the overall range of unique formations, potentially leaving only one or two “triumphant” patterns. Exceptionally turbulent local dynamics are sometimes evident, though their corrosive properties are not necessarily proportional to their level of activity.

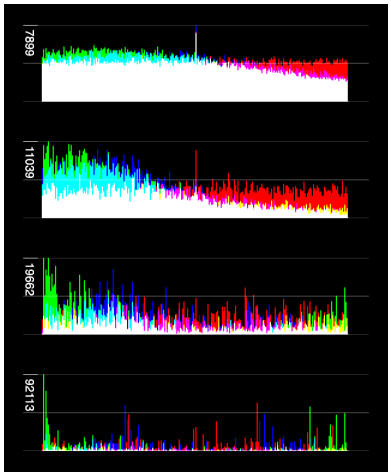


Figure 78.
Histogram Analyses of TU72 at 10, 100, 1000 and 10,000 steps.

This algorithm is highly heterogeneous, with all types of formations ultimately sharing a high degree of similarity in their dynamics, complexity, velocity and appearance. Emergent structures differ mostly in terms of their unique colour combinations, slight variations in scale, as well as their level of activity and corrosiveness. However, its many colour formations are separated by distinct boundaries and differentiation patterns. The vast majority of regions appear to consist of just two primary colours, with some instances featuring no more than five. However, there appear to be instances where a given colour pattern spans multiple neighbouring backdrop colours, indicating that there may be several layers of symbiotic relationships between colour patterns, allowing them to span multiple regions.

Limited by its homogeneous nature, this algorithm is also relatively convergent, with some distinct variations across separate runs, likely stemming from compounded variations of the random configuration pattern. Unlike TU8, which features significant encapsulation, the neighbouring relationships between different regions in TU72 can play a key role in pattern development. Any two patterns may react differently to one another when coming into direct contact. In some cases, a strict, rough boundary will form between them, allowing both to persist. In other cases, a third pattern may emerge as a potent reaction between the two. Alternatively, it may be that one pattern will simply absorb the other, while possibly enduring some level of mutation in doing so.

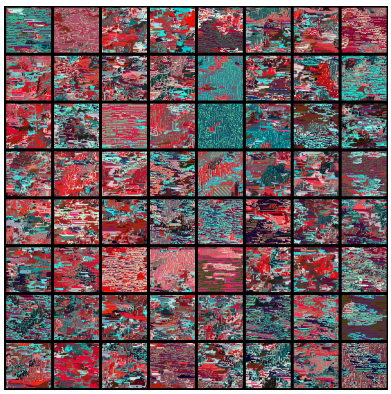


Figure 79.
64 separate runs of TU72.
*(128*128, 5000 steps)*

7.3. TU70

```
U(
  div(V.b, V.r),
  div(V24.r, V24.b)
)
```

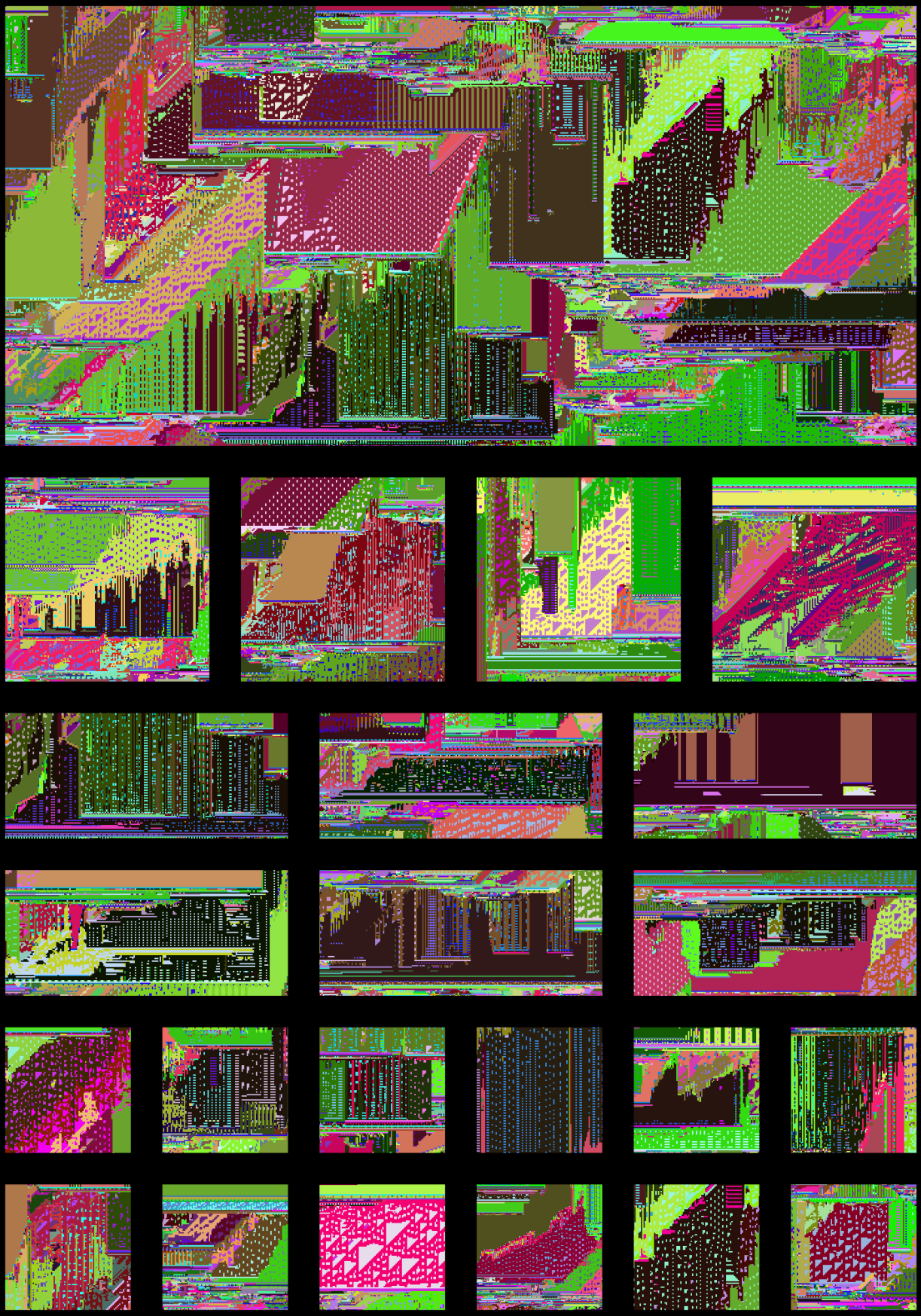


Figure 80.
Specimen collection from TU70.

TU70 features highly unusual phenotypic behaviour for a second-order Type-U algorithm. Its transition function features two division operators which dictate a particular dynamic between the red and blue channels: deltaX is the result of the blue over the red cell value (`V.b/V.r`) and the deltaY is the result of the Extended Moore neighbourhood reds over blues (`V24.r/V24.b`). This creates a pronounced, uniform presence of green values and a narrow, yet distinct, range of cyclic linear formations.

Dynamics consist of two distinct types of patterns which move along different directions. The first is a stagnant, stable arrangement of colours with rough, nondescript edges. It features a steady west-bound directional bias and seems to preserve much of its initially configured random colour values. As these patterns move horizontally, their top edges appear to “evaporate”, thus forming a second pattern type. It consists of highly corrosive, sparse vertical formations that move along a strong, non-uniform, north-bound directional bias. The colour diversity of these regions can be significant, apparently stemming from the retained randomness of their substrate. These pattern types appear to feature a range of turbulent dynamics that traverse the grid at various velocities. However, the development of complex interactions within these regions appears to be inhibited by their pronounced directional bias. Instead, they feature a considerable diversity of vertical, first-order cyclic formations. This includes triangular formations of various sizes and orientations, dashed and dotted lines, thin horizontal stripes, as well as thick diagonal patterns.

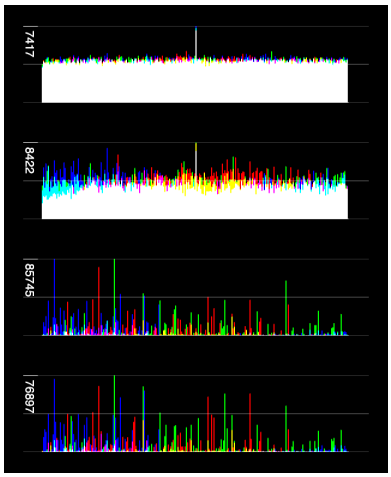


Figure 81.
Histogram Analyses of TU70 at 10, 100, 1000 and 10,000 steps.

Vertical formations are highly corrosive. They destroy everything in their path almost immediately upon contact. As a result, the stagnant, west-bound formations are relatively short lived, typically taking around 500-700 steps to completely disappear. At that point, all north-bound regions which feature different velocities begin to erode each other. Their reactions sometimes form static residual patterns that are more resilient to erosion. The system typically reaches a state of equilibrium within 2000-4000 steps, as all opposing vertical dynamics are resolved. This usually leaves a very limited set of simple, cyclic formations that traverse the grid along a uniform vertical velocity, marking a relatively early end to all potent activity in the system.

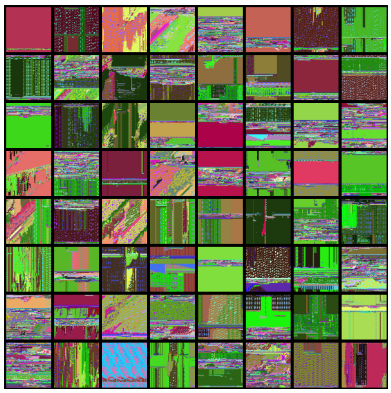


Figure 82.
16 separate runs of TU70.
*(128*128, 1000 steps)*

7.4. TU76

```
U(
  sub(
      sub(div(V.r, V.g),mlt(V.b, V8.g)),
      div(sub(V4.g, V4.b),add(V9.g, V8.b))
  ),
  sub(
      div(div(V8.g, V4.b),sub(V9.g, V4.a)),
      div(mlt(V.r, V9.r),add(V.r, V24.r))
  )
)
```

TU76 is a fourth-order algorithm consisting of two distinct, competing pattern types. Initial configuration triggers an immediate, strong differentiation between colour values. In only a few dozen steps, blue values are concentrated at the low end of the spectrum, while red and green values are concentrated at the high end. At this stage, the system is completely washed by a heterogeneous and turbulent, class 3, yellow-pink substrate with a mild east-bound directional bias. This pattern gradually gives rise to more ordered pink formations that may form large triangular regions and some second-order structures.

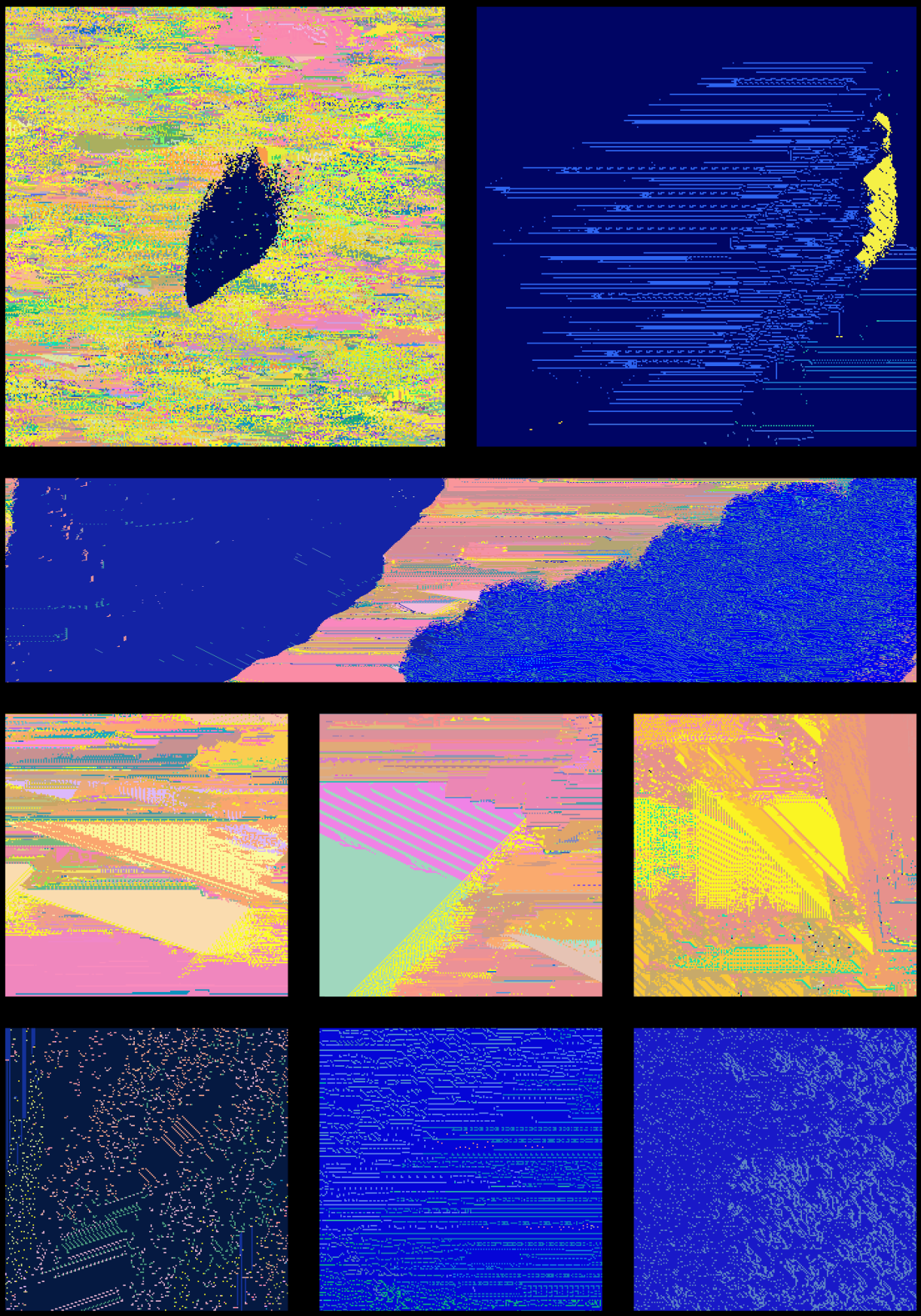


Figure 83.
Specimen collection from TU76.

The second pattern type consists of class 1, dark blue regions that emerge as a late stage phase transition which seems to occur in approximately 10% - 20% of runs (depicted in figure 84). They initially break out as small separate pockets within the uniform yellow-pink substrate. When they manage to persist through their initial window of opportunity at around 100 steps, these blue regions typically take over the entire system within the first two thousand steps.

