UNIVERSIDADE D
COIMBRA

Ricardo Rocha Sacadura

Towards Automated Generative Design

Dissertation in the context of the Master in Design and Multimedia, advised by Professor Penousal Machado and Professor Tiago Martins, and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

September 2023

Towards Automated Generative Design

Faculty of Science and Technology
of the University of Coimbra
September 2023

Master's Degree
in Design and Multimedia

Ricardo Sacadura
advised by Penousal Machado
and Tiago Martins

Abstract

Since the dawn of our civilisation, humans have sought means to express themselves creatively. Despite the vast set of different tools accessible throughout the years, this creative endeavour remains defined by the combination of two principles: to choose, making aesthetic and conceptual decisions within a solution space, and to diverge, exploring original, novel possibilities (reshaping the space). While these principles apply to most creative domains, from visual arts to music and architecture, the need to efficiently communicate ideas with a target audience accentuates their relevance to graphic design, as the quality/success of the design artefact relies on it.

The advent of computation and high-level programming languages has led to the creation of generative design models (in the present, many designers collaborate with machines to generate multiple design artefacts). While these models are programmatic in some cases, the results always depend on parameters and low-level decisions. Due to the number of parameters and consequent combinatorial explosion, designers only explore a small subset of possible solutions.

This dissertation describes the development of EvoProteus, a system that addresses this limitation by combining a generic generative design tool with an evolutionary algorithm. We intend to optimise the search through design spaces, creating a framework capable of granting some automation within generative processes and ultimately guiding designers to better design decisions, as it constitutes a critical part of their work. This project takes relevant steps towards an automated generative design paradigm.

Keywords

Generative Design, Parametric Spaces, Evolutionary Design, Genetic algorithms, Generic models.

Resumo

Desde os primórdios da nossa civilização que os seres humanos procuram meios de se expressar criativamente. Apesar do vasto conjunto de diferentes ferramentas acessíveis ao longo dos anos, esta empresa criativa permanece ancorada na combinação de dois princípios: escolher, tomando decisões estéticas e conceptuais dentro de um espaço de soluções, e divergir, explorando possibilidades novas e originais (reconfigurar o espaço). Embora esses princípios se apliquem à maioria dos domínios criativos, das artes visuais à música ou arquitetura, a necessidade de comunicar ideias com eficácia a uma audiência, acentua a sua relevância para o design gráfico, dado que a qualidade/sucesso do objeto de design depende disso.

O advento da computação e de linguagens de programação de alto nível levou à criação de modelos de design generativo (no presente, muitos designers colaboram com máquinas para gerar múltiplos artefatos). Apesar desses modelos serem programáticos em alguns casos, os resultados dependem sempre de parâmetros e decisões de baixo nível. Devido ao número de parâmetros e à consequente explosão combinatória, os designers exploram apenas um reduzido conjunto de possíveis soluções.

Esta dissertação descreve o desenvolvimento do EvoProteus, um sistema que responde a esta limitação combinando uma ferramenta genérica de design generativo com um algoritmo evolucionário. Pretendemos otimizar o processo de procura criativa, desenvolvendo uma framework capaz de conceder algum grau de automação em processos generativos e, em última instância, conduzir os designers a melhores decisões de design, o que constitui uma parte fundamental do seu trabalho. Este projeto dá passos significativos em direção a um paradigma de design generativo automatizado.

Palavras-chave

Design Generativo, Espaços Paramétricos, Design Evolucionário, Algoritmos Genéticos, Modelos Genéricos.

Agradecimentos

Aos meus orientadores,
por terem acreditado em mim,
por serem exigentes
e por me cultivarem o gosto de aprender.

Aos meus pais,
aquilo que sou hoje
devo ao vosso amor, dedicação e sacrifício.

À Mariana,
a razão da minha felicidade,
és tudo.

Dedico este trabalho à memória
do meu avô Amílcar.

Se algo me vale a genética,
é a herança do teu infindável
amor pelo conhecimento.

List of Figures

Table of contents

CHAPTER 1.

INTRODUCTION

A great ancient tradition conceives the visual arts, and images in general, as a language, revealing and embodying spontaneous human creativity. Barasch (1997) defines the artist's creative practice as a universal impulse. The most common instance is the urge to perform representative gestures in moments of intensive perception or imagination. These expressions are forms of communication and a primordial method for visually articulating events and emotions. Art then conquers what language aims to arrive at, replacing the confused world of sensual impressions with a limited but distinct grammar of forms (Barasch, 1997, pp.20-23). According to Meggs (1992), graphic design shares this "universal language of form" yet has a specific goal. While a painter, for instance, may not be concerned with how the work of art will impress the spectator, nor with what kind of feelings it will evoke (Barasch, 1997), the designer's goal is to solve problems, organise space, and permeate his work with novel[1] visuals and symbolic qualities to convey information from his own individual expression. Unlike other art forms, a design artefact exists in a context determined by the content it has to communicate, where functionality is crucial (Önduygu, 2010). This need for functionality requires making decisions from a range of possible solutions. Till recent times, this process implied using a set of limited tools. The archetypal design medium is a pencil, eraser and paper. Designers select materials, add elements, remove them, try a different composition and repeat the process. With the advent of computation, more sophisticated digital design tools (like Adobe software) are essentially emulations of this secular means of work (Woodbury, 2010, p. 11).

1. Novelty, in this context, means the quality of being new, original and different (Mcormack et al., 2004).

Nowadays, the design method is shifting as our world is increasingly invaded and mediated by electronic systems and devices. First, there has been an increased interest in "collaborative, interdisciplinary approaches to design problems" (McCormack et al., 2004). A more intimate relationship exists between ideas and concepts "through the flexibility introduced in design methodologies" (McCormack et al., 2004). This shift has been accompanied by a programmatic perspective of graphic design, where designers do not create a visual outcome but instead establish a set of rules and program computational systems that will. In this scenario, computers become creative partners in design decisions (Silva-Jetter, 2012). Writing code or using programming languages has brought new expressive possibilities to this discipline. A new generation of designers uses programmatic approaches by building their own programs (Shim, 2020).

1.1 Motivation

Given the previous context, in the present, countless artists and designers use computation to write programs that generate visual artefacts. These artefacts often result from mapping input parameters to visual parameters (Shim, 2020). Using parameters requires some decisions that affect the outcome, such as the range of variability (predict randomness level), relationships between variables, and value assignment. Parametric generative models grant designers increased autonomy in their work as they can shape rules and variables to attain their own design language from conception to execution.

However, while using these models, many designers still produce numerous variations of an artefact by hand, either because it is a fundamental part of their exploratory design process or because they must "manually" test different parameterizations until they find a suitable combination for their intentions. In the end, regardless of whether it is computational or not, the final form of the artefacts always depends on a set of parameters and low-level design decisions that the designer defines implicitly or explicitly. Due to the number of parameters and consequent combinatorial explosion, the designer only explores a small subset of possible solutions.

1.2 Goals and Implications

As more and more designers use code to create visual artefacts by manipulating parametric spaces, it is essential to find ways to optimize this process. That is the focus of this dissertation. We aim to create a computational tool to assist designers in exploring design solutions more efficiently. Addressing this goal involves attaining two sub-goals. The first is (a) to identify the list of parameters used in the generation of any artwork and build a solution space for that artwork. The second (b) is to guide designers through that space and help them find the most adequate solutions for their purposes.

These intentions translate into the development of a generic system capable of generating multiple artefacts from a single source code and the integration of a genetic algorithm to accelerate the search for interesting outcomes.

We will research definitions and relevant work in graphic design, generative design, and evolutionary design to accomplish these objectives. Then, we will create a framework structuring our intentions, and develop a system. testing it in distinct contexts. This work may represent significant progress in designers' decision-making methods, taking relevant steps towards the automation of generative design processes.

1.3 Document Overview

This document is structured into nine chapters: (a) Introduction; (b) Work plan and methodology; (c) Literature Review; (d) Framework proposal; (e) Proof of concept; (f) System development; (g) Evaluation and results; (h) Future work and (i) Conclusion.

In the opening chapter (a), we situated our dissertation within a scientific research zone we believe is still unexplored and described the issue we plan to tackle through this work. We also outlined a series of objectives that we believe will lead us to relevant results. The upcoming chapter (b) sets the work plan for the dissertation up to the final submission and the methodology that will guide it. The Literature review chapter (c) describes the research on the history and definitions of graphic design, generative design with parametric approaches and evolutionary systems (especially biology-inspired ones). We trace relevant work in these fields, identifying some limitations we expect to respond with our project. In the Framework proposal chapter (d), we consolidate our response to those limitations, presenting some models to structure and scientifically support the development process. Following proposal (e), we describe an initial implementation of the first framework's modules to prove our concept and gather insights for system development. We then present our system (f), discussing the integration of the remaining modules, technical challenges, refinements and considerations. The Evaluation and Results chapter (g) outlines the different testing levels our system underwent and presents the most encouraging results as well as some reflections. In Future work (h), we present further developments and ideas for our system. To conclude the report (i), we summarize the work done and discuss the results achieved both in theory and practice.

CHAPTER 2.

WORK PLAN AND METHODOLOGY

This section presents the work plan defined for this dissertation and the methodology chosen to guide it.

2.1 Tasks identification

In the project's early stages, we drew a plan outlining the required tasks until submission. This plan included the creation of a prototype, research, framework proposal and implementation, experimentation, validation, refinements and report writing. We created a Gantt chart (depicted in Figure 2.1.1.) to situate these tasks in time. However, after some analysis and reflection, we later restructured the plan to attain the necessary adjustments in task definitions and associated timespans. The actual plan that will guide our work encompasses the following tasks:

I. Research on graphic design. Research on the graphic design field. Definitions, history, design tools, programmatic perspectives in the discipline and relevant work.

II. Research on generative design. Research on the generative design field with emphasis on parametric approaches. Definitions, history, common techniques and relevant work.

III. Research on evolutionary design. Research on evolutionary design, with a special focus on genetic algorithms. Definitions, theoretical foundations, potential design applications and relevant work (as this task is the last instance in the literature review, we will also express some considerations regarding our concept).

IV. Framework proposal. A complete description of the framework proposal structuring system implementation, goals, pipeline & architecture, scientific approach and technologies to be used.

V. Proof of concept / Early Prototype. Developing an early prototype to prove our concept and incorporate the first crucial framework components. These tasks are subdivided into three sub-tasks: (a) parameter identification, (b) generation of variations and (c) initial interface design.

VI. System development. The system implementation as a Processing design tool for optimized solutions search. This task includes the integration of the remaining components, it divides into four sub-tasks: (a) genetic operations, (b) fitness assignment, (c) interface conclusion and (d) initial results.