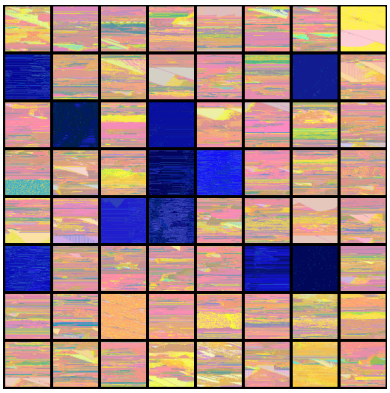


Figure 84.
16 separate runs of TU76.
*(256*256, 2000 steps)*

Late stage phase transitions such as this one are relatively rare among the many algorithms observed in this study. However, as previously noted, such patterns are very likely to have been overlooked in this exploratory study. Since it involves sifting through thousands of algorithms and selecting noteworthy behaviours based on visual differentiation, this methodology is heavily biased towards algorithms that instantly present unique behaviours.

Discounting its unique phase transition, this algorithm is highly convergent. Both of its dominant pattern types are relatively uniform, with some notable structural diversity apparent in each one. However, the particular interaction between the two types is nonetheless noteworthy. The manner in which yellow patterns ultimately dissolve to form sparse linear formations within the blue regions is highly reminiscent of sedimentary dynamics. Other exceptional distinct reactions can also occur, including pronounced vertical and diagonal regions, as well as distinct fluctuating patterns of sparse yellow formations, which manage to survive the initial phase transition. These fluctuating patterns may be linked to the presence of the div operator, though this has not been verified.

While TU76 exhibits relatively convergent and homogenous dynamics, It indicates a wider potential for unusual and surprising phenotypic behaviour in fourth-order Type-U. While fifth and higher order Spatial Mappings are exceedingly difficult to sift through, they are also likely to feature many unique behaviours that are not present in lower order mappings.

7.5. TU37

```
U(
  sub(mlt(V8.b, V8.b),sub(V8.g, V.b)),
  add(add(V4.r, V.r),sub(V.r, V24.b))
)
```

TU37 is a third-order Type-U which features a range of highly turbulent dynamic patterns, some of which resolve to form stable, organised structures at unusually late stages of development. The transition function refers to a range of neighbourhood types from the red and blue colour channels and features addition, subtraction and multiplication. The algorithm sustains high activity over long periods but, given enough time, is eventually overtaken by its most turbulent patterns.

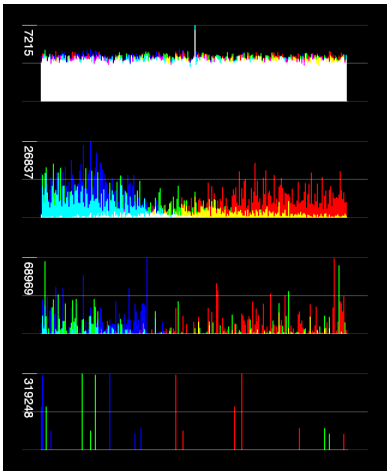


Figure 85.
Histogram Analyses of TU37 at 10, 100, 1000 and 10,000 steps.

Initial configuration becomes resolved within approximately 100 steps, consistently resulting in mostly low blue and green values, and high red values. This forms the algorithm's characteristically narrow colour palette. Directional bias is omnidirectional, as patterns propogate in all four primary directions. In addition, a number of distinct static formations are evident, as well as large regions of solid colour, though these are typically the first pattern types to be eradicated in sufficiently large grids.

Prominent vertical formations sometimes also emerge, going against the algorithm's typical turbulent dynamics. This unusual combination of

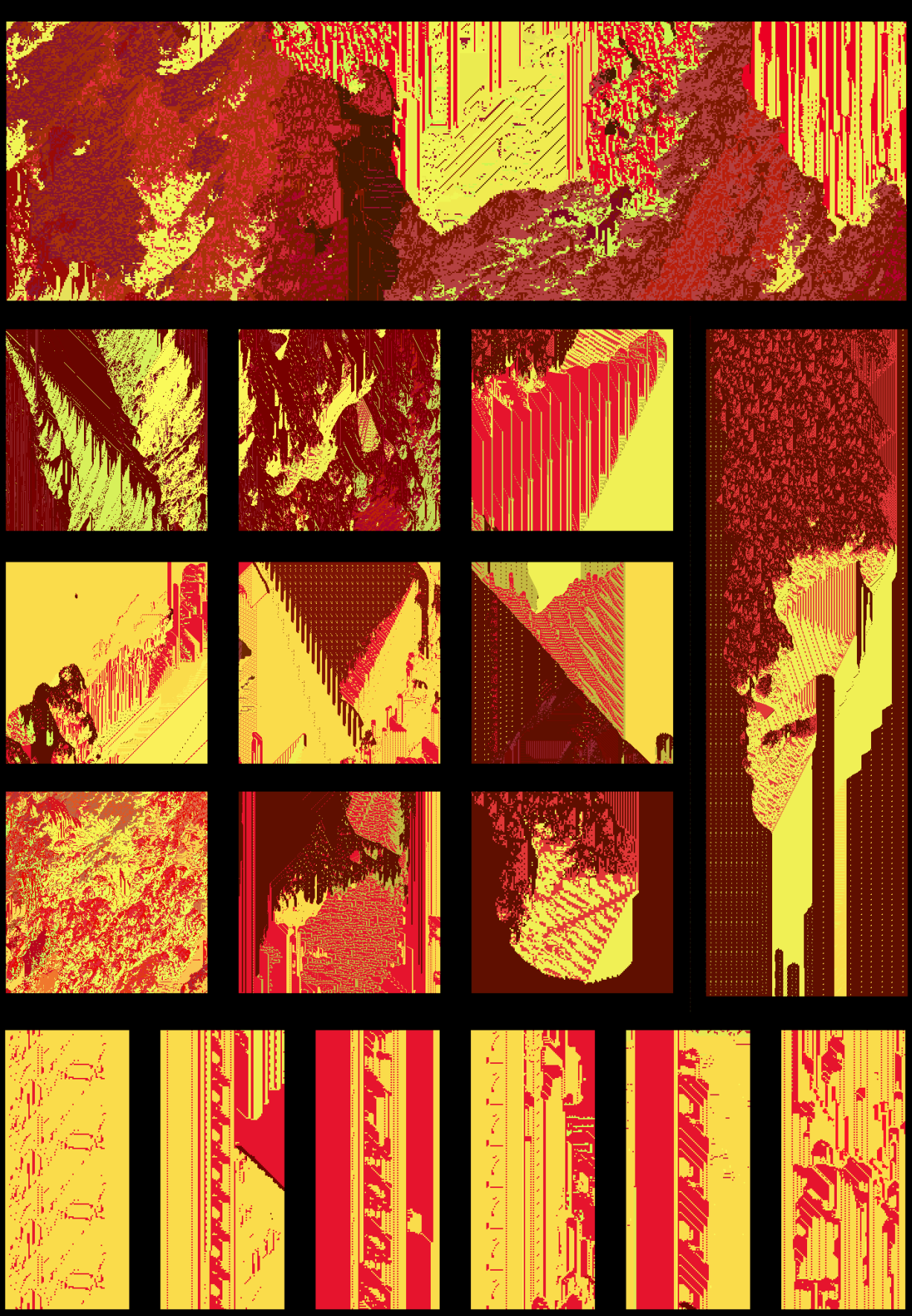


Figure 86.
Specimen collection from TU37.

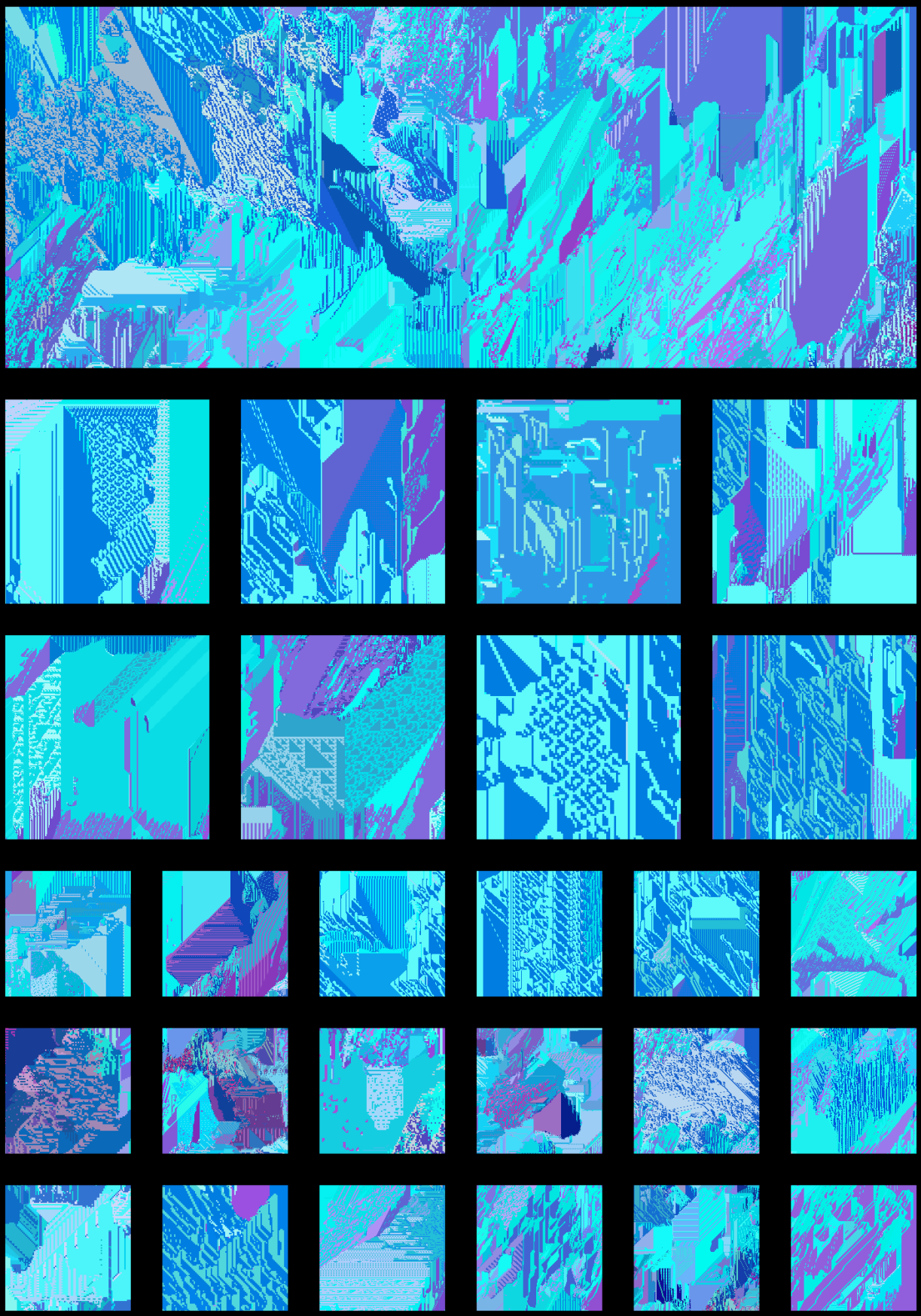


Figure 87.
Specimen collection from TU78.

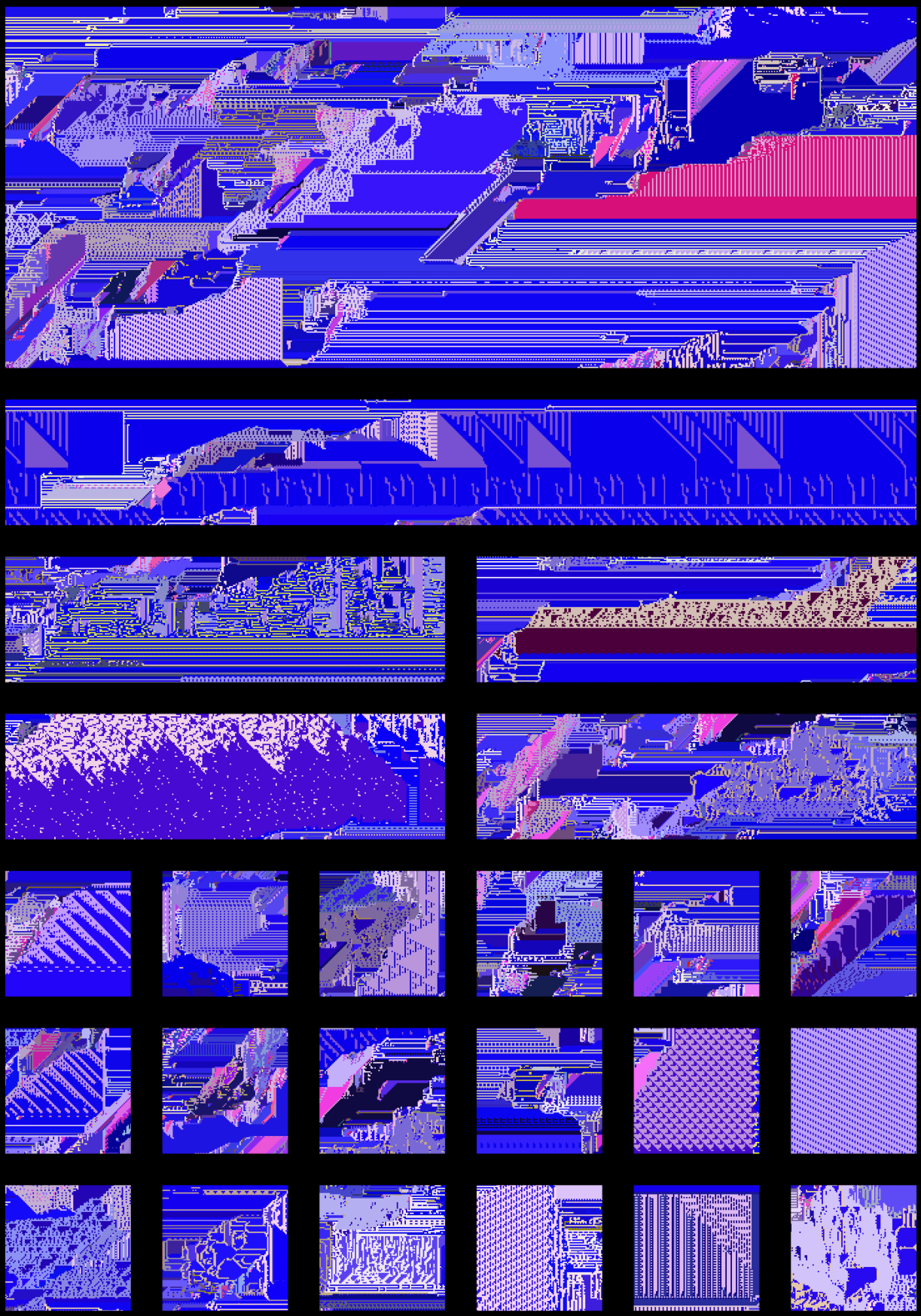


Figure 88.
Specimen collection from TU79.

omnidirectional movement, static formations and vertical towers sets it apart from other similar third-order algorithms. These three pattern types exhibit nearly equal corrosive power. Their constant clashes sustain the system's high potency over extended periods. This balanced power dynamic also makes this algorithm quite divergent, as almost any pattern may ultimately become "triumphant" on multiple separate runs (as seen in Figure 89).

Some collaborative patterns and symbiotic relationships have been observed to form higher-order class 2 structures, consisting of both turbulent and stable dynamics. These may take form as two equally powered formation types, with opposing directional biases that maintain a stable relationship with one another (see figure 71). Considering its highly constrained colour palette, TU37 supports a surprising variety of patterns and behaviours. All distinct formations appear to consist of just two colours, indicating that a wide range of strategies for self-organisation and persistence in Type-U may be possible with much more constrained configuration patterns than the fully random configuration used.

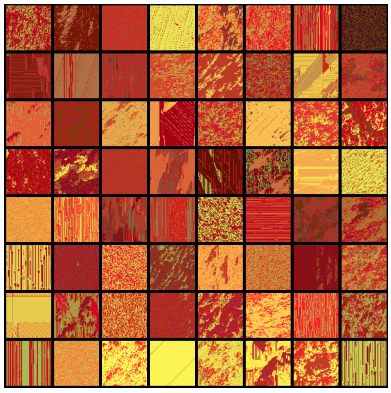


Figure 89.
16 separate runs of TU37.
*(128*128, 2000 steps)*

7.6. TU78

```
U(
  sub(sub(V8.r, V4.r), div(V24.r, V24.r)),
  add(sub(V.b, V8.b), div(V8.g, V.b))
)
```

TU78 is a third-order algorithm that features a unique combination of turbulent dynamics, periodic static formations and large regions of solid colour. Similar to TU37, its directional biases are relatively balanced. Cyclic and solid regions are typically static and most active patterns are typically north-west bound. Some linear formations feature a unique "trickle-down" effect, akin to dropping sand dynamics. Colour diversity is overall quite minimal, with most distinct patterns consisting of just two colours.

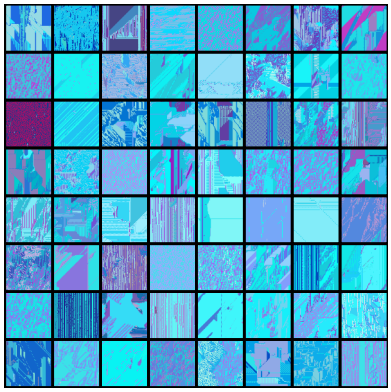


Figure 90.
16 separate runs of TU78.
*(128*128, 5000 steps)*

TU78 features unusual support for very large solid masses, intricate diagonal dotted patterning and long vertical formations. Unique branching formations and second-order grid textures are also sometimes evident, as well as distinctly unique triangular patterns with rounded corners. Distinct fractal patterns are also evident in the form of Sierpinski triangles. Considering its constrained colour palette, TU78 can be considered highly heterogeneous and divergent. Its numerous pattern types present a surprisingly balanced power dynamic, in which any one pattern may suddenly begin to thrive. This also contributes to sustained long term activity.

7.7. TU79

```
U(
  add(div(V8.g, V4.b),sub(V.g, V.r)),
  mlt(div(V24.g, V4.r),sub(V.r, V.g))
)
```

TU79 presents distinct similarity to structures found in TU78, though it is notably less temperamental. It features a strong west-bound directional bias, whereby almost everything traverses the grid at a constant rate. Localised north-bound regions of turbulent and solid colour formations act as the prime instigators of change in this algorithm. Upon configuration, the grid rapidly forms most of its stagnant patterns. These are characterised by long horizontal

white strips, forming over a dominant blue backdrop. Some colour diversity is retained in distinct localised “pockets”, alongside a range of textured grid patterns and diagonal solid regions, which sometimes contain vertical columns. Sparse Sierpinski-like fractal formations, as well as a range of imperfect semi-fractal triangular formations are also sometimes evident.

7.8. TU68

```
U(
  sub(V24.b,V24.g),
  sub(V24.b,V24.r)
)
```

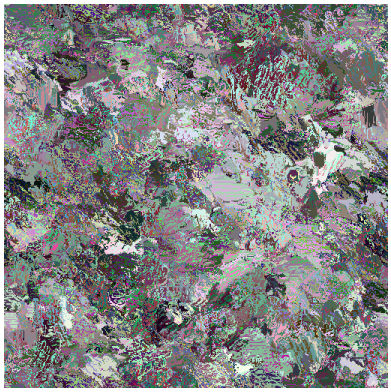


Figure 91.
TU68.

TU68 is an extremely turbulent second-order algorithm that features a monolithic algorithmic expression that simply subtracts Extended Moore neighbourhoods of red, green and blue channels. As previously noted, there are strong indications for a correlation between the Extended Moore neighbourhood and the presence of turbulent dynamics such as those present here. Higher delta X and Y values can result in a larger range of state values that are less likely to be resolved. Such patterns are mostly common in third and fourth-order Type-U, but are extremely rare in second-order algorithms. The fact that this algorithm consists of only V24 variables may support the above claim, though further testing is needed before this can be determined with confidence. While it is extremely homogeneous, this algorithm appears quite divergent; its numerous, yet almost identical, turbulent patterns present a balanced power dynamic, with no clear victor on multiple separate runs (as seen in Figure 92).

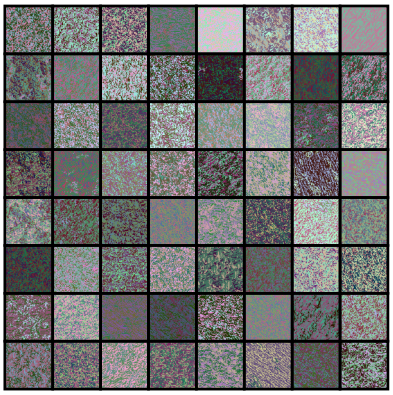


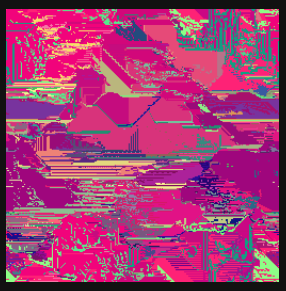
Figure 92.
16 separate runs of TU68.
*(128*128, 2000 steps)*

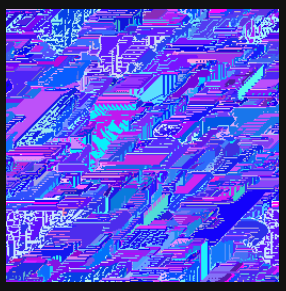
7.9. Summary

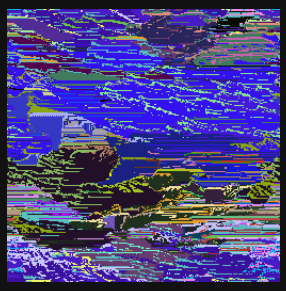
This chapter offers initial qualitative characterisations of a small set of Type-U algorithms which present notable phenotypic traits. Figure 93 showcases a wider selection of algorithms which have also been examined and considered for this study. The selection criteria for these algorithms and characterisations of their properties are highly subjective, representing the author’s aesthetic preferences and interests. However, this inherent subjectivity should not be regarded as a drawback; it firmly aligns with this project’s aim to explore and study algorithms which may be overlooked by more analytical methodologies.

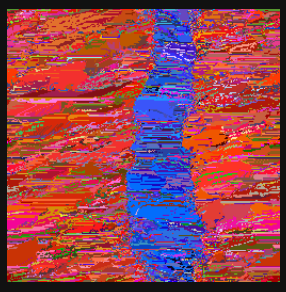
These case studies aim to showcase prominent behaviours that have been observed in Type-U systems over the course of this study. While each algorithm presents a unique blend of phenotypic traits, each such trait is likely also apparent in countless other algorithms, especially ones of the same order. Nevertheless, the unique balance of traits embedded in each specimen is what allows some of them to present highly rare and distinct dynamics. The most minute genotypic variation may result in the most minute change to an algorithm’s underlying dynamics. In turn, this could mean the difference between a thriving ecosystem and a barren wasteland.

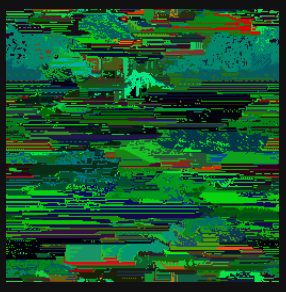

NU1

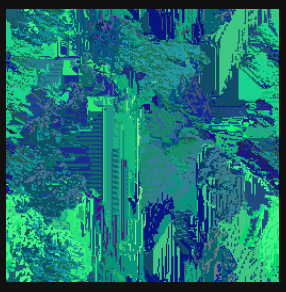

TU1

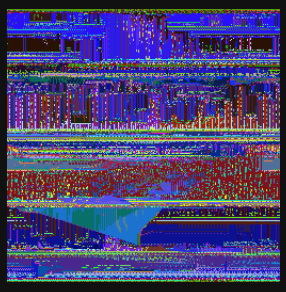

TU2

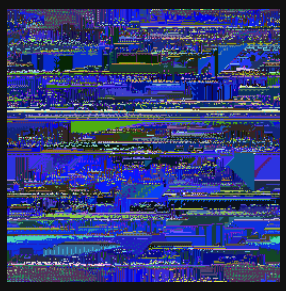

TU3

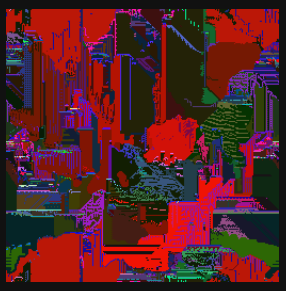

NU4

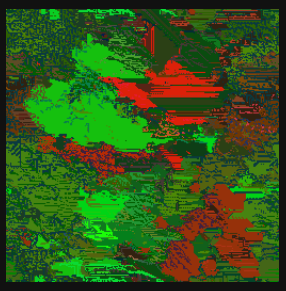

TU6

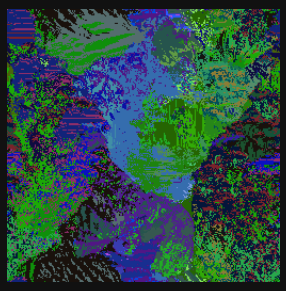

TU9

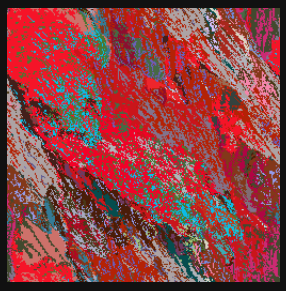

TU12

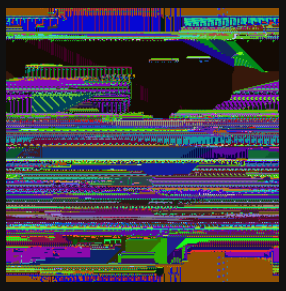

TU12A

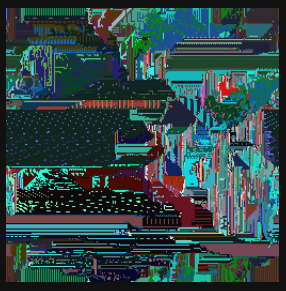

TU13

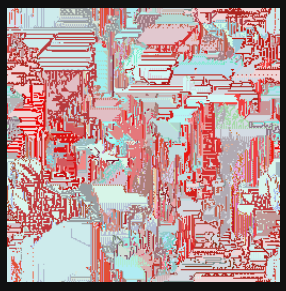

TU16

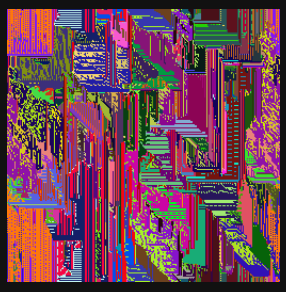

TU18

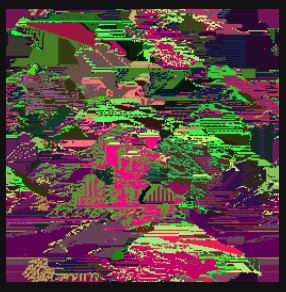

TU20

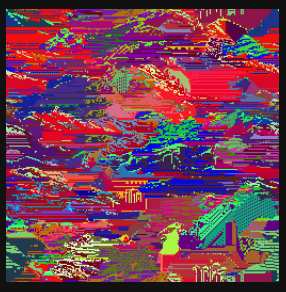

TU23


TU25


TU26

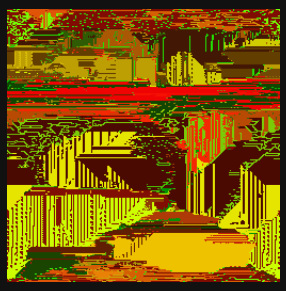

TU31

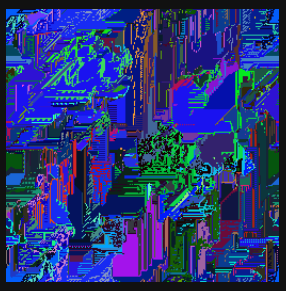

TU33

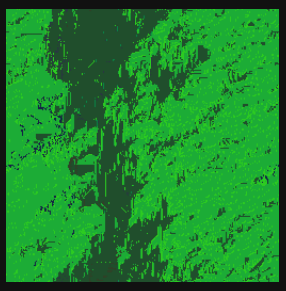

TU35

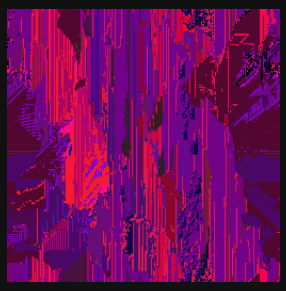

TU38

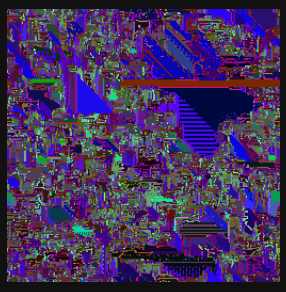
Figure 93.
Selected specimens from Utomata Lib.