VII. Evaluation and Results. System evaluation and results presentation. We will branch the evaluation into five different stages. (a) internal evaluation, continuously comparing the system's performance with our goals (this evaluation is implicit through the entire development process); (b) system's broadness assessment, to test its ability to operate with code from various sources with distinct characteristics; (c) system's relevance assessment, to test its relevance in solving design problems; (d) user evaluation, to test the interface with users and (e) experiments in a different environment beyond the regular computer screen execution.

VIII. Refinements. Identification and execution of minor improvements on the system. Interface refinements.

IX. Writing the dissertation. Documentation of the whole process, including goals, work plan, preliminary research, evolution of implementation over time, results and conclusions. This report serves as a comprehensive record of the project, presenting different stages of development, errors, achievements and results. Figure 2.1.2 illustrates all task timespans in a new Gantt chart.

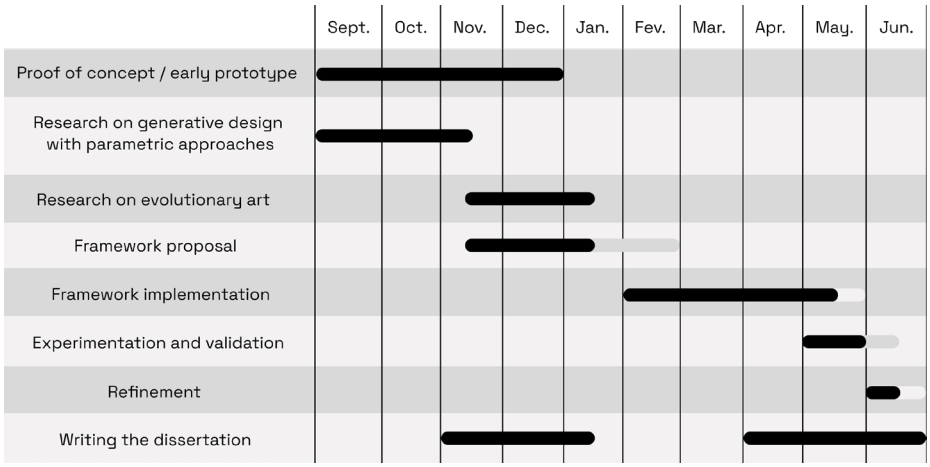


Figure 2.1.1.

Initial work plan's Gantt chart. The light grey areas correspond to predicted delays.

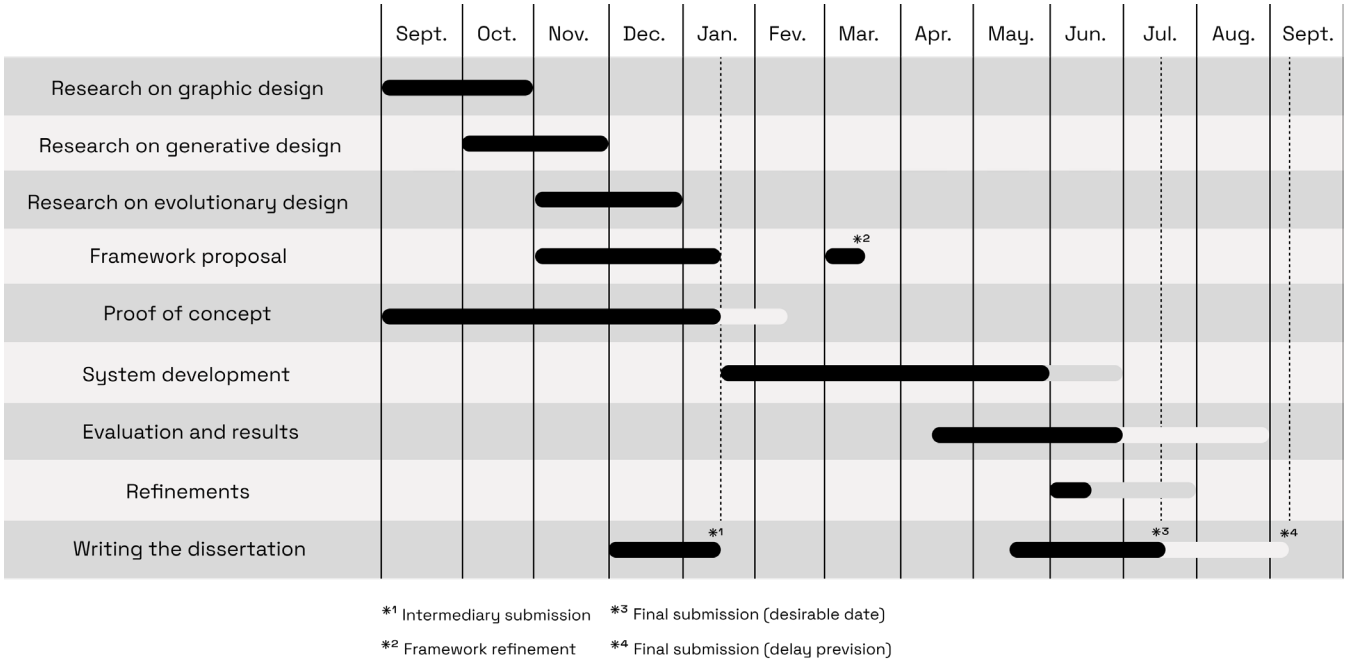


Figure 2.1.2. Actual Gantt chart locating all tasks in time
(we include a possible final submission extension to September).

2.2 Methodology

The methodology applied will follow Nigel Cross's "Four Stage Design Process". This descriptive model consists of four stages: (a) exploration, (b) generation, (c) evaluation, and (d) communication. The design process begins with an (a) exploration and comprehension of the "ill-defined problem space" Cross (2000). This is followed by a cyclical methodology that comprises two distinct yet interrelated stages: the (b) generation of a design concept and the (c) evaluation of the proposed design against established objectives, constraints, and metrics. The iteration of these two stages continues until the designer reaches a consensus that the design proposal has attained its full potential. Then, the process culminates in the formal (d) communication of the final artefact to the relevant stakeholders (Cross, 2000). This model is visually demonstrated in Figure 2.2.1 and was chosen for its minimalistic approach attaining directly what Cross (2000, p.29) defines as "the essential activities that the designer performs".

The dissertation employs Cross's process heuristically, meaning we had to adapt it to our own requirements. Here, Cross's Exploration step is equivalent to the Literature review. Then, we employ the Generation and Evaluation cycle in two different moments. The first loop involves prototyping and composing the Framework proposal. The prototype development supports the proposal, while the proposal offers insights into the technical aspects of development. When satisfied with the framework's state and initial results, we move on to the next cycle. This time, with the knowledge acquired previously, we begin to work between system implementation and evaluation, which will guide us through system refinement. The final report corresponds to Cross's Communication stage and documents the entire process. Figure 2.2.2 illustrates this adaptation.

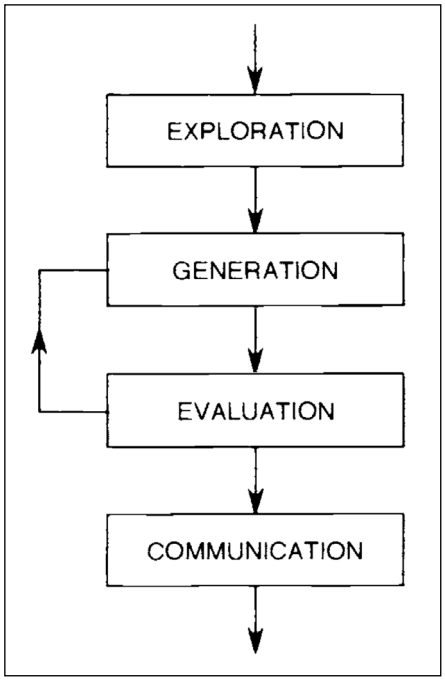
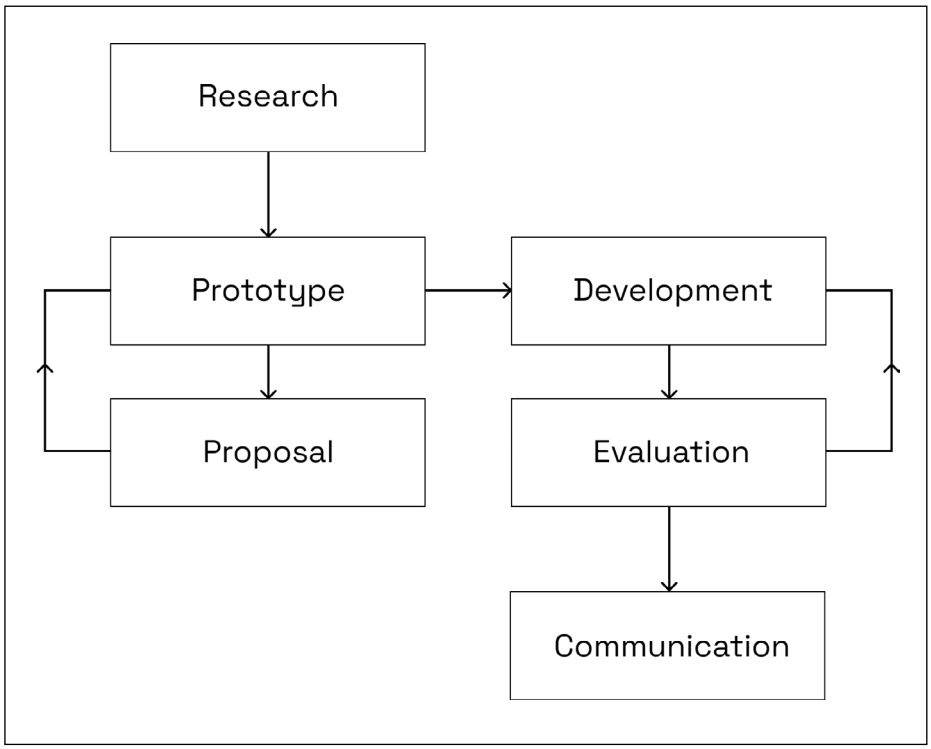


Figure 2.2.1.

A simple four-stage model of the design process. (Extracted from Cross's book *Engineering Design Methods*).

Figure 2.2.2.

Our adaptation of Cross's method,
with two separate iterations.



CHAPTER 3.

LITERATURE REVIEW

3.1 Graphic Design

I. History.

Until the 19th century, a graphic designer was primarily a printer who used woodblock types and illustrations in a letterpress, creating monochromatic books and posters (Hollis, 1994). With the advent of chromolithography (a technique in which a designer paints a visual artefact transposing it by hand to stone or metal surfaces), designers were able to print artwork with different colour tones (Galvan, 2020). This method was employed through several artistic movements, bringing new perspectives to the design field. One example of its impact is latent in the Art Noveau era, where designers, seen as skilled craftsmen, applied chromolithography in the form of organic lines and elegant shapes to decorate spaces or materials (Meggs & Purvis, 2006).