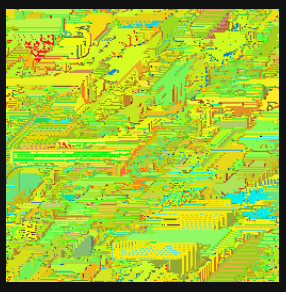

TU40

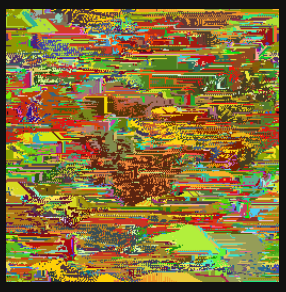

TU41

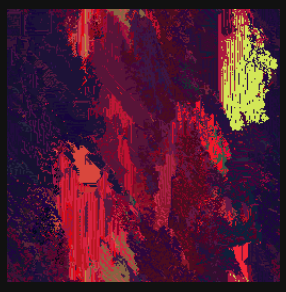

TU42

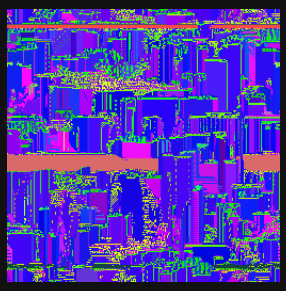

TU44

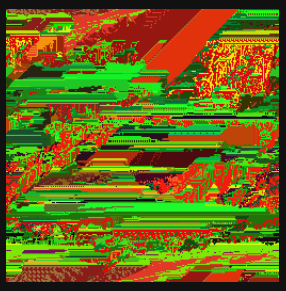

TU45

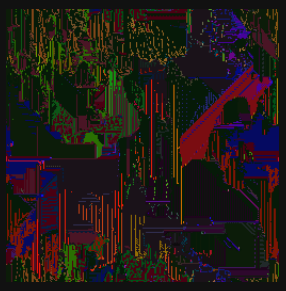

TU46

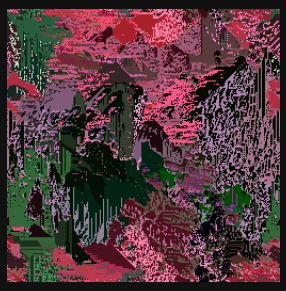

TU47

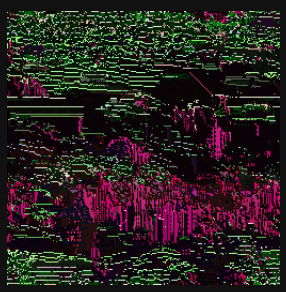

TU48

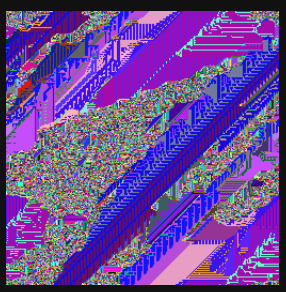

TU49

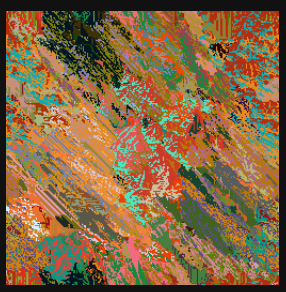

TU52

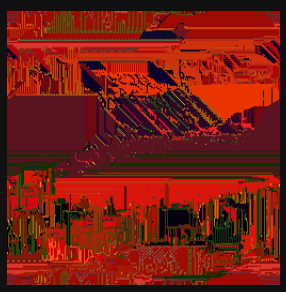

TU53

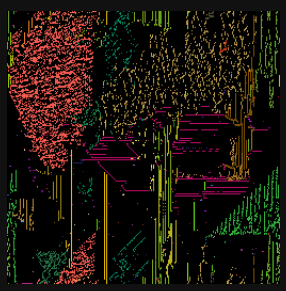

TU55

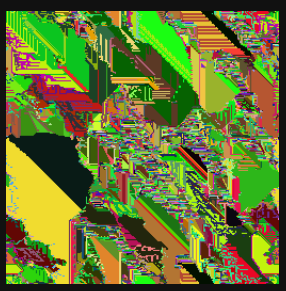

TU58

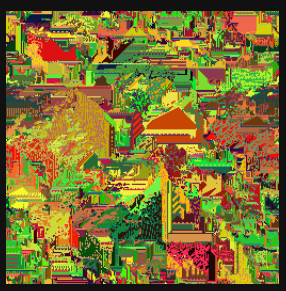

TU61

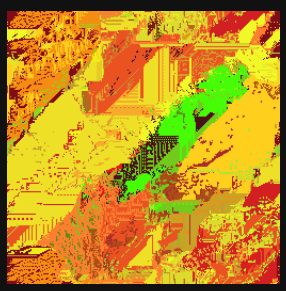

TU62


TU63


TU64


TU66


TU69


TU71

8.Discussion

8.1. Overview

This thesis presented a set of frameworks, tools and methods specifically designed for open-ended exploration of CA algorithms, employing a non-instrumentalist approach to exploration in order to expand the scope of inquiry. The effectiveness of the tools presented has been demonstrated through case studies involving both established and novel algorithms. Qualitative evaluations of novel findings demonstrated how this approach can also be used to interpret findings, while minimising reliance on representational and scientific concepts.

Utomata serves as a foundational framework which allows implementation of an exceptionally wide range of algorithms, as demonstrated through implementations of three well-established algorithms: GOL, ECA and RD. Further, it was used to conduct a low-level exploratory study of a new family of algorithms, called Type-C. The conception of the first Type-C algorithm, as well as the iterative process of extracting its unique algorithmic variants were made possible by Utomata's non-analytical approach to programming. This study demonstrated how direct manipulation of algorithmic expressions can facilitate the discovery of novel CA. This study also demonstrated a method for effective cultivation and examination of novel emergent structures, which did not rely on obtaining an analytical understanding of their underlying dynamics and avoided casing them as instruments or derivative objects.

Utomata can also be used as a foundational framework for high-level exploration methods. This was demonstrated through the introduction of the Spatial Mapping method, accompanied by its online software implementation — Utomata Lab. This method effectively formulates a mapping from a high dimensional algorithmic expression, written in Utomata's functional syntax, onto a large two-dimensional field. Spatial Mapping not only makes it possible to visualise and interact with an exceptionally wide range of algorithmic variations in real-time, but also aids in identifying patterns and relationships within the combinatorial spaces they occupy. The Spatial Mappings of GOL, ECA and RD showcased this method's ability to situate existing algorithms among their numerous possible variants. The notable limitations of this method were discussed through some of the unique properties of each of these three case studies, as well as the prospect of fine tuning the mapping parameters as a way to isolate potential findings of interest.

Lastly, a comprehensive exploratory study of a new CA family, called Type-U, was conducted. The unique genotypic and phenotypic properties of Type-U algorithms were examined, as well as some of the properties of their encompassing combinatorial spaces. These properties were further explored by examining a curated collection of Type-U algorithms which presented noteworthy phenotypic behaviour.

This chapter interprets and contextualises key findings from this research project, highlighting their contributions and discussing their limitations. It offers reflections on the project's underlying philosophical view and evaluates the tools and methods presented in this thesis towards accomplishing its research aim. The chapter also evaluates the implications of these results in regards to open-ended exploration of the broader class of emergent virtual structures, and suggests avenues for future research.

8.2. Key Contributions

Utomata Framework

As a novel computational framework for CA exploration, Utomata offers unique features that streamline the process of implementing and running CA algorithms. These differentiate it from general purpose graphics programming environments, such as P5js or Openframeworks. They include built-in neighbourhood variables, absolute and relative cell state retrieval, dedicated configuration and transition functions, support for high dimensional state, bounds management, custom operators, specialised randomisation features, parallelisation of multiple concurrent algorithms, real-time input, interaction and import-export functionality. Utomata also features built-in functionality which generalises the field parameterisation technique, allowing concurrent observation of numerous parametric variations of any algorithm.

Utomata's demonstrated ability to implement a wide range of real-valued CA also differentiates it from specialised CA environments such as Ready, Golly or VulcanAutomata, each of which specialises in a subdomain of CA. The open source, hardware accelerated Javascript implementation of Utomata is also unique in the landscape of CA software, providing a robust environment for running, storing and sharing CA online. The utomata.js library [201] is uniquely suited for embedding CA into interactive web applications, thus supporting a range of creative applications. Moreover, It can serve as an underlying rendering engine and procedural content generator for more advanced web based applications such as Utomata Lab.

Utomata's custom functional syntax constitutes a standardised method for implementing a wide range of CA algorithms. Its particular use of normalised state vectors and custom unary and binary operators, including boolean operators, provides a unified approach for implementing widely different CA algorithms. Moreover, unlike a general purpose programming language, it allows interchangeability of parameters, operators, subsections of algorithms, as well as entire algorithms. This makes it highly suitable for exploratory studies of CA that are guided by intuition, improvisation and aesthetics. As such it embodies the non-analytical approach to programming, advocated by this thesis.

This syntax also constitutes a highly minimalist form of algorithmic expression. This is most notable in The GOL algorithm, which consists of only a few characters. This algorithm differs substantially from other succinct expressions of GOL such as the Life-Like notation [70], [71] in which the underlying assumption of its structure — the number of live neighbours for births, spawns and deaths — is hard coded. In contrast, Utomata's GOL implementation constitutes a complete algorithm, which can easily be adapted to use continuous and vector states, be appended with any additional functionality, or be incorporated in its entirety as a sub branch of any other algorithm.

Lastly, a significant advantage of using a functional syntax lies in its compatibility with high-level, or meta-programming techniques, such as genetic programming and Spatial Mapping. The minimal design, recursive nature and strict topology of functional expressions all offer benefits for procedural generation of algorithms. The robust adaptation of a functional programming paradigm to CA exploration and research thus stands as a novel contribution made by this thesis.

Spatial Mapping

Spatial Mapping is a new, high-level exploration method that builds upon Utomata to produce a space consisting of all possible variations of any given algorithm, using a highly customisable set of parameters. Along with its accompanying software implementation, Utomata Lab, this meta-programming technique allows interactive exploration of numerous algorithmic variations of CA algorithms.

This method is based on a unique isomorphic mapping between a high-dimensional Utomata functional expression and a large two-dimensional field. The mapping defines a mixed radix system that encodes the input algorithm and a set of input symbols onto a coordinate system, allowing every unique permutation of the algorithm to be assigned a unique X and Y coordinate. Special care has been taken to enhance the intuitiveness of this method by converting these coordinates back into a decimal system, as well as to creating an inverse mapping, which is used to locate the input algorithm within its encompassing space. Different mapping parameters and spatial distribution methods can be applied to fine tune Spatial Mappings in order to accommodate different scopes of exploration, different algorithms and different research requirements.

Study of Type-C

An exploratory study of a new family of algorithms, called Type-C, demonstrates the use of a non-analytical programming approach in Utomata. The first Type-C algorithm was devised as an adaptation of a well-established algorithm, Abelian Sandpile, to feature a continuous state. Subsequent experiments yielded a range of unique behaviours through an iterative process, involving algebraic reduction, field parameterisation and real-time interaction. This study yielded a number of novel CA algorithms which feature a surprising range of unique phenotypic properties, alluding to the prospect of highly potent lineages of algorithms, concentrated as “pockets” of noteworthy behaviours within larger, more barren, spaces.

In particular, the variant *Red Nose Hexagliders* features persistent and stable glider formations which are semi-resilient to collisions. This unique feature brings rise to higher-order structures which form a diverse “ecosystem” of emergent phenomena. These were categorised in a preliminary taxonomical study, shown in figure 25, as well as documented in the short nature documentary film Vital Signs [203]. This study further demonstrates the effectiveness of Utomata as a tool for open-ended exploration and examination of CA, as well as that of the underlying approach to exploration, advocated by this thesis.

This approach considers emergent virtual structures, such as those found in Red Nose Hexagliders, as instances of observable phenomena, which can be cultivated and studied — irrespective of function or metaphor. Notably, these studies demonstrate how exploratory CA research can be conducted without obtaining a firm analytical understanding of the exact dynamics of the underlying algorithms — just as a firm understanding of biogenetics is not strictly required for farming. In that sense, an algorithmic expression that defines any given CA behaviour can be likened to its genotype, and any change to that algorithm can be likened to a mutation, which may or may not give rise to other behaviours of interest.

Study of Type-U

This study introduced a new family of CA algorithms that are characterised by a particular transition dynamic: the new state of each cell is set to be the current

state of any cell in the grid. Algorithmic expressions belonging to this family can be formulated in Utomata as functional expressions whose root is the `U(dx, dy)` function. Using the dx and dy parameters, the U function retrieves the current state of any cell in relative (discrete) coordinates. Spatial Mapping was then used to construct a complete combinatorial space of all such expressions. These were expanded recursively to form higher-order Type-U space which consist of exponentially more algorithms.

Type-U can technically be categorised as continuous, vector state, outer-totalistic CA. However, they feature a number of unique properties and characteristics. The use of multiple totalistic neighbourhoods in conjunction with the U function leads to a particular dynamic, whereby new cell states may be drawn from anywhere in the grid. This embodies a unique and novel relationship between transition, neighbourhood and state. Since Type-U transition functions are strictly incapable of introducing new state values into the system, this dynamic is inherently reductive. The result can be characterised as a cross between a CA and a sorting algorithm.

The use of Spatial Mapping to facilitate exploration of the vast space of possible Type-U algorithms across different orders constitutes a novel methodology for an exploratory study of this kind. Perhaps the closest well-known example is Wolfram's study of ECA [48], which also involves formulating and exploring a combinatorial space of permutations of CA algorithms. However, the significant size difference between these respective spaces is staggering. While all 256 variants of ECA can be implemented and rigorously studied using brute force techniques, the combinatorial spaces of Type-U algorithms would literally take centuries to compute exhaustively.

Therefore, this study adopted a more cautious approach, akin to naturalism in the sciences, where phenomena were studied with the understanding that only a minuscule part is directly observable. This approach focused on general properties of algorithms, their similarities, differences, and their unique phenotypic traits that differentiate them from other well-established CA. Observations of Type-U dynamics indeed revealed overwhelming diversity of emergent behaviours, with a surprising number of algorithms supporting persistent, higher-order structures. Examinations of these algorithms, according to their specific properties and characteristic behaviours potentially constitute a framework for qualitative evaluation of phenotypic behaviour in CA, which can be adapted to other algorithms and models.

8.3. Limitations

Computational Framework

While Utomata's non-analytical approach to programming potentially makes it more accessible to creative practitioners, it is nonetheless a programming environment. As such, creative practitioners who wish to conduct low-level explorations of CA may face a steep learning curve in mastering its use. The framework's custom functional syntax is a high-level concept which may present a challenge, even for experienced programmers who are unfamiliar with functional programming. This minimalist form of expression can also make algorithms potentially less legible in analytical terms, compared to a general purpose language. [1] Moreover, this syntax is not standardised across other CA frameworks and environments, nor is there currently an open collection of established CA algorithms written for Utomata.

This points to a need for comprehensive documentation, tutorials, examples and implementations of Utomata in other graphics programming environments.

***1** Though it should be emphasised that this feature is by design. It aims to encourage open-ended exploration and experimentation by minimising potential biases involved in efforts to make sense of a given algorithm.*

At the time of writing, efforts to create a library of Utomata algorithms and examples, as well as official documentation and tutorials are still in progress. Additionally, alternative implementations of Utomata have also been created for Processing, Openframeworks, TouchDesigner and Blender3D, yet these have not yet been made public.

Utomata's explicit focus on CA also makes it considerably limited for other, more general tasks, such as data processing or drawing and interacting with shapes and splines. That said, since Utomata's transition function is effectively a fragment shader program, it can still be used for many applications beyond CA.

Another notable aspect of Utomata is its reliance on GPU computing. While this provides exceptional computing power it can make algorithms more difficult to debug. The framework lacks robust debugging tools commonly found in conventional programming environments, such as console logs and break points. While the Utomata IDE and Utomata Lab do feature rudimentary tools for debugging and analysis, such as error messages and histogram analyses, significant untapped potential for such features remains. Additionally, the use of continuous normalised state values may introduce floating point error, which has been observed in some Type-C variants. This can often manifest as a breaking of symmetry when configured with a symmetric pattern. While floating point errors are not unique to GPU computation, they can be harder to identify and mitigate, compared to a CPU based implementation.

Non-Instrumentalist approach

This thesis is founded upon an unconventional approach to exploration of virtual phenomena. It seeks to uncover novel findings by employing a non-instrumentalist approach to computational modelling. This serves as the underlying foundation for the development of all software tools, methodologies and studies presented. However, this approach can easily overshadow efforts to obtain an analytical understanding of CA dynamics. It introduces significant subjectivity which may limit its appeal in educational and traditional research contexts, especially in contexts where analytical insight, precision and reproducibility are prioritised.

Many of the proposed research methodologies primarily rely on aesthetic evaluation and acquired intuition. It may therefore be challenging to establish objective criteria for evaluating certain CA algorithms. Moreover, generalising these findings to other algorithms, models or domains may not necessarily be applicable. Lastly, these techniques typically require extensive trial and error which can be exceedingly time-consuming and tedious, further limiting the effectiveness of this approach in time-sensitive and production contexts.

Spatial Mapping

Spatial Mapping is a meta-programming technique that envisions Utomata algorithms as pre-existing within vast combinatorial spaces. Utomata Lab is an implementation of this technique which allows real-time interactive browsing of these spaces, akin to exploring a vast online map. While Utomata Lab does technically achieve this stated goal, it should be acknowledged that Spatial Mapping possesses significant limitations. Anything resembling an exhaustive search using this method is strictly impractical, and any form of selective exploration is likely to overlook significant findings. Even targeted excursions which seek to study well-defined subregions of any given space would likely involve generating and processing vast amounts of data. Its interpretation may further require the development of novel techniques, expertise and intuition.

The continuity of a Spatial Mapping was defined as its overall tendency to group similar algorithms and behaviours together. Highly continuous Spatial Mappings are ideal targets for exploratory and taxonomical studies because such spaces can be sampled by region, rather than exhaustively searched. While the mappings presented in this thesis do exhibit continuity, compared to randomly distributed spaces, this crucial aspect has not been optimised.

Adapting a Spatial Mapping for more efficient exploration is a multi-faceted challenge which involves a combination of advanced techniques that are beyond the scope of this thesis. For example, a precise characterisation of genotypic redundancy in a given space, and subsequent elimination of genotypic twins can significantly reduce the size of a space. However, this involves careful analysis and optimisations that may not even be transferable between different types of algorithmic expressions. This is even more pronounced in the case of phenotypic redundancy, which is drastically more difficult to assess and mitigate.

While Spatial Mappings technically consist of all encompassing spaces of behaviour, a significant amount of possible phenomena in any given mapping can easily be made inaccessible due to a range of factors. For example, algorithms that feature real-valued numerical constants cannot be effectively explored using this method. While field parameterisation can be incorporated to address this, most possible behaviours will necessarily remain well beyond reach. Likewise, the configuration function can have a profound effect on the range of phenomena that may emerge in a mapping. Choosing an appropriate configuration may require preparatory experiments to ensure sufficient understanding of the subject matter in order to select an appropriate configuration function for use in the mapping.

While this method was never intended to be exhaustive and is likely ineffective for most research contexts, it may nonetheless have significant conceptual merit, which should not be underestimated. Overcoming conceptual barriers may be crucial for gaining a deeper understanding of nonlinear systems, their edge cases or their limits. Spatial Mapping can offer a glimpse into a universe of potential emergent phenomena. Despite the overwhelming presence of vacant or incoherent findings in this universe, the mere act of observing it can have a profound impact on one's perspective.

8.4. Implications

CA Research

Both Utomata and the Spatial Mapping method have the potential to broaden the scope of CA research. These tools enhance existing methodologies by enabling systematic exploration of algorithmic spaces that were previously inaccessible. The identification and exploration of new CA families, Type-C and Type-U, not only demonstrate the effectiveness of these tools but also indicate an enormous untapped potential for discovering novel emergent phenomena in CA.

The project's non-instrumentalist approach to exploration and its emphasis on qualitative assessment of findings shifts the focus towards a more holistic view of CA. It offers a conceptual framework that seeks to revive the experimental, exploratory, and interdisciplinary spirit of early work in AL, while acknowledging the theoretical limitations that have since been established in regards to emergent phenomena. This revision of the strong-AL approach has the capacity to contribute to the overall diversification of CA algorithms, whose properties, limits and implications are still largely unknown.

Computational Arts

This work could potentially empower and inspire artists and designers to engage with exploration and study of CA by framing this practice as a medium for creative expression, rather than a scientific tool. It proposes and demonstrates new ways of exploration and real-time interaction with CA systems, potentially offering new avenues for artistic experimentation with emergent virtual phenomena. These may include performative, collaborative, and improvisational work. For example, “Echosystem” is a live audiovisual performance, presented at Printscreen Festival in 2019 [205]. A collaborative project between the author and composer Maayan Tsadka, this experimental live performance incorporates a custom, sound-reactive implementation of Utomata with real-time examination of microscopic samples and musical improvisation with found objects.

Figure 94.
Echosystem, An audio-visual live performance, in collaboration with Maayan Tsadka and Ensemble Nova (2019)

CA can also be incorporated into a wider range of real-time art forms, such as digital installations and net-art, as well as towards forming physical artefacts through digital fabrication, print, and sculpture. Moreover, in an artistic context, the act of CA exploration itself could be valued for its creative potential. This approach could lead to novel research paradigms, derived from artistic rather than scientific motivations, contributing to further diversification of known emergent phenomena.

Procedural content generation

The tools and methods presented in this thesis can also be applied to procedural content generation in computer games, simulations, and interactive media. Utomata can serve as a foundational engine for generating complex virtual content, including structure and texture synthesis, and as an alternative to random number and noise generators, especially in web based applications. Spatial Mapping techniques can facilitate automatic generation of vast amounts of diverse, adaptive virtual content, potentially serving as a basis for more advanced procedural generation techniques. Furthermore, this approach is not limited to CA; Spatial Mappings of other models and parametric spaces may prove valuable in other research contexts, such as numerical analysis and optimisation, physics simulation, combinatorial mathematics and data visualisation.

Virtual Worlds

It is important to emphasise the notable difference between nonlinear approaches such as CA and procedural approaches such as Perlin Noise. Procedural approaches are widely used for texture synthesis and terrain generation due to their predictability, reliability and consistency. However, these advantages also impose strict limitations on their output. For instance, Perlin Noise is a reliable and efficient technique for generating vast landscapes of continuous values, but it ultimately yields an extremely narrow range of phenomena that are essentially uniform throughout. In contrast, the expressive range of nonlinear approaches, such as CA and swarm dynamics, remains largely unknown. While these methods are considerably more difficult to predict and thus harder to incorporate in production environments, crucially, they can give rise to genuinely novel emergent virtual phenomena.

This can have significant implications for the prospect of virtual worlds. A mass diversification of algorithms capable of generating emergent systems can reduce the reliance on established procedural approaches, which are widely used in open-world games and virtual reality environments, or so-called metaverses. Novel approaches for exploring and implementing nonlinear systems may play a crucial role in facilitating the future creation of virtually infinite open worlds that are capable of supporting the emergence of complex

higher-order virtual phenomena. This untapped potential extends far beyond simulated physical environments. It evokes endless virtual environments which feature real-time, complex, intricate and unpredictable forms and processes.

In accordance with the concept of weak emergence, such instances of virtual phenomena can be detached from physical reality and explored as such. Spatial Mapping offers a glimpse into the immense size and potential of virtual worlds, strengthening the notion that algorithms and their outputs are not created but rather *discovered*. The exploration and study of these novel realms may indeed extend beyond conventional science, necessitating new paradigms for generative research and forming new bridges between scientific and artistic exploration. Ultimately, these would someday fulfil Langton's vision of *"something more poetic in the future of Science"* [13].

8.5. Future Research

A Model of Natural Selection

Type-U transition functions have a unique property, whereby new state values cannot be introduced into the system. This reductive property creates a highly competitive environment which forces the emergence of symbiotic relationships between neighbouring state values, thus forming self-assembling structures. These employ a range of strategies for maintaining internal stability, as well as equilibrium with their surrounding environment — both of which are crucial for persistence. Some algorithms, such as TU8, exhibit dozens of unique formations, all collaborating and competing with each other in a dynamic that can be likened to a delicate ecosystem.

This points to the potential of Type-U systems as abstract models of symbiogenesis and evolutionary dynamics. The unique and innately digital aesthetic of Type-U potentially distances these types of phenomena from other evolutionary models, which often rely on simulating physical properties such as harsh environments or limited food resources for creating competition. Type-U dynamics can be regarded as an implicit process of natural selection, whereby the genotype and phenotype are fused together as structural patterns that self-assemble within the program itself. Natural selection then emerges as an open-ended process in which emergent structures interact and adapt. Those who manage to persist, given the constraints of the algorithm and the conditions of their local environment, are those deemed the fittest. This stands in contrast to genetic algorithms, in which the genotype is explicitly defined as code or numerical data, and in which fitness is evaluated by an external function.

Type-U algorithms which have been observed to feature sufficient "fluidity" for supporting complex structures typically appear to converge towards one or two "victorious" patterns, constituting "dead-end" evolution. However, the prospect of sustained higher-order activity at a systematic level remains plausible. Future studies which aim at avoiding dead-end evolution may include exploring higher-order Type-U spaces, utilisation of asymmetric or more compound algorithmic expressions, combinations with other CA algorithms or careful introduction of random state values to explicitly induce mutations.

The "Goldilocks" Zone

The *Goldilocks Zone* in astronomy refers to the habitable zone around a star where conditions are just right for liquid water to exist. This hypothesis suggests that complex life forms are more likely to emerge in fluid environments due to their inherent chemical instability. Unlike solid environments, which inhibit chemical interactions, or gaseous environments, where interactions occur too

rapidly, fluid environments provide an optimal balance for the complex biochemical reactions necessary for life.

As discussed in Section 6.4, there are indications that complex higher-order structures in Type-U are more likely to emerge within phase transitions between chaotic and ordered dynamics. Future studies of Type-U algorithms can explore this hypothesis through both qualitative and quantitative means. To this end, quantitative measures of the phase of a system can be used as a ranking system for Type-U algorithms to explore a hypothetical narrow band where conditions are optimal for the emergence of higher-order structures.

For example, image compression can be used as a heuristic function for classifying the level of complexity in a static image. According to this approach, images of class 1 Type-U algorithms, which feature “simple stable or periodic structures”, would result in very small file sizes, as repetitive structures are highly compressible as JPEG images. On the other hand, class 3 algorithms, which “retain a chaotic pattern”, would result in large file sizes, due to their inherent randomness. It thus follows that class 2 algorithms may be identified along a particular band of image compression size. A normalised measure of entropy through JPEG compression has already been incorporated into Utomata Lab and Utomata Editor. This measure can be used to classify, sort, and filter Type-U algorithms in a future study.

Another example of a heuristic function examines the transition function over a single step. This method measures the polar vector between each transitioning cell and its target cell, as sampled by the U function. With the distance and angle represented by the red and green channels, a new image can then be generated by normalising and visualising the vector at each cell. The average gradient distance of this image can then be generated. Initial testing indicates a possible correlation between Type-U dynamics and such analyses, whereby lower average values may predict class 1 dynamics, high values predict class 3 dynamics, and mid-range values predict class 2 dynamics.

Exploratory studies of the Goldilocks zone in Type-U algorithms can help identify and collect class 2 algorithms, which can form the basis for deeper studies of Type-U dynamics. These may assist in identifying other unique signatures of class 2 algorithms, perhaps even at a genotypic level. This, in turn, would potentially allow for significantly more efficient Spatial Mappings by filtering out most of the incoherent phenomena in Type-U spaces.

Perfecting quantitative measures of emergent structures in Type-U may also prove useful towards other exploration methods, such as genetic programming. Using them as fitness functions, automatic evolutionary processes can be optimised for searching larger and more diverse combinatorial spaces of Type-U expressions, such as asymmetric, hybrid or exceptionally large expressions. In this context, Spatial Mapping can still be used as an effective complementary tool for gaining instant and unmediated access into any given combinatorial space.

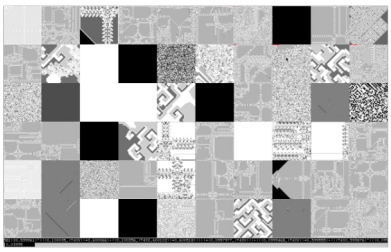


Figure 95.
Interactive evolutionary algorithm using an early version of Utomata in Openframeworks.

Evolutionary Programming

As previously stated, Utomata’s functional syntax is highly suited for evolutionary programming techniques. An interactive software experiment, which used an early version of Utomata in Openframeworks, demonstrates this capacity. The experiment consisted of an interactive evolutionary algorithm, where a user can visually examine a small set of Utomata algorithms and select a subset, from which a new generation is created. The use of aesthetic evaluation as the fitness function yielded mixed results; interactive evolutionary approaches are inherently slow and results are difficult to recreate. While this

approach was ultimately not pursued for this thesis, the prospect of evolving CA algorithms remains promising.

In Silico Experimental software

The interactive features of Utomata are currently limited to simple probes and interventions using the cursor, direct programming or the use of global variables. This is largely due to this work's focus on open-ended exploration. Extending Utomata to support analytical studies can be an important next step towards conducting deeper studies of CA behaviours. Such studies could emulate certain methodologies from microbial interaction studies through a set of tools for sampling and manipulating both genotypic and phenotypic specimens. Taking in-vitro studies as a metaphor, various software tools for analytical studies of CA can be developed:

- Specimen collection: a robust, streamlined and standardised method for sampling a specific region of interest within an active CA system, including its algorithm.
- Culture Preparation: a standard method for storing samples and basic tools for maintaining or recreating a nurturing environment.
- Co-culturing: a method for combining several different samples together to study their interactions. This may involve phenotypic samples of the same algorithm or combining multiple algorithms within the same culture.
- Advanced field parameterisations: this technique, discussed in Chapter 4, can be significantly improved through automatic symbol substitution and advanced UI features to allow freely moving between parametric variations.
- Time-Lapse Imaging: a reliable tool for running and periodically sampling algorithms over sustained periods of time.
- Automated Testing: scripting tools for automating precise repetitive perturbations and unit tests on large sets of CA samples.

A Study of Genotypic Redundancy

Redundancy was discussed in section 6.2.1 as the overall level of identical or nearly identical algorithms in a given Spatial Mapping. While some redundancy is unavoidable in Spatial Mappings, it can be mitigated to make exploration far more effective. The two main culprits of genotypic redundancy in Type-U are diagonal symmetry, which creates a perpendicular genotypic twin for every algorithm, and operator symmetry, which results from the use of symmetric operators such as addition and multiplication. While a precise determination of genotypic redundancy in Type-U algorithms is beyond the scope of this thesis, it represents an intriguing prospect for future cross-disciplinary research.

Future studies of genotypic redundancy in Type-U algorithms and other algorithms can be conducted by forming a bridge between CA research and the domains of combinatorial mathematics and set theory. These studies could involve exploring and analysing countable permutations of well-defined expressions to characterise their underlying properties. Such cross-disciplinary efforts could be mutually beneficial: they can introduce challenging problems for mathematicians to solve, contribute to a deeper understanding of CA algorithms and dynamics, and offer practical, testable results by reducing the size of Spatial Mappings.