At the turn of the century, the modern movements in visual arts, particularly the Bauhaus school (founded in 1919), redefined this discipline by introducing ideas about form, geometry, colour, and space into the design language, recognizing for the first time design's potential to solve social problems (Meggs & Purvis, 2006). This led to the invention of novel design tools, such as photomontage and new type design approaches (Meggs & Purvis, 2006).

During the 1950s, "The International Typographic Style" (the embodiment of modernism) provided designers with a framework of logic and structure based on grid systems that allowed for systematic control of visual elements in uniform layout schemes (Meggs & Purvis, 2006), taking the first steps towards a programmatic approach to design (Figure 3.1.1.) (Hollis, 1994). This modernism modularity concept consisted of generating variations, iterated within predefined constraints to grant a system's flexibility. These constraints enabled the designer to compose visual artefacts with hierarchy and composition concerns. Karl Gerstner, a Swiss designer (1930-2017), took this perspective even further. Gerstner emphasized the system and its rules as an element of primary importance. He redefined design as a program "or the process of building, selecting, and combining parameters" (Shim, 2020, p.3). To the author, this approach was more about a designer's logical thinking over the technical process of writing code for algorithms.

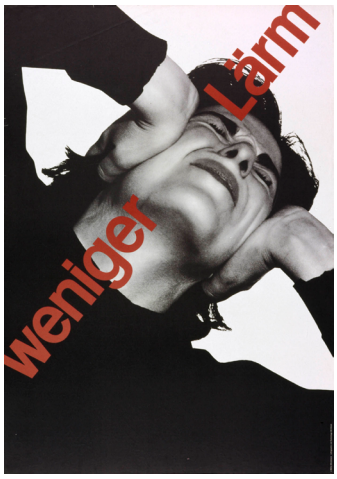


Figure 3.1.1.
Weniger Lärm poster by Josef Müller-Brockmann (1960).

Even so, it would play a fundamental role in the evolution of computer programs for graphic design purposes.

A good example of Gerstner's method is his visual identity project for *Boîte à musique* record shop in Basel, Switzerland. In this work, he designs several instances applicable to different required formats. In Figure 3.1.2, the rectangular frame is defined as a variable parameter, the proportion of which is altered depending on the available space. While containing visual consistency, the project also cares for personality. In a second instance, Gerstner added an element of playfulness. The tension generated by trying to be functional and playful simultaneously turned *Boîte à musique* recognisable while still consistent (Hewitt, 2008) (Figure 3.1.3).

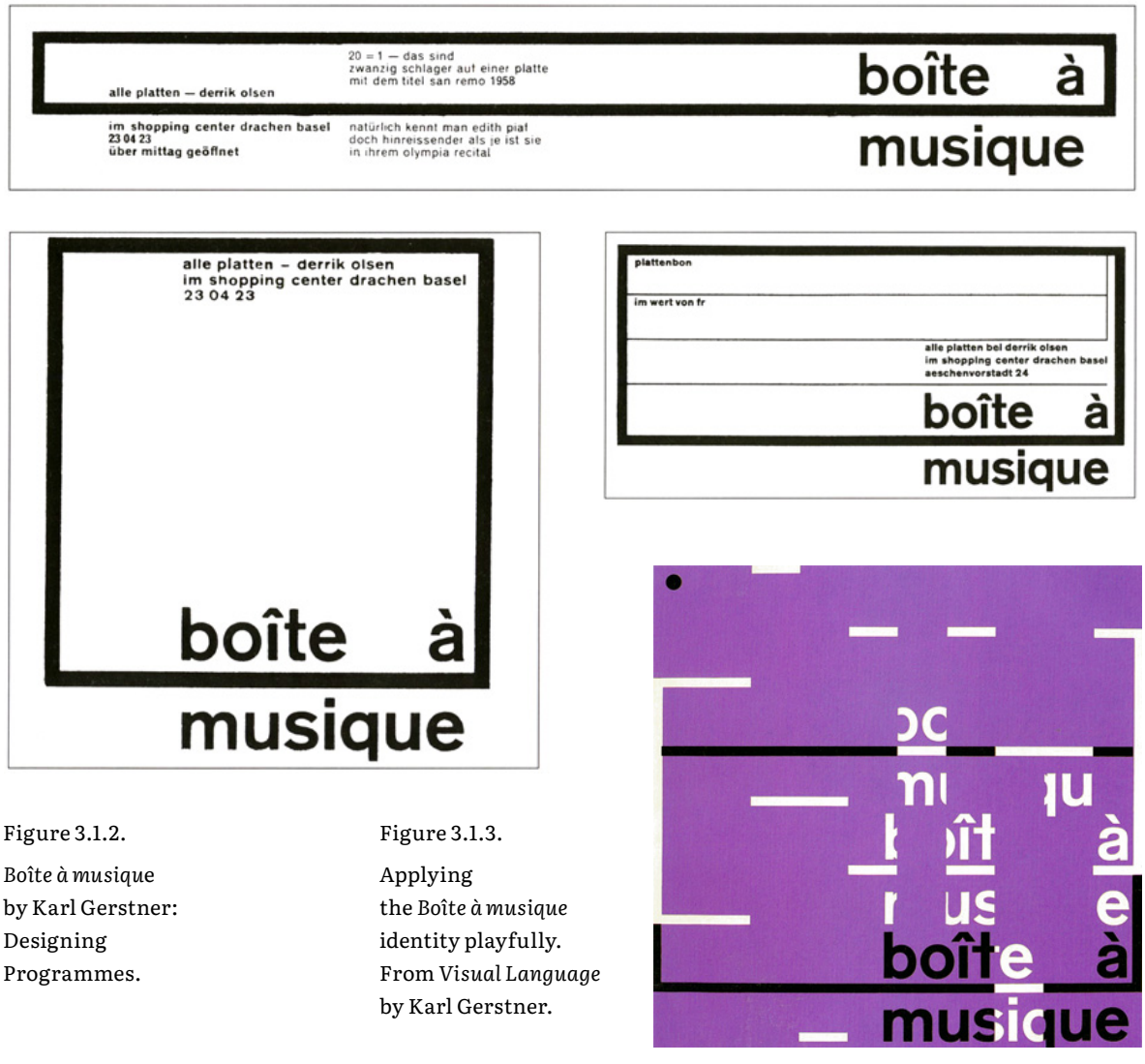


Figure 3.1.2.

Boîte à musique by Karl Gerstner: Designing Programmes.

Figure 3.1.3.

Applying the *Boîte à musique* identity playfully. From *Visual Language* by Karl Gerstner.

These 20th-century movements were accompanied by continuous progress in design resources, such as the proliferation of economic paper formats and the rise of the phototype technique. Meggs & Purvis (2006) explain how this method fragmented the discipline of graphic design into "a series of specialized steps".

> *After phototype became prevalent (...), skilled specialists included graphic designers, who created page layouts; typesetters, who operated text and display typesetting equipment; production artists, who pasted all of the elements into position on boards; camera operators, who made photographic negatives of the pasteups, art, and photographs; strippers, who assembled these negatives together; platemakers, who prepared the printing plates; and press operators, who ran the printing presses* (Meggs & Purvis, 2006, p.353).

In the late 20th century, digital technology, especially digital computer software, aggregated all of these specialized subfields into one, empowering the designer with the ability to carry out computer-aided design (CAD) processes with unprecedented creative potential (Meggs & Purvis, 2006). Postmodern designers rejected the formal structures established by modernism and soon incorporated principles of mixed media and deconstruction into their own work by embracing technology (Levanier, 2022). Two of the first artists to master these computer-made pieces were April Greiman (Figure 3.1.4.) and Paula Scher (Figure 3.1.5).

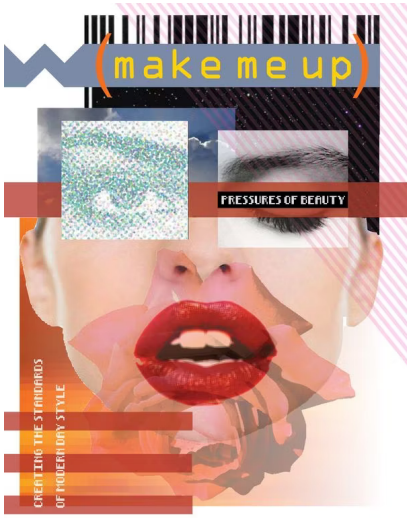


Figure 3.1.4.

"Make Me Up" poster by April Greiman.

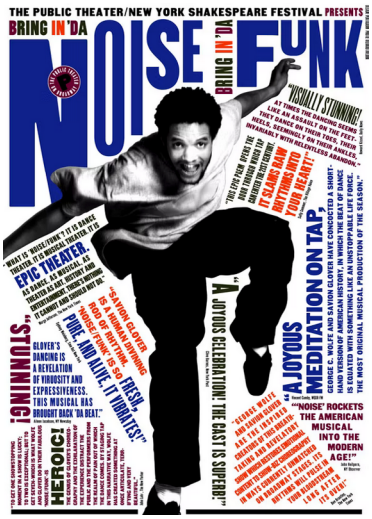


Figure 3.1.5.

"Bring in 'da Noise, Bring in 'da Funk" poster by Paula Scher.

Most of this software consisted of commercial tools focused on accessibility and usability (for example, Adobe Photoshop, PageMaker or FreeHand). The widespread adoption of these digital tools has revolutionised how we design, and it continued to gain momentum through the early 21st century years. However, as digital literacy among young designers and artists proceeded to rise, there was an increasing desire to break free from these tools' limitations (as they do not encourage "programmatic perspectives in graphic design" (Shim, 2020, p.1)). This turning point began in the late 1990s when Gerstner's perspective was revisited by some designers who were applying computational approaches to graphic design.

II. A programmatic perspective.

John Maeda (technologist and designer) created "Design by Numbers" (DNB), which Silva-Jetter (2012) describes as "a programming language intended for people who do not know how to program". These targeted users could be designers and artists (instead of engineers) who were incited to explore the computational possibilities associated with their code writing. Maeda himself pointed out the essence of his project - "a medium for mapping input parameters to visual parameters" (Shim, 2020, p.7). He highlighted the key role parameters play in programming, as they enable the customization and optimization of input and output processes, thereby revealing the underlying potential of computation in the design field (Shim, 2020). Figure 3.1.6 gives a glimpse of Maeda's work.

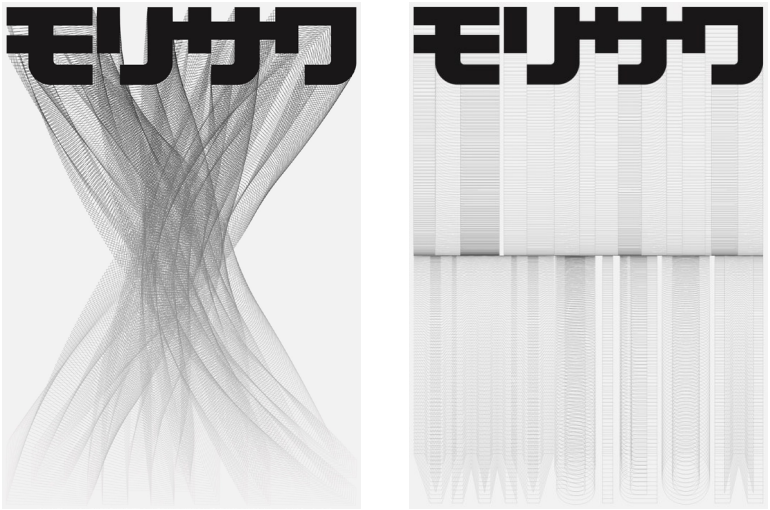


Figure 3.1.6.
"Morisawa" posters [1] and [6] by John Maeda (1996).

Later, Ben Fry and Casey Reas found Processing [2] within Maeda's "Aesthetics and Computation" research group. A high-level programming language made directly on top of DNB. Compared to its precursor, Processing has had a greater impact on the design community. Currently, it is widely adopted by many visual designers, artists, and architects to create their works. Furthermore, Processing has been instrumental in fostering programming integration as a core aspect of the creative process among a new generation of artists (Fry & Reas, 2022). More recently, a new platform has emerged as another foundation for designers to explore rule-based programmatic approaches in their creative work. OPENRNDR[3] is an open-source platform for creative coding that combines art and design elements with software functionalities. Its iterative process of coding sketches allows for discovering and developing new ideas. The platform is versatile and can be used for sketches and interactive media installations. It's also optimized to handle real-time data and is suitable for dynamic data visualization and interactive installations (Figure 3.1.7).

This leads us to recent years, in which the designer's role is not necessarily creating a visual outcome but rather forming rules and programming systems that will (Silva-Jetter, 2012). Nowadays, graphic designers write their own code to attain specific problems, navigating through a solution space built from parameters and constraints. Silva-Jetter (2012) gives an example. "Typically, the design of a visual identity consists of static imagery that is not intended to change. In this case, the visual identity is constantly invigorated and is always different. It is constantly re-computed and regenerated. In this way, it is more related to its inherently digital nature" (p.356). We are witnessing a transition from a computer-aided design paradigm to a "programmatic-assisted design", as Erik van Blokland and Just van Rossum describe in Shim (2020, p.6). This shift in the discipline has helped develop what McCormack et al. (2004) refer to as "Generative Design Culture", defined by a growing taste in collaborative design across different fields and the use of digital tools to bridge the gap between concept and final execution. This has led to the integration of models that do not only map predefined inputs to visuals but also "free designers from 'design fixation' and the limitations of conventional wisdom, thereby allowing them to explore a huge number of possible proposals for a design problem" (Janssen et.al., 2002, p.119), inhabiting new properties that may surpass the designer's expectations. These parameter-based models have facilitated the encounter of designers and artists with generative design practices (McCormack et al., 2004).

2. **Processing,** processing.org

3. **OPENRNDR,** openrndr.org

Figure 3.1.7.

Samples of Generative× Data-driven Posters made during the Workshop by OPENRNDR × Politecnico di Milano (February 17-21).

3.2 Generative Design

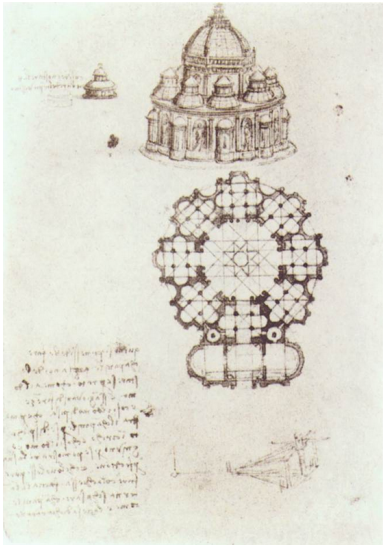
I. Precursors of Generative design.

The history of generative design relates to the first algorithmic combinations of goals and constraints in order to reveal solutions, making it also a history of design methodology. (Li, 2019). Before the 20th century, this methodology was primarily associated with architecture. In his doctoral thesis, Martins (2021) presents several examples. We outline three of them.

One notable instance given by the author dates back to the 15th century and can be observed in the (a) combinatorial methods utilized by Leonardo Da Vinci when designing central-plan churches (Figure 3.2.1). Da Vinci began with fundamental spatial shapes, such as squares, octagons, and circles, and then methodically integrated circular, semi-circular, or octagonal spaces into the design according to pre-established geometric principles. This allowed him to generate every possible central-plan church design with minimal effort or imagination (p.22). In the 18th century, a (b) German landscape game was developed as an example of design systems recombination (Figure 3.2.2). The game includes 24 cards that can be arranged horizontally in any order to create landscapes of different sizes and compositions. By exploring different permutations of the cards, players could build various continuous landscapes (pp.22-3). In 1926, the famous Swiss-French architect Charles-Édouard Jeanneret (also known as Le Corbusier) came up with (c) a set of principles that he believed defined his unique architectural style. These principles are described in his book, *Les Cinq Points d'une Architecture Nouvelle* (The Five Points of a New Architecture). Le Corbusier's five points were: using pilotis to lift the building off the ground, allowing for free design of the ground plan, allowing for free design of the façade, incorporating long horizontal windows and including roof gardens. A good example of a building that incorporates all of these five points is Villa Savoye. Le Corbusier's principles go beyond describing an architectural style. They also provide clear guidelines for designing a building. These points are an example of generative systems (p.24).

Figure 3.2.1.

Study of a central church by Leonardo Da Vinci (1488), Milan, Italy.



During the early 20th century, Karl Gerstner was among the first artists to bring programmatic approaches to design. His *Boîte à musique* project is a notable example of this (as shown in the previous segment). With the rise of computing technology, many people became interested in exploring its potential for generating visual art. This led to a new era of experimentation with algorithmic artistic creation. For instance, in 1964, Philip Peterson used automatic methods to represent digitized images by scanning an input image and converting those values into a grid of symbols. An example of this process is the digital recreation of "Mona Lisa" (Peterson, 1965). This algorithmic trend applied in visual domains extends to John Maeda's design programs and current generative design practices.

Figure 3.2.2.

Myriorama, A Collection of Many Thousand Landscapes, Designed by Mr. Clark, *16 hand-coloured aquatints mounted on thin strips of card* by Samuel Leigh (1824).

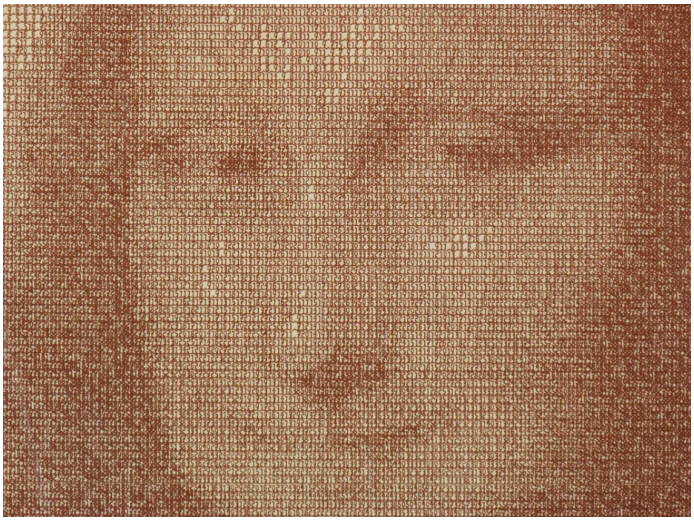


Figure 3.2.3.

The Digital Mona Lisa by H. Philip Peterson (1965).

II. Current definitions.

This segment delves into some contemporary definitions of generative design applied to visual domains. While Galanter (2003, pp.4) proposes a definition for generative art, his description may be adapted to graphic design. He refers to it as any practice where the artist uses a system, such as a set of natural language rules, a program, a machine, or other procedural invention, which is then set into motion with some degree of autonomy, resulting in complex artworks. Rodenbröker (2022) defines generative design as a method that employs the use of algorithms to transform an input (such as data or user interaction) into an output in the form of a new medium with multiple variations. This involves the development of systems with a set of parameters and constraints that serve as the foundation for generating visual outcomes in a broad sense. Normally, these systems are user-driven, where the user actively controls the generation process by directly manipulating a parameter space (Rodenbröker, 2022). According to Gradišar et al. (2022), generative design is an approach that facilitates collaboration between the designer and a computer algorithm, a method empowered by the complementary capabilities of both parties, with the computer being able to efficiently sort through data, generate and evaluate numerous solutions, and achieve optimal results through iterative refinement. On the other hand, the designer brings expertise in the form of real-world experience, in-depth knowledge and understanding of the field. Within this definition, the emphasis is shifted from generating the design solution to specifying the design problem with its constraints and criteria. This allows for the use of various algorithms to produce a wide range of solutions from which the designer selects the most suitable ones.

Upon these definitions, we list some key aspects of generative design applied to the visual arts. (a) Generative design is often a collaborative practice between a designer and a machine. (b) It transforms an input into a visual output, where the input is usually parameter-based and (c) produces a wide range of design solutions. For a deeper understanding of parameter-based generative design concepts, we studied Robert Woodbury's "Elements of Parametric Design" (Figure 3.2.4).

III. Parametric design.

Woodbury (2010) provides a comprehensive overview of the various factors involved in parametric approaches, which is highly relevant to generative design practices. On p.11, he explains how a parametric approach differs from traditional design methods. Rather than adding and removing visual elements, parametric design involves relating and changing these elements in a connected manner. This coordination requires the designer to consider how these relationships affect the outcome. Later on, on p.39, Woodbury defends that using parametric modelling in design allows for new possibilities, particularly in the realm of form. While, from the author's perspective, some may view this exploration as aimless and without direction, taking a step back shows that there is a serious purpose behind it. He says that throughout history, design has constantly evolved through the exploration of new ideas and concepts, using the available tools, stating that as new design languages and styles emerge, this exploratory approach becomes even more important.

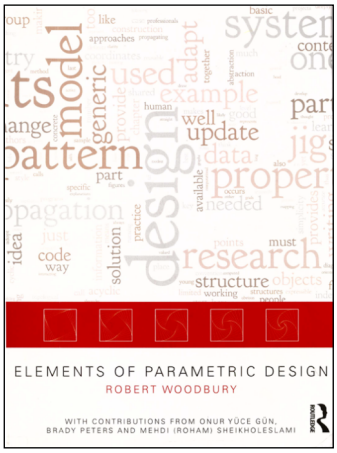


Figure 3.2.4.
"Elements of Parametric Design" by Robert Woodbury, book cover.