Towards Other Domains

The Spatial Mapping method stems from a philosophical perspective that considers virtual phenomena as pre-existing entities which reside in a vast combinatorial space of potentialities. It employs custom algorithms and techniques to organise such spaces in order to make them accessible. While this method was specifically developed for CA algorithms, there is nothing about it that limits it to this particular domain. The prospect of extending Utomata and Spatial Mapping towards other computational models or analytical domains was indeed considered in various stages of this project and remains a prominent candidate for subsequent research. Initial efforts to adapt Utomata's functional syntax to three dimensional CA, L-systems, particle swarms and networks have been attempted but not yet perfected.

As noted in Chapter 3, Utomata's exceptionally wide expressive range provides a standardised method for implementing algorithms that are commonly considered too different to compare, such as GOL, ECA and RD. In the same sense, generalising Utomata to accommodate other models may offer a similar advantage, potentially offering novel ways for comparing different models of nonlinear systems.

For example, Craig Reynolds' seminal work on swarm dynamics [96], applies three local behaviours to each component in a system: separation, cohesion and align. While this model was adapted and implemented in various contexts beyond its initial purpose, such as in Swarm chemistry [86], Many such adaptations, particularly in the arts, primarily focus on parametric exploration. In contrast, exploring this model through Spatial Mapping techniques could offer significant diversification by effectively forming a space consisting of all possible ways in which a particle may react to those around it.

The Laboratory of Babel

As evident by its title, this thesis entertains the notion of a combinatorial space consisting of all possible CA algorithms. The Spatial Mapping method presented in Chapter 5 embodies a partial fulfilment of this idea as a tool for open-ended exploration. However, this method also constitutes a robust conceptual and practical foundation for a literal realisation of this idea.

Much of the effectiveness of Spatial Mapping stems directly from its ability to use minimalist symbol domain sets in order to constrain the resulting space to an (arguably) explorable size. In other words, very large Spatial Mappings are simply too big for manual exploration to yield meaningful findings. Nonetheless, setting out to create a literal implementation of a Laboratory of Babel stands as a future project which may hold considerable theoretical, conceptual and artistic merit.

In order to accomplish this, the Spatial Mapping method would need to be extended to three dimensions rather than two. The third dimension — algorithm topology — would replace the source algorithm, which it currently accepts as one of its mapping parameters. Besides the X and Y coordinates, which are mapped to symbol domains, a third coordinate, T, would be mapped to algorithm topology using a custom function that enumerates binary/unary tree topologies. This mapping would also need to use maximalist symbol domain sets in order to maximise coverage. Such a "total laboratory" would include, at the very least, all of Utomata's binary and unary operators, all of Utomata's totalistic and outer-totalistic neighbourhood variables, all colour channel combinations (`.rrr`, `.grg`, `.bbr`), as well as a comprehensive set of numerical constants.

While theoretically possible, this endeavour presents several significant challenges. Enumerating binary tree topologies is a well-known combinatorial problem with various algorithmic trade-offs. Moreover, even the largest possible Spatial Mapping can never contain *all* CA algorithms, if only due to the presence of real-valued numerical constants. However, most importantly, the sheer magnitude of such a combinatorial space is quite simply beyond comprehension. This last statement, in itself, may be regarded as either the strongest argument for pursuing this endeavour — or the strongest argument not to.

8.6. Final Reflections

This thesis presents an experimental computational arts practice spanning nine years of research. The tools, methods and studies featured here have been carefully selected from numerous experiments, artworks, trials and errors. In retrospect, this Sisyphean path may have been the only viable route towards cultivating the philosophical view that underlies this thesis. While this view is not a formal contribution, it is regarded by the author as this project's greatest achievement.

The development of this research project has coincided with a time of historical leaps in artificial intelligence. Its revolutionary potential notwithstanding, generative AI is rooted in the human experience; it is designed to model and enact human thinking, interaction, intuition, knowledge, and creativity. In contrast, this work aims to draw attention away from these inherent familiarities towards the abyss of the inherently unfamiliar, where true novelty may reside. This should not be regarded as a critique of anthropocentric or instrumentalist views of computing technology but an acknowledgment of its boundless potential. It serves as a humbling reminder that human intelligence is merely a special case of intelligent behaviour, itself, a property of only a small subset of living systems, which, in turn, constitute only a minuscule fraction of possible emergent phenomena.

Through its focus on open-ended exploration, this project aspires to join the canon of works that expand human horizons by looking in the most unlikely of places. Such works must often exert significant energy on breaking free from conceptual bias. Future explorers of emergent virtual structures may find this to be an exceedingly confusing and lonely endeavour. If nothing else, this thesis hopes to serve as a compass to help them navigate this vast, unstructured domain. To these future explorers, the author extends one last piece of advice, which he has often failed to follow: open-ended research is best thought of as a bottomless pit. While it is tempting to insist on reaching the bottom, this takes a lifetime to achieve. Instead, simply describe the fall.

9.Bibliography

[1] J. L. Borges, “The Library of Babel,” in *Collected Fictions*, Penguin, 1999.

[2] J. Basile, “The Library of Babel,” *Library of Babel* [website], 2015. [Online]. Available: https://libraryofbabel.info. (Accessed: Jan. 12, 2025).

[3] J. Basile, *Tar for Mortar: “The Library of Babel” and the Dream of Totality*. punctum books, 2018.

[4] S. Ulam, “On some mathematical problems connected with patterns of growth of figures,” in *Proceedings of Symposia in Applied Mathematics*, vol. 14, Providence, RI, USA: Am. Math. Soc., 1962, pp. 215–224.

[5] J. von Neumann, A. W. Burks, *Theory of Self-Reproducing Automata*, vol. 1102024. Urbana, IL, USA: University of Illinois Press, 1966.

[6] U. Lagerkvist, *Pioneers of Microbiology and the Nobel Prize*. World Scientific, 2003, pp. 51–55.

[7] B. Skuse, “The third pillar,” *Physics World*, vol. 32, no. 3, p. 40, 2019.

[8] P. Manneville, N. Boccara, G. Y. Vichniac, and R. Bidaux, *Cellular Automata and Modeling of Complex Physical Systems: Proceedings of the Winter School, Les Houches, France*, February 21–28, 1989. Springer Science & Business Media, 2012.

[9] D. Eppstein, “Growth and decay in life-like cellular automata,” in *Game of Life Cellular Automata*, A. Adamatzky, Ed. London, UK: Springer London, 2010, pp. 71–97.

[10] J. E. Pearson, “Complex patterns in a simple system,” *Science*, vol. 261, no. 5118, pp. 189–192, Jul. 1993.

[11] P. Bak, C. Tang, and K. Wiesenfeld, “Self-organized criticality: An explanation of the 1/f noise,” *Physical Review Letters*, vol. 59, no. 4, pp. 381–384, Jul. 1987.

[12] C. Langton, Artificial Life: *Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*. Routledge, 2019.

[13] J. Horgan, “From complexity to perplexity,” *Scientific American*, vol. 272, no. 6, pp. 104–109, 1995.

[14] H. H. Pattee, “Simulations, realizations, and theories of life,” in *Artificial Life (ALIFE)*, 1987, pp. 63–78.

[15] M. A. Bedau, “Weak emergence,” Philosophical Perspectives, vol. 11, pp. 375–399, 1997.

[16] G. Deleuze, *Difference and Repetition*. [Online]. Available: https://www.academia.edu/download/18052021/20110414150123784.pdf. (Accessed: Sep. 27, 2022), p. 208.

[17] R. Frigg, “Models and representation: Why structures are not enough,” *Measurement*, 2002.

[18] W. R. Ashby, “Principles of the self-organizing system,” in *Modern Systems Research for the Behavioral Scientist*, pp. 108–118, 1968.

[19] L. von Bertalanffy, “General system theory,” *General Systems*, vol. 1, no. 1, pp. 11–17, 1956.

[20] V. Illingworth, Ed., *The Penguin Dictionary of Physics. London*, U.K.: Penguin Books, 1991.

[21] D. Campbell, D. Farmer, J. Crutchfield, and E. Jen, “Experimental mathematics: The role of computation in nonlinear science,” *Communications of the ACM*, vol. 28, no. 4, pp. 374–384, 1985.

[22] E. Fermi, J. Pasta, S. Ulam, and M. Tsingou, *Studies of Nonlinear Problems*, Los Alamos Scientific Laboratory, LA-1940, May 1955.

[23] A. M. Turing, “The chemical basis of morphogenesis,” *Bulletin of Mathematical Biology*, vol. 52, no. 1–2, pp. 153–197, 1990.

[24] A. M. Turing, “I.—Computing machinery and intelligence,” *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950.

[25] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[26] N. A. Baricelli, “Numerical testing of evolution theories, part II preliminary tests of performance,” *Symbiogenesis and Terrestrial Life*, Acta Biotheoretica, vol. 16, pp. 99–126, 1962.

[27] W. R. Ashby, *An Introduction to Cybernetics*. London, UK: Chapman & Hall Ltd, 1961.

[28] R. Melnik, *Mathematical and Computational Modeling*. Wiley Online Library, 2015.

[29] J. S. Mill, *A System of Logic, Ratiocinative and Inductive*. London, UK: John W. Parker, 1843.

[30] W. Banzhaf, “Self-organizing systems,” in *Encyclopedia of Complexity and Systems Science*, vol. 14, pp. 589, 2009.

[31] V. Darley, “Emergent phenomena and complexity,” *Artificial Life,* vol. 4, pp. 411–416, 1994.

[32] . M. Teo, B. L. Luong, and C. Szabo, “Formalization of emergence in multi-agent systems,” in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2013, pp. 231–240.

[33] Y. Bar-Yam, “A mathematical theory of strong emergence using multiscale variety,” *Complexity*, vol. 9, no. 6, pp. 15–24, 2004.

[34] P. Huneman, “Emergence and adaptation,” *Minds and Machines*, vol. 18, no. 4, pp. 493–520, 2008.

[35] H. Thorén and P. Gerlee, “Weak emergence and complexity,” in *Artificial Life XII: Proceedings of the Twelfth International Conference on the Synthesis and Simulation of Living Systems*, H. Fellerman et al., Eds. Cambridge, MA: MIT Press, 2010, pp. 879–886.

[36] T. S. Ray, “Evolution and optimization of digital organisms,” in *Scientific Excellence in Supercomputing: The IBM 1990 Contest Prize Papers*, K. R. Billingsley, E. Derohanes, and H. Brown III, Eds. Athens, GA: The Baldwin Press, The University of Georgia, 1991, pp. 489–531.

[37] E. P. Rybicki, “The classification of organisms at the edge of life, or problems with virus systematics,” *South African Journal of Science*, vol. 86, no. 4, pp. 182–186, 1990.

[38] E. Salvucci, “Microbiome, holobiont and the net of life,” *Critical Reviews in Microbiology*, vol. 42, no. 3, pp. 485–494, 2016.

[39] D. E. Koshland, “The seven pillars of life,” *Science*, vol. 295, no. 5563, pp. 2215–2216, 2002.

[40] B. Szigeti *et al.*, “OpenWorm: An open-science approach to modeling Caenorhabditis elegans,” *Frontiers in Computational Neuroscience*, vol. 8, p. 137, 2014.

[41] S. Bullock, “Levins and the lure of artificial worlds,” *The Monist*, vol. 97, no. 3, pp. 301–320, 2014.

[42] C. G. Langton, “Self-reproduction in cellular automata,” *Physica D: Nonlinear Phenomena*, vol. 10, no. 1–2, pp. 135–144, 1984.

[43] C. Langton, “Computation at the edge of chaos: Phase transitions and emergent computation,” *Physica D: Nonlinear Phenomena*, vol. 42, no. 1–3, pp. 12–37, Jun. 1990.

[44] M. A. Bedau, “A functional account of degrees of minimal chemical life,” *Synthese*, vol. 185, no. 1, pp. 73–88, 2012.

[45] M. A. Bedau, “What is life?,” in *A Companion to the Philosophy of Biology*, S. Sarkar and A. Plutynski, Eds. Oxford, UK: Blackwell Publishing Ltd, 2008, pp. 455–471.

[46] M. A. Bedau et al., “Open problems in artificial life,” *Artificial Life*, vol. 6, no. 4, pp. 363–376, 2000.

[47] T. Grüne-Yanoff, “Appraising non-representational models,” *Philosophy of Science*, 2013. [Online]. Available: https://philsci-archive.pitt.edu/9420/1/ApprNonRepModels120227.pdf. (Accessed: Jan. 12, 2025).

[48] S. Wolfram, *A New Kind of Science*. Champaign, IL, USA: Wolfram Media, 2002.

[49] M. Sipper, “Fifty years of research on self-replication: An overview,” *Artificial Life*, vol. 4, no. 3, pp. 237–257, 1998.

[50] T. J. Hutton, “Evolvable self-reproducing cells in a two-dimensional artificial chemistry,” *Artificial Life*, vol. 13, no. 1, pp. 11–30, 2007.

[51] T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*. Cambridge, MA, USA: MIT Press, 1987.

[52] A. Ilachinski, *Cellular Automata: A Discrete Universe*. Singapore: World Scientific, 2001.

[53] M. Gardner, “Mathematical games: The fantastic combinations of John Conway’s new solitaire game ‘Life,’” *Scientific American*, vol. 223, no. 4, pp. 120–123, Oct. 1970.

[54] J. Conway, “Conway’s Game of Life on RosettaCode,” 1970. [Online]. Available: https://rosettacode.org/wiki/Conway%27s_Game_of_Life. Accessed: Jul. 28, 2020.

[55] S. Wolfram, “Universality and complexity in cellular automata,” *Physica D: Nonlinear Phenomena*, vol. 10, no. 1, pp. 1–35, Jan. 1984.

[56] A. K. Dewdney, “Computer recreations: The cellular automata programs that create Wireworld, Rugworld, and other diversions,” *Scientific American*, vol. 262, no. 1, pp. 146–149, Jan. 1990.

[57] K. Sims, “Reaction Diffusion Tutorial,” 2013. [Online]. Available: https://www.karlsims.com/rd.html. Accessed: Aug. 23, 2020.

[58] I. Chen and F. Liu, “Spatial simulation of COVID-19 new cases development,” *medRxiv*, preprint, Sep. 2021. [Online]. Available: https://doi.org/10.1101/2021.09.19.21263799. Accessed: Feb. 2, 2025.

[59] D. Alonso and R. V. Sole, “The DivGame simulator: A stochastic cellular automata model of rainforest dynamics,” *Ecological Modelling*, vol. 133, no. 1–2, pp. 131–141, 2000.

[60] H. Arata et al., “Free-form shape modeling by 3D cellular automata,” in *Proc. Shape Modeling* Int. Conf. (SMI’99), Aizu-Wakamatsu, Japan, 1999, pp. 242–251.

[61] D. Di Caprio, D. Dung, et al., “3D cellular automata simulations of intra and intergranular corrosion,” *Corrosion Science*, vol. 112, pp. 438–450, 2016.

[62] L. Benguigui, D. Czamanski, and R. Roth, “Modeling cities in 3D: A cellular automaton approach,” *Environ. Plan. B: Plan. Des.*, vol. 35, no. 3, pp. 413–430, 2008.

[63] J. Hemmingsson and G. Peng, “Phase transition from periodic to quasiperiodic behaviour in 4D cellular automata,” *J. Phys. A: Math. Gen.*, vol. 27, no. 8, p. 2735, 1994.