Two critical aspects of parametric design are the concept of parameter and the concept of design space. On p.50, a parameter is described as a variable, a named container that holds a value. In parametric design, variables can hold multiple values. The variable name is written first, followed by the value it contains and the type of object it represents. Parametric design allows us to describe a design as a collection of values. However, parameters do not impose any particular order, and a collection of parameters with no duplicates can be considered in any sequence without affecting the values held within those variables. Different combinations of variable settings constitute the design space. To explain the idea, the author divides it into two categories: implicit design space and explicit design space. The implicit design space refers to all possible design solutions that can be reached through a symbol system. It is a network that includes all solutions that a designer may or may not explore. On the other hand, the explicit design space is a smaller portion of the design space that includes only the states that have been explored in the current or previous design moments. The explicit space is developed through the design process and reflects the exploration behaviour of designers based on their cognitive limits and knowledge. Both implicit and explicit design spaces are interconnected, with the

latter being a subset of the former (Woodbury, 2010, pp.275-6). To the author, exploring multiple alternatives can lead to better designs. To Woodbury's assessment, we may add Krish's (2010) idea of an exploration envelope limiting the design space, meaning the definition of minimum and maximum values each variable may assume. This grants designers better control over the range of possible solutions.

Before the end of this segment, we emphasize some strategies Woodbury believes designers employ while using parametric approaches. The first one is language. Most current systems require a scripting language, which is a type of programming language. However, according to the author, designers must switch from their familiar visual and interactive representation to working with textual instructions (p.35). The second strategy is sketching, which is highly valued in design. Parametric models are dynamic and can easily be changed to answer design questions, favouring this secular design practice (p.36). Another strategy is coined by the author as *copy-and-modify*, which involves using existing code to achieve a particular effect. This requires a community of practice that generates code, similar to how web designers often mine existing pages for code snippets (p.38).

IV. Generative design projects.

After this concept revision, we analyze recent projects that are relevant to parametric, generative design approaches. In 2008, LUST Design Studio created the Poster Wall for the 21st Century for the Graphic Design Museum in Breda, Netherlands. The museum installation, "100 Years of Graphic Design in the Netherlands," featured a digital and physical poster wall (shown in Figure 3.2.5) that automatically generated numerous distinct posters using content from various internet sources (Manaranche, 2014). In 2017, MuirMcNeil Design Studio employed a systematic approach to create 8,000 distinct covers for Eye magazine's 94th edition (shown in Figure 3.2.6). They used the HP Mosaic program, which allows the generation of various unique compositions through a collection of input designs called "seed" image files. For this project, the seed images were typographic compositions that repeated the letters of the word 'eye' in fixed intervals, using different fonts from their TwoPoint and TwoPlus type families in three layers (Martins, 2021). MuirMcNeil established specific rules within the Mosaic program to adjust each seed

file's sections' scale, position, crop, and colour, resulting in thousands of covers (McNeill and Muir, 2017). André Burnier's 2020 visual identity for CIDDIC, a research centre at the Faculty of Contemporary Music at Unicamp, explores user-driven generative systems (Figure 3.2.7). Burnier utilized a "designer bot" created with p5.js to facilitate the generation of graphics (accessible to everyone) for the research centre. The bot combines parametrized visual elements with textual inputs, resulting in the generation of unique and coherent artefacts. This example demonstrates the creative potential of parametric approaches in creating visual identities (Burnier, n.d.). The Munken Creator was created by The Munken Paper Mill in partnership with Patrik Hübner in 2022. It's a web-based tool that allows users to create customized and visually appealing artefacts using the Munken Sans typeface (Figure 3.2.8). The tool has an interactive user interface that updates generated visuals in real time. It allows users to make changes and adjust their designs by manipulating a settings list, such as the font size, letter spacing or number of columns. The generated designs are rendered on screen and stored in the platform's URL. "So users can bookmark a design they like, save it for later or share their settings with others by sending them the URL" (Munken Paper Mill & Hübner, 2022).

(a)

(b)

Figure 3.2.5.

Two images of Poster Wall for the 21st Century for the Graphic Design Museum in Breda, Netherlands by LUST Studio. (a) Installation, (b) Posters.

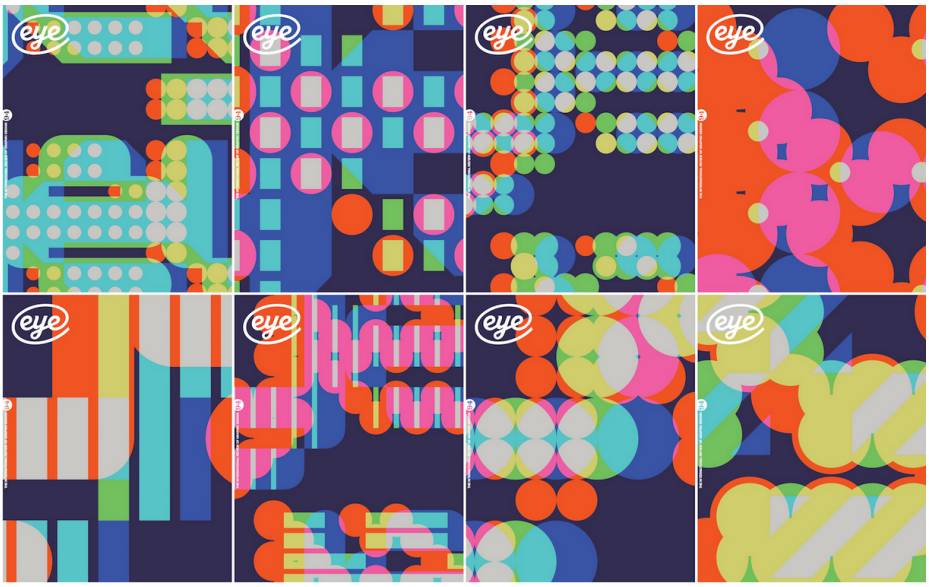


Figure 3.2.6.

Covers of the Eye magazine (issue 94) by MuirMcNeil Design Studio (2017).

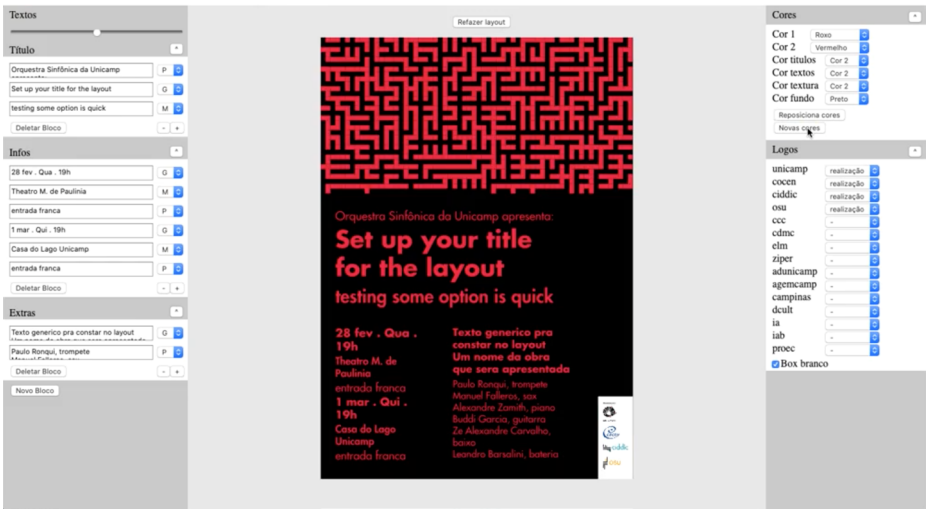


Figure 3.2.7.

"The Designer Bot's" by André Burnier (2020). Interface.

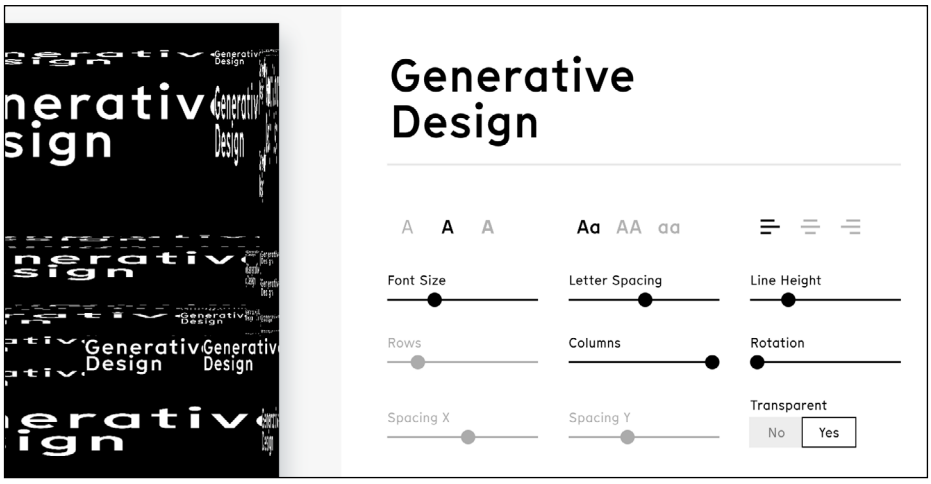


Figure 3.2.8.

Exploration of 'Generative Design' with The Munken Creator (2022). colab.munken.com

These projects reinforce the feeling of an emergent generative design culture that is becoming more prevalent in contemporary design practices. By systematically using computational tools and methods based on rules and parameters, designers have been exploring alternative approaches to the field. According to McCormack (2004), this culture promotes ideas of evolution, breeding and adaptation, making generative design a territory that sparks new possibilities beyond "simple" generative models. Evolutionary concepts are becoming part of the designers' vocabulary. As well as being generated, design artefacts may also be evolved and optimised to suit specific purposes. In the next section, we will analyse concepts and relevant work related to evolutionary approaches in design and determine how they may fit our goals.

3.3 Evolutionary Design

> (...) *stuck at that point just before the creative leap. They know where they have come from and have a general sense of where they are going, but often do not have a precise target or goal.*(Janssen et. al, 2002, p.120)

According to Bentley (1999, p.5), Evolutionary design applies evolutionary systems to the arts and graphic design field. In computer science, the evolutionary computation process involves searching for the best solutions to a computational problem. The search space is a space filled with all possible solutions. Each point in that space defines a solution. The goal is to improve parameter values by searching for better solutions within a solution space. This conception corresponds to Woodbury's (2010) implicit design space. To Bentley, there are distinct search algorithms, including evolutionary search, a rapidly growing subset. These algorithms are inspired by Darwinist biological evolution, using a natural evolution method to evolve solutions to problems by considering a large population of solutions at once rather than working with one solution at a time in the search space.