[64] V. Sardenberg, I. Tebaldi, E. Bier, and N. Prado, *Four-Dimensional Objects, Cellular Automata, and Virtual Reality – The Hypercocoon Project*, presented at the 2022 International Conference on Virtual Reality and Visualization (ICVRV). [Online]. Available: https://www.researchgate.net/publication/362751310_Four-dimensional_objects_Cellular_Automata_and_Virtual_Reality_-The_Hypercocoon_project. Accessed: Feb. 2, 2025.

[65] Wolfram Physics Project, “The notion of dimension,” *Wolfram Physics: Technical Introduction*, 2020. [Online]. Available: https://www.wolframphysics.org/technical-introduction/limiting-behavior-and-emergent-geometry/the-notion-of-dimension/. Accessed: Accessed: Feb. 2, 2025.

[66] P. Rendell, Turing Machine Universality of the Game of Life, Cham, Switzerland: *Springer*, 2016.

[67] S. A. Silver, “Life lexicon (introduction),” *ConwayLife.com*, Oct. 2022. [Online]. Available: http://conwaylife.com/ref/lexicon/lex.htm. Accessed: Oct. 9, 2022.

[68] N. Johnston and D. Greene, “Conway’s Game of Life: Mathematics and construction,” *Zenodo*, 2022. [Online]. Available: https://doi.org/10.5281/ZENODO.6097284. Accessed: Oct. 9, 2022.

[69] S. Rafler, “Generalization of Conway’s ‘Game of Life’ to a continuous domain—SmoothLife,” *arXiv preprint*, arXiv:1111.1567, 2011. [Online]. Available: https://arxiv.org/abs/1111.1567. Accessed: Oct. 5, 2022.

[70] “List of Life-like cellular automata,” ConwayLife.com, Oct. 2022. [Online]. Available: https://conwaylife.com/wiki/List_of_Life-like_cellular_automata. Accessed: Oct. 5, 2022.

[71] D. Eppstein, “Searching for spaceships,” *arXiv*, vol. cs.AI, Apr. 10, 2000. [Online]. Available: https://arxiv.org/abs/cs.AI/0004003. Accessed: Oct. 5, 2022.

[72] E. Peña and H. Sayama, “Life worth mentioning: Complexity in life-like cellular automata,” *Artificial Life*, vol. 27, no. 2, pp. 105–112, Nov. 2021.

[73] S. Wolfram, “Tables of cellular automaton properties,” in *Theory and Applications of Cellular Automata*, S. Wolfram, Ed. Singapore: World Scientific, 1986, pp. 485–557.

[74] N. H. Packard and S. Wolfram, “Two-dimensional cellular automata,” *Journal of Statistical Physics*, vol. 38, no. 5, pp. 901–946, Mar. 1985.

[75] S. Wolfram, “Random sequence generation by cellular automata,” *Advances in Applied Mathematics*, vol. 7, no. 2, pp. 123–169, Jun. 1986.

[76] G. J. Martinez, A. Adamatzky, and R. Alonso-Sanz, “Complex dynamics of elementary cellular automata emerging from chaotic rules,” *arXiv*, vol. nlin.CG, Mar. 27, 2012. [Online]. Available: https://doi.org/10.1142/s021812741250023x. Accessed: Oct. 15, 2022.

[77] K. Sutner, “Classification of cellular automata,” in Encyclopedia of Complexity and Systems Science, Part, 2009. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.68.1572&rep=rep1&type=pdf. Accessed: Oct. 15, 2022.

[78] K. Sutner, “Computational classification of cellular automata,” *International Journal of General Systems*, vol. 41, no. 6, pp. 595–607, Aug. 2012.

[79] A. Adamatzky, “Identification of cellular automata,” in *Encyclopedia of Complexity and Systems Science*, R. A. Meyers, Ed. New York, NY, USA: Springer, 2009, pp. 4739–4751.

[80] G. Braga, G. Cattaneo, P. Flocchini, and C. Q. Vogliotti, “Pattern growth in elementary cellular automata,” *Theoretical Computer Science*, vol. 145, no. 1, pp. 1–26, Jul. 1995.

[81] P. Gray and S. K. Scott, “Autocatalytic reactions in the isothermal, continuous stirred tank reactor: Isolas and other forms of multistability,” *Chemical Engineering Science*, vol. 38, no. 1, pp. 29–43, Jan. 1983.

[82] R. Munafo, “Reaction-diffusion by the Gray-Scott model: Pearson’s parametrization,” 2013. [Online]. Available: http://mrob.com/pub/comp/xmorphia/. Accessed: Aug. 23, 2020.

[83] “Reaction-diffusion explorer tool,” [Online]. Available: https://www.karlsims.com/rdtool.html. Accessed: Oct. 4, 2022.

[84] Y. H. Chew and F. Spill, “Discretised flux balance analysis for reaction-diffusion simulation of single-cell metabolism,” bioRxiv, preprint, Aug. 1, 2023. [Online]. Available: https://doi.org/10.1101/2023.08.01.551453. Accessed: Aug. 23, 2020.

[85] A. Witkin and M. Kass, “Reaction-diffusion textures,” in *Proc. 18th Annu. Conf. Comput. Graph. Interact. Techn.* (SIGGRAPH ’91), Las Vegas, NV, USA, 1991, pp. 299–308.

[86] H. Sayama, “Swarm chemistry,” *Artificial Life,* vol. 15, no. 1, pp. 105–114, 2009.

[87] B. W.-C. Chan, “Lenia and expanded universe,” arXiv, vol. nlin.CG, May 7, 2020. [Online]. Available: http://arxiv.org/abs/2005.03742. Accessed: Aug. 25, 2020.

[88] Q. T. Davis, Q. T. Davis, and J. Bongard, “Selecting continuous life-like cellular automata for halting unpredictability,” in Proc. Genetic Evol. Comput. Conf. Companion (GECCO), 2022. [Online]. Available: https://doi.org/10.1145/3520304.3529037. Accessed: Aug. 25, 2020.

[89] K. Bhattacharjee, N. Naskar, S. Roy, and S. Das, “A survey of cellular automata: Types, dynamics, non-uniformity and applications,” Natural Computing, vol. 19, no. 2, pp. 433–461, Jun. 2020.

[90] D. L. Turcotte, “Seismicity and self-organized criticality,” *Physics of the Earth and Planetary Interiors,* vol. 111, no. 3–4, pp. 275–293, 1999.

[91] C. G. Langton, “Studying artificial life with cellular automata,” *Physica D: Nonlinear Phenomena*, vol. 22, no. 1–3, pp. 120–149, 1986.

[92] A. Gajardo, A. Moreira, and E. Goles, “Complexity of Langton’s ant,” *Discrete Applied Mathematics*, vol. 117, no. 1, pp. 41–50, Mar. 2002.

[93] G. Tempesti, “A new self-reproducing cellular automaton capable of construction and computation,” in Advances in *Artificial Life*, Berlin, Germany: Springer, 1995, pp. 555–563.

[94] V. Gladkikh, A. Nigay, and the International University of Information Technologies, “Wireworld++: A cellular automaton for simulation of nonplanar digital electronic circuits,” Complex Systems, vol. 27, no. 1, pp. 19–44, Mar. 2018.

[95] “The Wireworld computer,” [Online]. Available: https://www.quinapalus.com/wi-index.html. Accessed: Jun. 4, 2023.

[96] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” in *Proc. 14th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH ’87)*, Anaheim, CA, USA, 1987, pp. 25–34.

[97] W. T. Reeves, “Particle systems—A technique for modeling a class of fuzzy objects,” *ACM Transactions on Graphics*, vol. 2, no. 2, pp. 91–108, Apr. 1983.

[98] M. Sandler, A. Zhmoginov, L. Luo, A. Mordvintsev, E. Randazzo, and B. Agúera y Arcas, “Image segmentation via cellular automata,” *arXiv*, vol. cs.CV, Aug. 11, 2020. [Online]. Available: http://arxiv.org/abs/2008.04965. Accessed: Sep. 30, 2022.

[99] B. Chopard, “Cellular automata and lattice Boltzmann modeling of physical systems,” in *Handbook of Natural Computing*, Berlin, Heidelberg: Springer, 2012, pp. 287–331.

[100] H. Situngkir, “Exploring ancient architectural designs with cellular automata,” *arXiv*, vol. cs.CY, Aug. 13, 2015. [Online]. Available: http://arxiv.org/abs/1508.03610. Accessed: Sep. 30, 2022.

[101] P. Coates, N. Healy, C. Lamb, and W. L. Voon, “The use of cellular automata to explore bottom-up architectonic rules,” 1996. [Online]. Available: https://repository.uel.ac.uk/download/d3aedf91a9fe865d5d4863f29c921ad4b4a43b77838fb3ba785cfaa4db97b5b9/507983/Coates%20P%20%281996%29%20Eurographics.pdf. Accessed: Sep. 30, 2022.

[102] K. Sims, “Interactive evolution of dynamical systems,” in Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, F. J. Varela and P. Bourgine, Eds. Cambridge, MA, USA: MIT Press, 1992, pp. 171- 178.

[103] D. Ashlock and J. Tsang, “Evolved art via control of cellular automata,” in *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, Trondheim, Norway, 2009, pp. 3338–3344. doi: 10.1109/CEC.2009.4983368.

[104] A. G. Forbes, “Interactive Cellular Automata Systems for Creative Projects,” in *Cellular Automata in Image Processing and Geometry*, P. Rosin, A. Adamatzky, and X. Sun, Eds. Cham, Switzerland: Springer, 2014, pp. 253–272.

[105] J. Heaton, “Evolving continuous cellular automata for aesthetic objectives,” *Genetic Programming and Evolvable Machines*, vol. 19, no. 1–2, pp. 27–63, 2018. doi: 10.1007/s10710-018-9336-1.

[106] M. Mitchell, P. T. Hraber, and J. P. Crutchfield, “Revisiting the edge of chaos: Evolving cellular automata to perform computations,” *Complex Systems*, vol. 7, no. 2, pp. 89–130, 1994.

[107] R. Das, M. Mitchell, and J. P. Crutchfield, “Evolving globally synchronized cellular automata,” in *Proceedings of the 6th International Conference on Genetic Algorithms*, 1995, pp. 336–343.

[108] M. Sipper, *Evolution of Parallel Cellular Machines: The Cellular Programming Approach.* Berlin, Heidelberg: Springer, 1997.

[109] N. H. Packard, “Adaptation toward the edge of chaos,” in *Dynamic Patterns in Complex Systems*, J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, Eds. Singapore: World Scientific, 1988, pp. 293–301.

[110] H. Juillé and J. B. Pollack, “Coevolutionary learning: A case study,” in *Proceedings of the 15th International Conference on Machine Learning*, 1998, pp. 251–259.

[111] T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*. Cambridge, MA: MIT Press, 1987.

[112] A. Mordvintsev, E. Randazzo, E. Niklasson, and M. Levin, “Growing neural cellular automata,” Distill, vol. 5, no. 2, Feb. 2020. [Online]. Available: https://distill.pub/2020/growing-ca. Accessed: Sep. 30, 2022.

[113] E. Niklasson, A. Mordvintsev, E. Randazzo, and M. Levin, “Self-organising textures,” Distill, vol. 6, no. 2, Feb. 2021. [Online]. Available: https://distill.pub/selforg/2021/textures. Accessed: Sep. 30, 2022.

[114] S. Earle, J. Snider, M. C. Fontaine, S. Nikolaidis, and J. Togelius, “Illuminating diverse neural cellular automata for level generation,” in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO ’22)*, Boston, MA, USA, Jul. 2022, pp. 68–76. [Online]. Available: https://doi.org/10.1145/3512290.3528754. Accessed: Sep. 30, 2022.

[115] W. Aguilar, G. Santamaría-Bonfil, T. Froese, and C. Gershenson, “The past, present, and future of artificial life,” *Frontiers in Robotics and AI*, vol. 1, p. 8, 2014.

[116] K. Sims, “Particle animation and rendering using data parallel computation”, in *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, 1990, pp. 405–413.

[117] K. Sims, “Evolving 3D morphology and behavior by competition,” *Artificial Life*, vol. 1, no. 4, pp. 353–372, 1994.

[118] K. Sims, “Evolving virtual creatures,” in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, 1994, pp. 15–22.

[119] K. Sims, “Choreographed image flow,” Journal of Visualization and Computer Animation, vol. 3, no. 1, pp. 31–43, 1992.

[120] K. Sims, “Galápagos,” 1997. [Online]. Available: http://www.karlsims.com/galapagos/. Accessed: Sep. 27, 2022.

[121] K. Sims, “Reaction-Diffusion Media Wall.” [Online]. Available: https://www.karlsims.com/rd-exhibit.html. Accessed: Sep. 27, 2022.

[122] K. Sims, “Karl Sims – Home.” [Online]. Available: https://www.karlsims.com/. Accessed: Sep. 27, 2022.

[123] K. Sims, “Reaction-Diffusion Explorer Tool.” [Online]. Available: https://www.karlsims.com/rdtool.html. Accessed: Sep. 27, 2022.

[124] A. Lomas, “Aggregation: Complexity out of simplicity,” in *ACM SIGGRAPH 2005 Sketches*, Los Angeles, CA, USA, 2005, p. 98.

[125] T. A. Witten Jr. and L. M. Sander, “Diffusion-limited aggregation, a kinetic critical phenomenon,” *Phys. Rev. Lett.*, vol. 47, no. 19, pp. 1400–1403, Nov. 1981.

[126] A. Lomas, “Cellular forms: An artistic exploration of morphogenesis,” in *ACM SIGGRAPH 2014 Studio*, Vancouver, BC, Canada, Aug. 2014, p. 1.

[127] A. Lomas, “On hybrid creativity,” Arts, vol. 7, no. 3, p. 25, Jul. 2018.

[128] J. McCormack and A. Lomas, “Understanding Aesthetic Evaluation using Deep Learning,” in *Proceedings of the 9th International Conference on Computational Intelligence in Music, Sound, Art and Design (EvoMUSART 2020)*, Seville, Spain, Apr. 2020, pp. 118–133.

[129] A. Lomas, “Species Explorer: An interface for artistic exploration of multi-dimensional parameter spaces,” in Electronic Visualisation and the Arts (EVA 2016), London, UK, Jul. 2016, pp. 95–102.

[130] T. J. Hutton, “A Functional Self-Reproducing Cell in a Two-Dimensional Artificial Chemistry,” in *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (Artificial Life IX)*, J. Pollack, M. Bedau, P. Husbands, T. Ikegami, and R. A. Watson, Eds., Boston, MA, USA: MIT Press, 2004, pp. 444–449.

[131] T. J. Hutton, “Codd’s Self-Replicating Computer,” *Artificial Life*, vol. 16, no. 2, pp. 99–117, Spring 2010.

[132] A. Trevorrow, T. Rokicki, and D. Wills, “Golly: An open-source, cross-platform tool for exploring Conway’s Game of Life and other cellular automata,” 2013. [Online]. Available: http://golly.sourceforge.net/ Accessed: Sep. 7, 2020.

[133] T. Hutton, R. Munafo, A. Trevorrow, T. Rokicki, and D. Wills, “Ready: A cross-platform implementation of various reaction-diffusion systems,” 2015. [Online]. Available: https://github.com/GollyGang/ready Accessed: Sep. 7, 2020.