Richard Dawkins was among the first to apply evolutionary algorithms to the visual domain. He created The Blind Watchmaker program in 1986 (Figure 3.3.1.) to explore the artificial evolution of biomorphs through tree-growing procedures with a human-guided selection criterion. (Dawkins, 1986). In the early 1990s, Karl Sims and William Latham collaborated with Stephen Todd to create artistic images by combining evolutionary techniques and computer graphics, following Dawkins's track. As seen in Figure 3.3.2. Sims employed user-guided evolution to evolve abstract images, three-dimensional shapes, and animations, while Todd and Latham presented evolutionary techniques for creating biomorphic forms (Martins, 2021). Their work contributed heavily to the emergence of evolutionary art (Lewis, 2008).

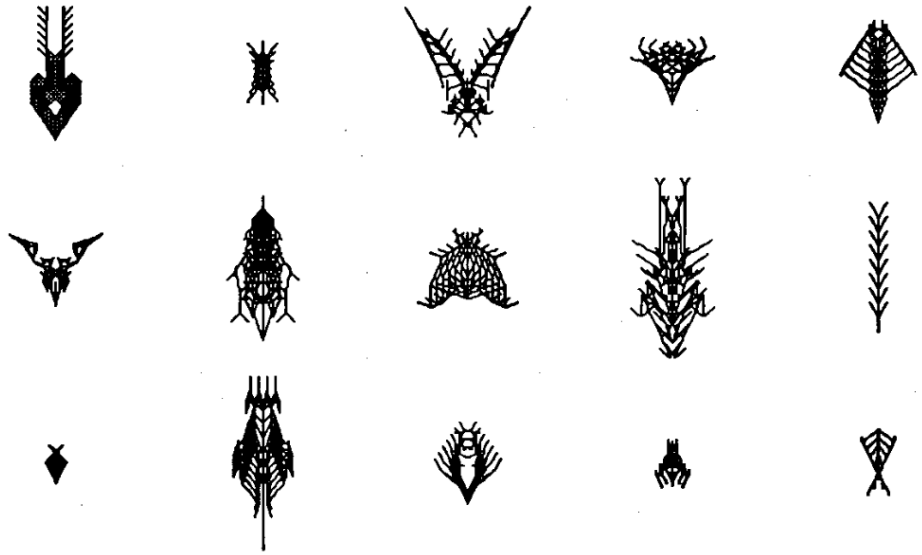
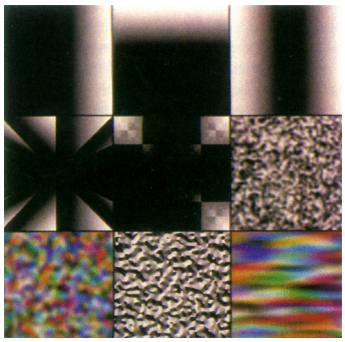


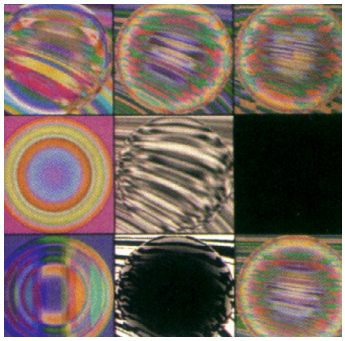
Figure 3.3.1.

Evolution of computer biomorphs by Richard Dawkins (1986).

(c)



(a)



(b)

Figure 3.3.2.

Three artworks by Karl Sims. (a) “Simple expression examples”. Evolution of images.
(b) “Parent with 19 random mutations”. Evolution of volume textures.
(c) “Fire of Faces”. Evolution of animations.

I. Genetic algorithms, revision of concepts.

There are four types of evolutionary algorithms: genetic algorithms, genetic programming, differential evolution, evolution strategy and evolutionary programming (Slowik and Kwasnicka, 2020). This segment focuses on genetic algorithms, widely used for a broad range of optimisation problems.

A genetic algorithm is a concept that draws upon the Darwinist principles of biological evolution. Such an algorithm creates populations of individuals representing potential solutions to a given problem (Holland, 1992). The goal is to improve upon these solutions through a search process. Each solution is represented by a string of numbers of a predetermined length (Janssen et al., 2002, p.120). This string is referred to as a genotype. Each component within the genotypes represents a gene that determines the final appearance of the design solution, also known as the phenotype. The genotype representation outlines the search space of a problem, while the phenotype representation outlines the solution space (Bentley, 1999). It is crucial to carefully design both spaces to ensure that finding good solutions is manageable. First, a group of designs is formed by assigning various values to the genes. This group is known as the initial population, and it usually comprises diverse solutions to the problem created by randomly assigning gene values to each genotype.

The evolution process commences with an evaluation of each potential solution using fitness criteria. This criterion determines which candidates should be chosen for reproduction in the next generation and which ones should be eliminated, similar to natural selection (Kour et al., 2015). The search space is explored during the evaluation stage to find potential solutions. This process creates a fitness landscape, which shows all possible solutions and their fitness levels (Janssen et al., 2002, p.120). Usually, built-in fitness functions are used to analyze and rank solutions automatically, but sometimes human evaluators are needed, especially when aesthetic choices are involved (Önduygu, 2010). Successful genetic algorithms tend to have increasing fitness scores over time. After assigning fitness scores to the candidates, the next generation is created through an operation called crossover, which combines genetic material from two parent solutions to produce offspring with a mixture of traits (Lewis, 2008). Mutations, random changes to the offspring's genetic material, may also occur to grant some level of genetic diversity. A genetic algorithm's task is to evolve generations of solutions and converge them into satisfactory solutions to the problem.

II. Advantages to the field.

In the segment, we share some ideas on the advantages that evolutionary systems may bring to the design field. We begin with a poetic thought from Bentley (1999) that praises the success of designs created through natural evolution, stating that "The most successful and remarkable designs known to mankind were created by natural evolution" (p.5). In fact, the human brain, arguably the most complex and remarkable design ever created, was generated through natural evolution. Janssen et al. (2002) argue that evolutionary systems are incredibly well-suited to design problems. By considering populations of proposals instead of just one proposal at a time, the designer can quickly choose the most suitable solutions for their intention. Samara (2007) adds that the design process is divided into three stages: conceptualization, execution, and iterative redesign based on evaluation. The author emphasizes that this iterative nature of graphic design aligns well with classic local search optimization algorithms, as they share similar characteristics.

We conclude that evolutionary design methods fit with current generative models utilized by designers. Since programming languages allow designers to generate diverse solutions from a set of instructions, refining those solutions through progressive optimization seems a logical path towards enhancing design practices. The next section reviews some projects that already integrate evolutionary systems into generative design tools.

3.4 Relevant work

In 2009, Quiroz et al. developed a human-guided evolution of brochure documents. The user interacts with a genetic algorithm, which evolves placeholders. Each placeholder is represented with one of three shapes, and the user can then guide the evolutionary process by evaluating a small subset of documents from a larger population. In addition to the user's subjective input, the genetic algorithm also evaluates individuals in the population based on a set of objective heuristics for document design. Figure 3.4.1. depicts two different moments in their evolution process. The "Best" button in the top-left corner of each candidate allows for user subjective evaluation. Önduygu (2010) created Gráphagos, an evolutionary graphic design system using genetic algorithms to randomly mutate and replicate human-evaluated design layouts (Figure 3.4.2). The program was written in Processing. Önduygu intended to create a design tool based on an evolutionary approach to the creative process. An interesting aspect of Gráphagos is how it incorporates design elements. Some are created from scratch, while others are sourced from image and font pools. This concept aims to translate human creative appropriation practices into a digital realm.

The Letterspecies project by Pereira et al. (2019) is a web-based tool combining type design with generative processes to generate unique letterforms or glyphs automatically. The tool uses an algorithmic drawing technique to fill a pre-extracted typographical skeleton, resulting in a visually distinct style for each glyph. Users can adjust specific parameters for each drawing technique, allowing for a wide range of glyphs that suit their visual preferences. A strong feature of this project is that it allows for exporting generated glyphs as a type font. Figure 3.4.3. presents Letterspecies' main interface and system exploration on glyph "a".

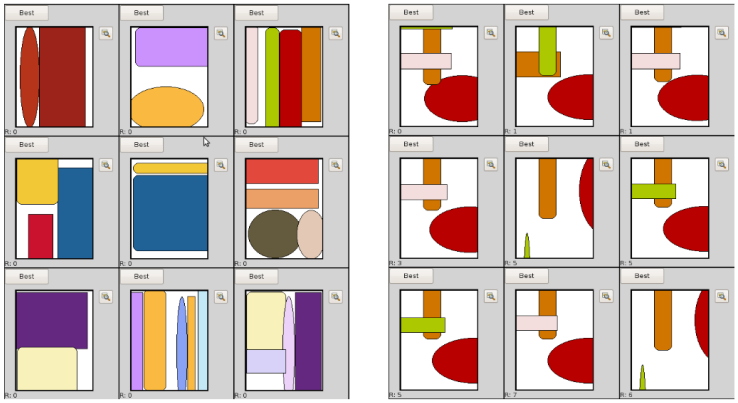
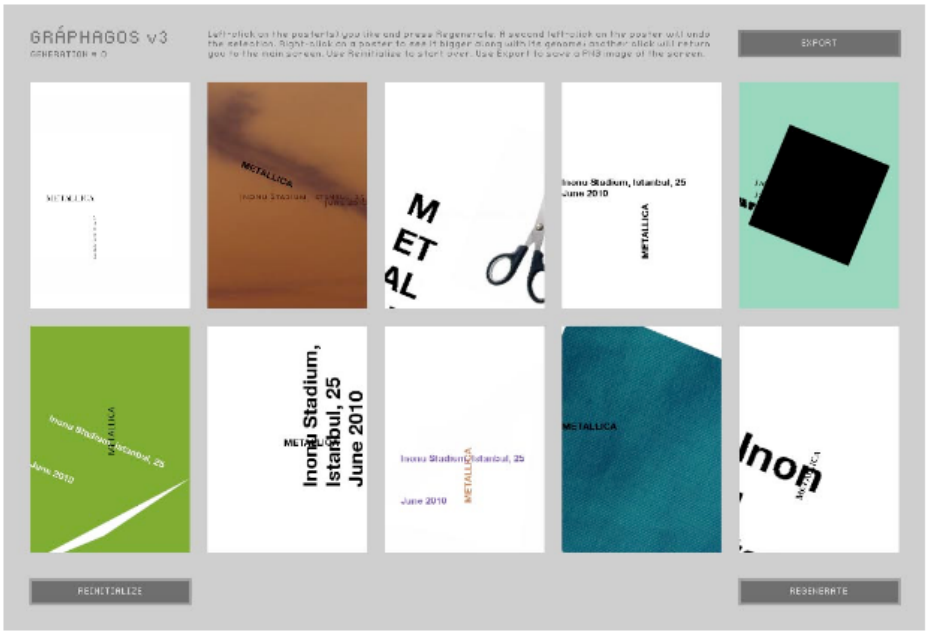
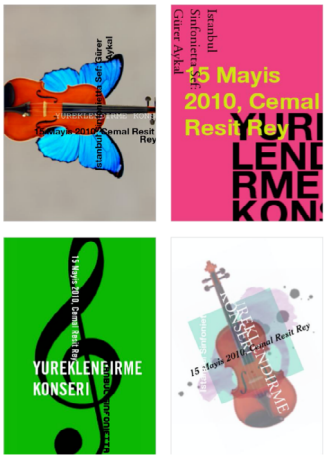


Figure 3.4.1.