[134] S. Jenson, “Physarum,” Feb. 2019. [Online]. Available: https://sagejenson.com/physarum. Accessed: Jul. 27, 2020.

[135] J. Jones, “Characteristics of pattern formation and evolution in approximations of Physarum transport networks,” *Artificial Life*, vol. 16, no. 2, pp. 127–153, Spring 2010.

[136] U. Wilensky, “NetLogo,” Center for Connected Learning and Computer-Based Modeling, Northwestern University, *Evanston*, IL, 1999. [Online]. Available: https://ccl.northwestern.edu/netlogo/. Accessed: Sep. 17, 2020.

[137] C. Reas and B. Fry, Processing: A Programming Handbook for Visual Designers and Artists, 2nd ed. Cambridge, MA: *MIT Press*, 2014.

[138] C. Reas, A Mathematical Theory of Communication. Paris, France: *RRose Editions*, 2018.

[139] C. Reas, “Process 10 (Installation 1),” 2004. [Online]. Available: https://reas.com/ti_s/. Accessed: Sep. 17, 2020.

[140] C. Reas, “Textile Room,” 2013. [Online]. Available: https://reas.com/textile_room/. Accessed: Sep. 17, 2020.

[141] C. Reas, “Compendium Series,” 2004–2010. [Online]. Available: https://reas.com/compendium_b_p/. Accessed: Sep. 17, 2020.

[142] Slackermanz, YouTube Channel. [Online Video]. Available: https://www.youtube.com/channel/UCmoNsNuM0M9VsIXfm2cHPiA. Accessed: Oct. 4, 2022.

[143] Beautypi, “SlackermanzCA - Shadertoy BETA.” [Online]. Available: https://www.shadertoy.com/user/SlackermanzCA. Accessed: Oct. 4, 2022.

[144] CellularAutomationUploader, YouTube Channel. [Online Video]. Available: https://www.youtube.com/channel/UCZD4RoffXIDoEARW5aGkEbg. Accessed: Oct. 4, 2022.

[145] S. Alexander-Adams, *Simon Alexander-Adams*. [Online]. Available: https://www.simonaa.media/. Accessed: Oct. 4, 2022.

[146] Simon Alexander-Adams, “Creating Generative Visuals with Complex Systems”, Sep. 8, 2020. [Online Video]. Available: https://www.youtube.com/watch?v=VBzIPLh-ECg. Accessed: Oct. 4, 2022.

[147] Arsiliath, “psychobiotik.” [Online]. Available: https://twitter.com/psychobiotik?lang=en. Accessed: July 27, 2020.

[148] L. Wilson, “TodePond - Overview” [Online]. Available: https://github.com/TodePond. Accessed: Oct. 4, 2022.

[149] J. Giles, “Jazer Giles on Instagram.” [Online]. Available: https://www.instagram.com/jazergiles/. Accessed: May 28, 2021.

[150] J. Giles (@jazergiles). Instagram. [Online]. Available: https://jazergiles.com. Accessed: Oct. 4, 2022.

[151] Kerim Safa (@kerimsafa). Instagram. [Online]. Available: https://www.instagram.com/kerimsafa/. Accessed: Sep. 25, 2022.

[152] KiM ASENDORF (@kimasendorf). Instagram. [Online]. Available: https://www.instagram.com/kimasendorf/. Accessed: Sep. 25, 2022.

[153] Julian Hespenheide (@julian_hespenheide). Instagram. [Online]. Available: https://www.instagram.com/julian_hespenheide/. Accessed: Sep. 25, 2022.

[154] A. Gysin, Andreas Gysin Portfolio. [Online]. Available: https://ertdfgcvb.xyz/. Accessed: Sep. 17, 2020.

[155] SPACEFILLER (@space.filler.art). Instagram. [Online]. Available: https://www.instagram.com/space.filler.art/. Accessed: Oct. 4, 2022.

[156] Clusters. [Online]. Available: https://www.ventrella.com/Clusters/. Accessed: Oct. 4, 2022.

[157] Josef Pelz. [Online]. Available: https://www.josefpelz.com. Accessed: Oct. 4, 2022.

[158] Sander Sturing (@sandersturing). Instagram. [Online]. Available: https://www.instagram.com/sandersturing/. Accessed: Oct. 4, 2022.

[159] CoC (@codeofconquer). Instagram. [Online]. Available: https://www.instagram.com/codeofconquer/. Accessed: Oct. 4, 2022.

[160] A. Hoff, Inconvergent. [Online]. Available: https://inconvergent.net/. Accessed: Sep. 17, 2020.

[161] J. Rosenkrantz and J. Louis-Rosenberg, Nervous System Studio. [Online]. Available: https://n-e-r-v-o-u-s.com/. Accessed: Oct. 3, 2020.

[162] Lia, *liaworks*. [Online]. Available: https://www.liaworks.com/. Accessed: Sep. 17, 2020.

[163] J. T. Laurie Tarbell, *Levitated*. [Online]. Available: http://levitated.net/. Accessed: Sep. 17, 2020.

[164] N. Leto, *SAILOR*, 2010. [Online]. Available: https://www.imdb.com/title/tt2007453/. Accessed: Jul. 28, 2020.

[165] N. Leto, *Lifeshapes/Bryły życiorysów*, 2010. [Online Video]. Available: https://www.youtube.com/watch?v=DDF5uPNBuSk. Accessed: Jul. 28, 2020.

[166] N. Oxman, Mushtari, *Jupiter's Wanderer*. [Online]. Available: https://neri.media.mit.edu/projects/details/mushtari.html. Accessed: Oct. 3, 2020.

[167] F. Nassetti, *Filippo Nassetti Design.* [Online]. Available: http://mhoxdesign.com/design-en.html. Accessed: Oct. 3, 2020.

[168] E. Driessens and M. Verstappen, *Breed*. [Online]. Available: https://notnot.home.xs4all.nl/breed/Breed.html. Accessed: Oct. 3, 2020.

[169] T. Beddard, *sub.blue - Artworks*. [Online]. Available: http://sub.blue/. Accessed: Oct. 3, 2020.

[170] R. Jarman and J. Gerhardt, *Semiconductor Films - Artworks*. [Online]. Available: https://semiconductorfilms.com/art/. Accessed: Oct. 3, 2020.

[171] Processing. [Online]. Available: https://processing.org/. Accessed: Feb. 1, 2025.

[172] p5.js. [Online]. Available: https://p5js.org/. Accessed: Feb. 1, 2025.

[173] L. McCarthy, C. Reas, and B. Fry, Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing. *Sebastopol*, CA, USA: Maker Media, Inc., 2015.

[174] OpenFrameworks – Creative Coding Framework. [Online]. Available: https://openframeworks.cc/. Accessed: Oct. 29, 2022.

[175] Three.js – JavaScript 3D Library. [Online]. Available: https://threejs.org/. Accessed: Oct. 29, 2022.

[176] Cycling '74, *Max/MSP* [Computer software], Version 8.3.3, San Francisco, CA: Cycling '74, 2023. [Online]. Available: https://cycling74.com/products/max. Accessed: Feb. 1, 2025.

[177] M. S. Puckette, "Pure Data," in *Proceedings of the International Computer Music Conference (ICMC)*, 1997. [Online]. Available: https://www.researchgate.net/profile/Miller-Puckette/publication/230554908_Pure_Data/links/577c1cca08aec3b743366f5c/Pure-Data.pdf. Accessed: Oct. 29, 2022.

[178] Pure Data. [Online]. Available: https://puredata.info/. Accessed: Oct. 29, 2022.

[179] Derivative, *TouchDesigner* [Computer software]. Toronto, Canada. [Online]. Available: https://derivative.ca/. Accessed: Feb. 1, 2025.

[180] Epic Games, *Unreal Engine* [Computer software]. [Online]. Available: https://www.unrealengine.com/. Accessed: Feb. 2, 2025.

[181] Unity Technologies, *Unity* [Computer software]. San Francisco. [Online]. Available: https://unity.com/. Accessed: Feb. 2, 2025.

[182] Godot Engine Community, *Godot Engine* [Computer software]. MIT License. [Online]. Available: https://godotengine.org/. Accessed: Feb. 2, 2025.

[183] McNeel & Associates, *Grasshopper* [Computer software]. Integrated with Rhinoceros 3D. Seattle. [Online]. Available: https://www.grasshopper3d.com/. Accessed: Feb. 2, 2025.

[184] Robert McNeel & Associates, *Rhinoceros 3D* [Computer software]. Seattle. [Online]. Available: https://www.rhino3d.com/. Accessed: Feb. 2, 2025.

[185] SideFX, *Houdini* [Computer software]. Toronto. [Online]. Available: https://www.sidefx.com/. Accessed: Feb. 2, 2025.

[186] Blender Foundation, *Blender* [Computer software]. Amsterdam, Netherlands. [Online]. Available: https://www.blender.org/. Accessed: Feb. 2, 2025.

[187] Wolfram Research, *Mathematica* [Computer software]. [Online]. Available: https://www.wolfram.com/mathematica/. Accessed: Feb. 2, 2025.

[188] Northwestern University, *NetLogo Web* [Computer software]. [Online]. Available: https://www.netlogoweb.org/. Accessed: Oct. 29, 2022.

[189] J. Rampe, *Softology - Visions of Chaos* [Computer software]. [Online]. Available: https://softology.pro/voc.htm. Accessed: Oct. 29, 2022.

[190] A. Wuensche, *DDLab: Discrete Dynamics Lab* [Computer software]. [Online]. Available: https://www.ddlab.org/. Accessed: Oct. 29, 2022.

[191] A. Wuensche, "Discrete dynamics lab: tools for investigating cellular automata and discrete dynamical networks," *Kybernetes*, vol. 32, no. 1/2, pp. 77–104, Jan. 2003.

[192] Slackermanz, *VulkanAutomata: Cellular Automata GPU Renderer using the Vulkan API.* [Online]. Available: https://github.com/Slackermanz/VulkanAutomata. Accessed: Oct. 4, 2022.

[193] W. B. Slackermanz, "Understanding Multiple Neighborhood Cellular Automata," *Slackermanz - Exploring complexity through code*, May 23, 2021. [Online]. Available: https://slackermanz.com/understanding-multiple-neighborhood-cellular-automata/. Accessed: Oct. 4, 2022.

[194] *Neuralpatterns*. [Online]. Available: https://neuralpatterns.io/. Accessed: Oct. 29, 2022.

[195] *The life engine*. [Online]. Available: https://thelifeengine.net/. Accessed: Oct. 29, 2022.

[196] R. Rucker and J. Walker, *CelLab*. [Online]. Available: https://www.fourmilab.ch/cellab/. Accessed: Oct. 05, 2022.

[197] *LifeViewer.* [Online]. Available: https://lazyslug.com/lifeviewer/. Accessed: Oct. 29, 2022.

[198] L. M. Antunes, "CellPyLib: A Python Library for working with Cellular Automata," *Journal of Open Source Software*, vol. 6, no. 62, p. 3608, 2021. [Online]. Available: https://joss.theoj.org/papers/10.21105/joss.03608. Accessed: Oct. 29, 2022.

[199] L. Ben-Gai, *SEMICOLONY*, 2015. [Online]. Available: https://soog.bet/semicolony. Accessed: Feb. 2, 2025.

[200] Obsolete Studio, *The Infinite Bridge*, live performance, Royal College of Music, Britten Theatre, London, May 5, 2015. [Online]. Available: https://obsolete.studio/portfolio/the-infinite-bridge/. Accessed: Feb. 2, 2025.

[201] L. Ben-Gai, "utomata.js," *GitHub*, Feb. 12, 2019. [Online]. Available: https://github.com/soogbet/utomata. Accessed: Feb. 2, 2025.

[202] *The Coding Train*, "Coding Challenge #85: The Game of Life," YouTube, Dec. 11, 2017. [Online Video]. Available: https://www.youtube.com/watch?v=FWSR_7kZuYg Accessed Oct. 07, 2022.

[203] L. Ben-Gai, *Vital Signs - Red Nose Hexagliders* [Online Video], May 30, 2021. Available: https://www.youtube.com/watch?v=Nk3TiIMtFSs. Accessed Feb. 15, 2024.

[204] L. Ben-Gai, *Utomata Lab*, 2022. [Online]. Available: https://utomata.net/lab/ Accessed May 2, 2024.

[205] L. Ben-Gai and M. Tsadka, *Echosystem*, Apr. 2019. [Online]. Available: https://soog.bet/echosystem/ Accessed Jun. 29, 2024.

10. Appendix

10.1. Type-U Topologies

```
U(Vx.Cx, Vy.Cy)
```

First Order Type-U

```
U(
  Fx(Vx.Cx, Vx.Cx),
  Fy(Vy.Cy, Vy.Cy)
)
```

Second Order Type-U

```
U(
  Fx(
    Fx(Vx.Cx, Vx.Cx),
    Fx(Vx.Cx, Vx.Cx)
  ),
  Fy(
    Fy(Vy.Cy, Vy.Cy),
    Fy(Vy.Cy, Vy.Cy)
  )
)
```

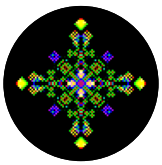
Third Order Type-U

```
U(
  Fx(
    Fx(
      Fx(Vx.Cx, Vx.Cx),
      Fx(Vx.Cx, Vx.Cx)
    ),
    Fx(
      Fx(Vx.Cx, Vx.Cx),
      Fx(Vx.Cx, Vx.Cx)
    )
  ),
  Fy(
    Fy(
      Fy(Vy.Cy, Vy.Cy),
      Fy(Vy.Cy, Vy.Cy)
    ),
    Fy(
      Fy(Vy.Cy, Vy.Cy),
      Fy(Vy.Cy, Vy.Cy)
    )
  )
)
```

Fourth Order Type-U

```
U(
  Fx(
    Fx(
      Fx(
        Fx(Vx.Cx, Vx.Cx),
        Fx(Vx.Cx, Vx.Cx)
      ),
      Fx(
        Fx(Vx.Cx, Vx.Cx),
        Fx(Vx.Cx, Vx.Cx)
      )
    ),
    Fx(
      Fx(
        Fx(Vx.Cx, Vx.Cx),
        Fx(Vx.Cx, Vx.Cx)
      ),
      Fx(
        Fx(Vx.Cx, Vx.Cx),
        Fx(Vx.Cx, Vx.Cx)
      )
    )
  ),
  Fy(
    Fy(
      Fy(
        Fy(Vy.Cy, Vy.Cy),
        Fy(Vy.Cy, Vy.Cy)
      ),
      Fy(
        Fy(Vy.Cy, Vy.Cy),
        Fy(Vy.Cy, Vy.Cy)
      )
    ),
    Fy(
      Fy(
        Fy(Vy.Cy, Vy.Cy),
        Fy(Vy.Cy, Vy.Cy)
      ),
      Fy(
        Fy(Vy.Cy, Vy.Cy),
        Fy(Vy.Cy, Vy.Cy)
      )
    )
  )
)
```

Fifth Order Type-U



LABORATORY OF BABEL

Lior Ben-Gai
February 2025