Two instances (gen. 0 and 10) of brochure templates were displayed to the user for evaluation by Quiroz et al. (2009).



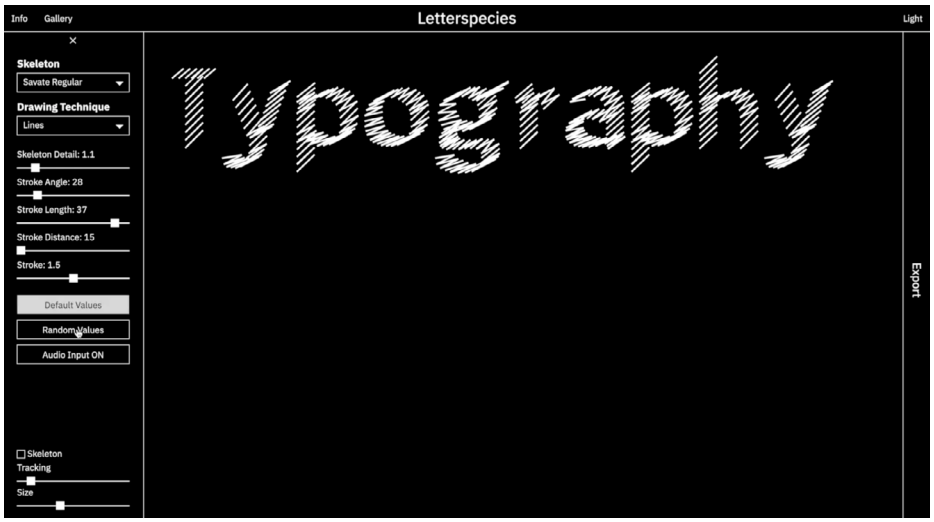
(a)



(b)

Figure 3.4.2.

(a) Gráphagos interface with a random initial population. (b) Results for the poster of Yüreklendirme Konseri.



(a)

(b)

Figure 3.4.3.

(a) Letterspecies project interface. (b) Glyph “a” is generated using different drawing techniques of the system on the same typographical skeleton.

(a)

Rebelo et al. (2020) The Evo-Poster Composer (evo-poster) is a collection of generative experiments that use Evolutionary Computation (EC) methods to create, improve, and assess typographical posters (Figure 3.4.4.). These experiments aim to establish a generative process that mimics the traditional letterpress print houses workflow in the 19th century. An essential aspect of this work involves exploring diverse evaluation methods. The authors integrate multiple fitness assignment schemes into the system to enhance the quality of evolution. Figure 3.4.4, image (b), depicts a valuable instance of Rebelo's exploration. Users can submit text inputs to generate multiple typographical posters.

Still, in 2020, Lopes et al. proposed an evolutionary engine for generating glyphs, aiding designers to explore during the creative process. Their system employs a Genetic Algorithm to evolve SVG paths using interactive and automatic fitness assignments (Figure 3.4.5.). The system was made to serve as a tool to generate ideas for designers to create novel glyphs. To authors, evolutionary approaches have great potential to promote novelty as they are similar to human design processes.

(b)

Figure 3.4.4.

(a) Example output of the Evolutionary Poster Composer Approach using the content gathered from a poster designed for the May 1968 protests (France) and Coimbra's Academic crisis of 1969 (Portugal).
(b) Some executions of the Composer succeeded by the user's textual input.

(a)

(b)

Figure 3.4.5.

(a) Snapshot of Adea's interface.
(b) Artefacts are designed using novel glyphs generated by the system.

3.5 Considerations

This literature review has allowed us to reach some conclusions. (a) The discipline of design consists of formulating a visual language with the intention to communicate ideas. (b) The efficient communication of these ideas depends on a set of decisions made during the artefact execution. (c) The content of this communication constitutes a design problem requiring a solution. (d) Current programming practices based on parametric manipulation have expanded the spectrum of perceivable solutions to address a design problem. (e) In addition, evolutionary approaches allow groups of solutions to be optimised, assisting the designer in identifying the best ones. (f) Some designers and researchers have sought to implement tools encompassing most of these elements in a single platform. (e) However, the analysed works fitting this description foresee a particular type of visual output beforehand. In these projects, the search space is predefined by the creator, who channels the results to a specific group of outputs.

Upon these assertions, we consider it necessary to explore the implementation of a tool capable of integrating all the elements inherent to the design practice, which may accelerate important aspects of the creative process. With this tool, the designer could create and explore his own search space to find solutions to his particular design problem.

CHAPTER 4.

FRAMEWORK PROPOSAL

This chapter proposes a framework for structuring system development. It is divided into four sections: (a) description and objectives; (b) system overview, architecture and pipeline, which involves a comprehensive description of the required modules; (c) user flow diagram, given our interest in ensuring the system's relevance as a tool, and (d) development plan, as a Processing application.

4.1 Description and Objectives

This framework represents a generic model designed to generate, evolve and export multiple instances of a provided sketch code (featuring nearly [4] unpredicted characteristics). Here are our objectives: to build a parameter-based search space from an external input, manipulate parameters within this space to generate variations, employ a genetic algorithm to evolve those variations, pursue optimal solutions and export the outcomes as independent, executable code sketches. This model is designed to assist users in exploring design solutions and should work as a tool for designers, relocating them "at the centre of the design process" (Janssen et al., 2002). The following sections explain how we intend to accomplish these objectives.

4. See segment "Reflections on the Input" from Chapter 6.

4.2 System Overview

Our framework, just like Rebelo et al. (2020, p.111) and other similar frameworks, operates with inputs and outputs. The user feeds the system with a sketch code as input and generates successive variations of that sketch code. Each variation showcases a unique visual outcome by assigning diverse values to the parameters used to create the design artefact (outputs). This process comprises five components:

Parameters Identification;
Generation Module;
Evaluation Module;
Evolution Module;
Interface.

After the user uploads the code he wishes to variate, the (a) Parameters Identification module initiates. This module is responsible for extracting information on parameters present in the code while parsing them into a table that includes the type of parameter, its starting value, and the boundaries for exploration (min. and max. values). Once the user approves the table, the (b) Generation Module creates a population of sketches corresponding to slight variations on the input by stochastically modifying the parameter's default values and injecting them back into the source code. While in this module, those variations are exported to a folder and rendered on-screen in separate windows. Rendering allows the user to evaluate each sketch window, closing the less promising ones and leaving the ones worth exploring further intact. (c) The Evaluation Module maps that interaction to a fitness score, as a client-server utility permits the system to determine which windows were closed and which were left open. After fitness assignment, the (d) Evolution Module breeds values for a new population of variated sketches, this time stemming from the previous population based on evolutionary thinking, where each variable parameter corresponds to a gene and the fittest individuals (sketches) have more chance to reproduce their genotype than the less fit. Those values constitute an offspring generation compiled again inside the Generation Module, yet this time, values were not assigned randomly as they result from that user-guided genetic algorithm. At this stage, the system enters a loop of evolving, exporting and rendering populations of sketches as long as the user decides. Figure 4.2.1. illustrates this system architecture. We will now describe each component in detail.

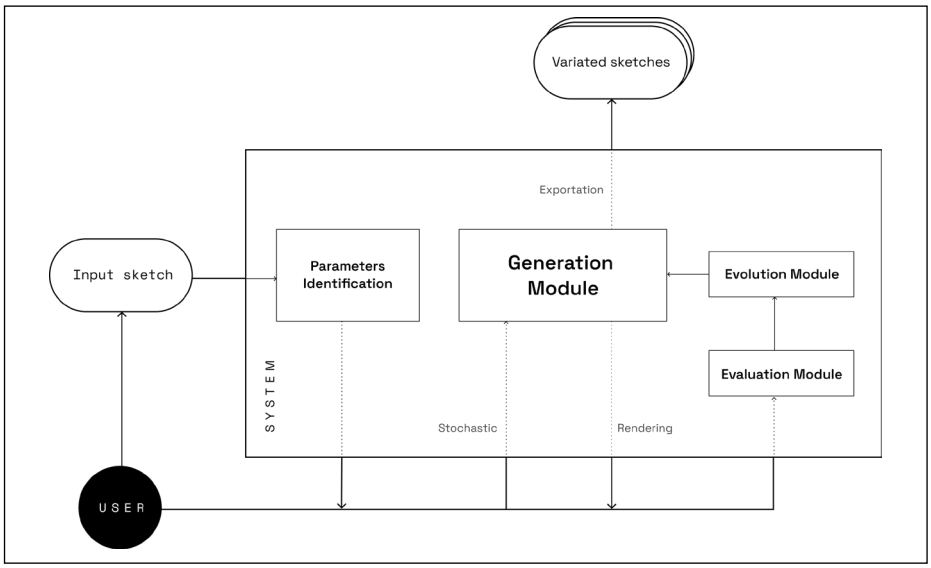


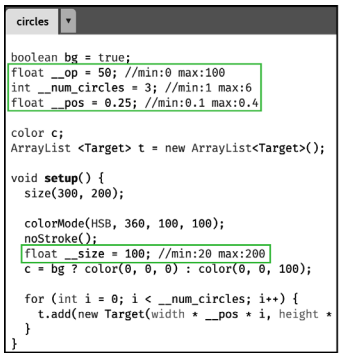
Figure 4.2.1.
Framework's architeture diagram.

I. Parameters Identification
Input (User uploaded sketch) → Output (Table with extracted parameters)

This module triggers a cascade of tasks that starts with the user's input and ends with a table of parsed parameters constituting the parametric space we intend to work with. It reads the input's source code as a regular text file and searches each line for parameters. The system must know what it is looking for. We achieve that by defining a markup language (to tag parameters) that needs to be applied before uploading. We require two underscores preceding the parameter's name (tag 1.) and the definition of what Krish (2011, p.93) presents as the exploration envelope (tag 2.), the minimum and maximum values that may be assigned to each parameter (two slashes followed by "min:" and "max:"). Below is an example of a correctly marked parameter set to be variated:

Original parameter → *loat radius* = 10;
Tagged parameter → *float __radius* = 10; *//min: 5 max: 30*

The system starts searching for tag 1. When tag 1. is detected, it inspects the respective line of code to confirm its status as a valid variable parameter (while removing outliers). A parameter is considered valid only when it is initialized, which means the first time it occurs in the code (further value assignments depend on the user's code design and may relate to other parameters and operations). We verify this by checking if the data type (e.g., *float* or *int*) precedes the declaration (the data type will also be useful for further value manipulations). After validation, the system searches for the next tag (tag 2., boundaries). If tag 2. is absent, the parameter remains technically valid but is not processed as it is unusable. Parameters get parsed to a table only if tags 1. and 2. exist. This table collects valuable information on that parameter data type, name, initial value, and envelope. The task proceeds until all viable parameters are recognised and parsed, as we need to define the parametric space before advancing to generation. Finally, the user validates the list of parameters and triggers the next module. Figure 4.2.2. illustrates the extraction of parameters from an uploaded source code and their representation in a table.



```
circles

boolean bg = true;
float __op = 50; //min:0 max:100
int __num_circles = 3; //min:1 max:6
float __pos = 0.25; //min:0.1 max:0.4

color c;
ArrayList <Target> t = new ArrayList<Target>();

void setup() {
  size(300, 200);

  colorMode(HSB, 360, 100, 100);
  noStroke();
  float __size = 100; //min:20 max:200
  c = bg ? color(0, 0, 0) : color(0, 0, 100);

  for (int i = 0; i < __num_circles; i++) {
    t.add(new Target(width * __pos * i, height *
  }
}
```

Parameter	Type	Name	Value	Limits
0	float	op	50	min: 0 max: 100
1	int	num_circles	3	min: 1 max: 6
2	float	pos	0.25	min: 0.1 max: 0.4
3	int	size	100	min: 20 max: 200

Figure 4.2.2.

Parsing identified parameters from source code. On the left a code example. On the right, a table gathers the identified parameters.

II. Generation Module

Input (Array of genotypes) → Output (Phenotypes exportation and rendering)

This module generates variations of sketches from the original input. As we've pointed out, the parametric space is built from a table of parameters. Each parameter has an associated value that may be modified within its envelope. Changing these values produces a different parameter combination, resulting in a different visual outcome. We consider this combination of values a **genotype** (or chromosome), with each value corresponding to a gene. The Generation Module gathers multiple genotypes (coded solutions), identifies a search space, and creates a population of variations from them (actual solutions), building a solution space. It processes genotypes in two distinct moments: (a) creating the initial population and (b) generating new populations.

A. Initial Population. In this stage, our system randomly assigns a new value to each parameter within the minimum and maximum limits, creating a genotype. The user chooses the number of genotypes he wants to generate. That number defines the population size of sketches during execution. This stage runs once, and it's used to create an initial, stochastic population of sketches.

B.Evolved Populations. Stage II is very similar, but the genotypes are not randomly generated. Instead, they result from the evaluation and evolution modules application (4.2.3 and 4.2.4, respectively). This stage is used to breed an offspring population from the previous one and runs continuously until the user decides to end it.

Once the genotypes are read, the system iteratively replaces the initial input values by injecting each genotype's genes back into the original code (injection 1). This results in variations that are identical to the source, except in parametrization. These variations are phenotypes - sketches that generate a family of different visuals from the same code structure. The process ends by exporting all phenotypes as runnable files to a folder and rendering them on screen through the system in separate, grid-displayed windows. However, we apply another code injection (injection 2) before that. As phenotypes are displayed in separate windows from independent computer processes, we use sockets to facilitate communication between sketches and the system, which will allow us to evolve them. Each rendered sketch functions as a client, sending information to a server allocated by the system (detailed in section 4.2.3). So, extra code must be added to sketches to enable this communication (each injected line is tagged as such and may be deleted later when the execution stops).

III. Evaluation Module, Sockets approach
Input (User interaction) → Output (Ordered list of fitness scores)

A genetic algorithm requires evaluating candidate solutions, which affects the evolution process. Visual cues about potential sketch code appearances are provided through phenotypes so the user can guide the search process toward better solutions. That assessment is made through a fitness assignment processed in this module.

When the Generation module renders a population on screen, each window represents an individual, corresponding to a possible design solution. To classify these candidate solutions, we apply a user-guided fitness assignment. In this approach, the user's interaction frequently results in positive increments on the phenotype fitness score as he selects the solutions he likes the most. This method is standard in projects where the entire execution occurs within a single system interface (Önduygu, 2010). However, we have a distinct design, as our system creates and executes sketches that run through independent processes. In modern operating systems, when a computer executes a process associated with a visual object, that object is typically displayed within a graphical user interface window, which acts as a container for that process's visual representation, allowing users to interact with it. This interface based on

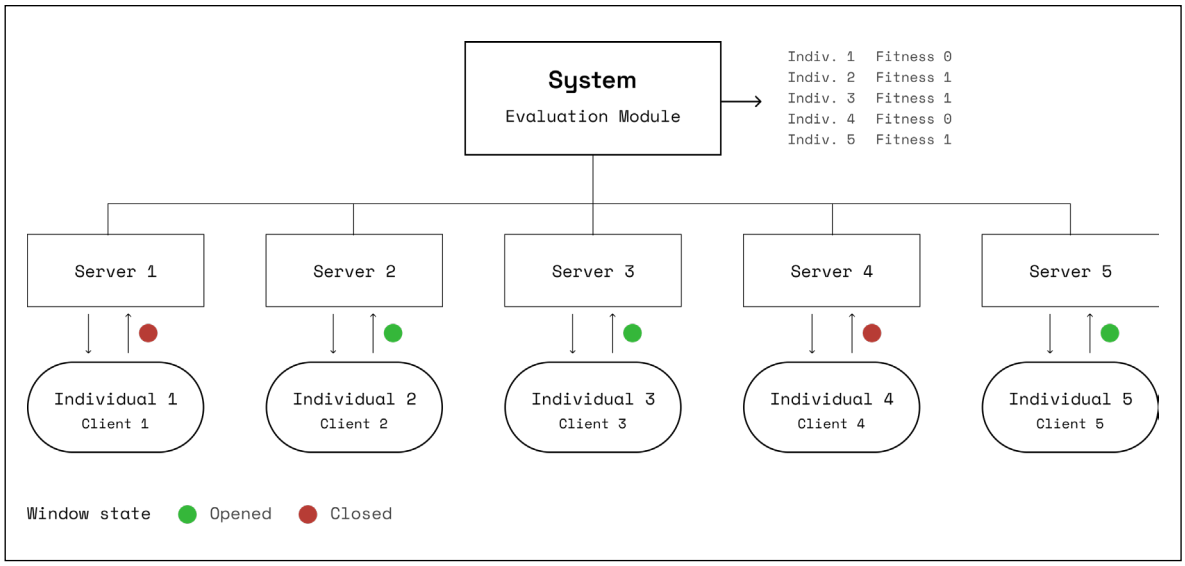
windows presents us with a different way to evaluate sketches. Instead of working on complex methods to select the ones we like, we close the ones we do not. All phenotypes start with good scores, and if the user chooses to cease their execution (closing the window), the score turns null. This approach constitutes a binary, inverted fitness assignment based on closed-opened windows.

The Evaluation module maps the described interaction to a numerical fitness score of 0 or 1, where 1 represents a good candidate and 0 a bad one (we found this binary classification the most effective method[5]). The mapping is done with a TCP-IP-based socket communication[6], a client-server architecture to enable exchanges between sketch windows and the system. Each phenotype operates as a client program that creates a socket and requests a connection to a server. The system assigns a port and server to that client, creating a socket object at the other end of the communication, enabling the server (system) and client (sketch) to read and write data. This module creates a table listing the currently rendered individuals and their fitness scores. Each sketch window updates the system with its current score, depending on its state (open or closed). When the user decides to breed the next generation, the last read values are ordered from the highest to the lowest and sent to the Evolution module. It is worth noting that this architecture allows for design features beyond fitness assignment and will be mentioned again. Figure 4.2.3. synthesizes the described component.

5. See considerations on the fitness assigment from Chapter 6.

6. Transmission Control Protocol/Internet Protocol (Shacklett et.al, n.d).

Figure 4.2.3.
Client-server architecture for fitness assignment.



IV. Evolution Module

Input (Parent population + fitness score) → Output (Genotypes for a new population)

The Evolution module executes a standard genetic algorithm used to breed a new population of variated sketches from the previous population. This new group should reflect improvements based on user feedback. It branches into two stages: selection and reproduction.

A. Selection. In order to attend user evaluation, it is necessary to select which genotypes from the ancestor pop. will reproduce the next one. The previous module outputs a list of all individuals ordered by fitness. With that list, we conduct a tournament selection by randomly selecting a subset of *x* individuals (the tournament size) and ranking them based on their fitness. The most fit individual is chosen as a parent for reproduction. This process is repeated *n* times for the entire population (Jebari & Madiafi, 2013, p.338).

B. Reproduction (Genetic Operators). After parent selection, we employ a sequence of operators to reproduce the next generation: elite, crossover and mutation. Elite children are direct copies of the fittest genotypes within the ancestors. The number of individuals bred from this process is determined by an elite size parameter. For instance, if the elite size is set to one, only the most fit individual will survive to the next generation. The remaining offspring is reproduced through a combination of crossover and mutation operations.

The crossover operator combines the genetic information of two parents. There are several ways to conduct this. We use a single-point approach. A random crossover point is selected in the genotype string, and the genetic information of two parents beyond that point is swapped with each other (Katoch et al., 2011, p.8098).

The mutation operator maintains diversity (ensures some divergence), simulating what is known in biology as errors in the copying process (Janssen et al., 2002). It employs random changes to specific sections of the individual's encoding scheme. Our encoding scheme is value-based, as each genotype is defined by a string of values/parameters (integers, floats, booleans, and others). To apply the mutations correctly, we must identify the data type associated

with a specific gene and mutate it within its scope. To avoid hard-coded mutations with rough alterations on the gene value (that may lead to drastic changes in the final artefact), we apply a Gaussian distribution to the mutation ranged between the gene's minimum and maximum permissible values with a scale factor on the standard deviation. These operators work on a probability base, meaning the user may choose their recurrence in reproduction. When the entire offspring is reproduced, it is sent to the Generation Module.

V. Interface

Our desktop application interface has two components: (a) a control panel for guiding the generation process, calibrating genetic operators, and manipulating the parametric space, and (b) a group of windows arranged in a grid layout that displays the current population. Figure 4.2.4. depicts this structure. On the left is a visual representation of a population. On the right is the control panel with parsed parameters and system settings.

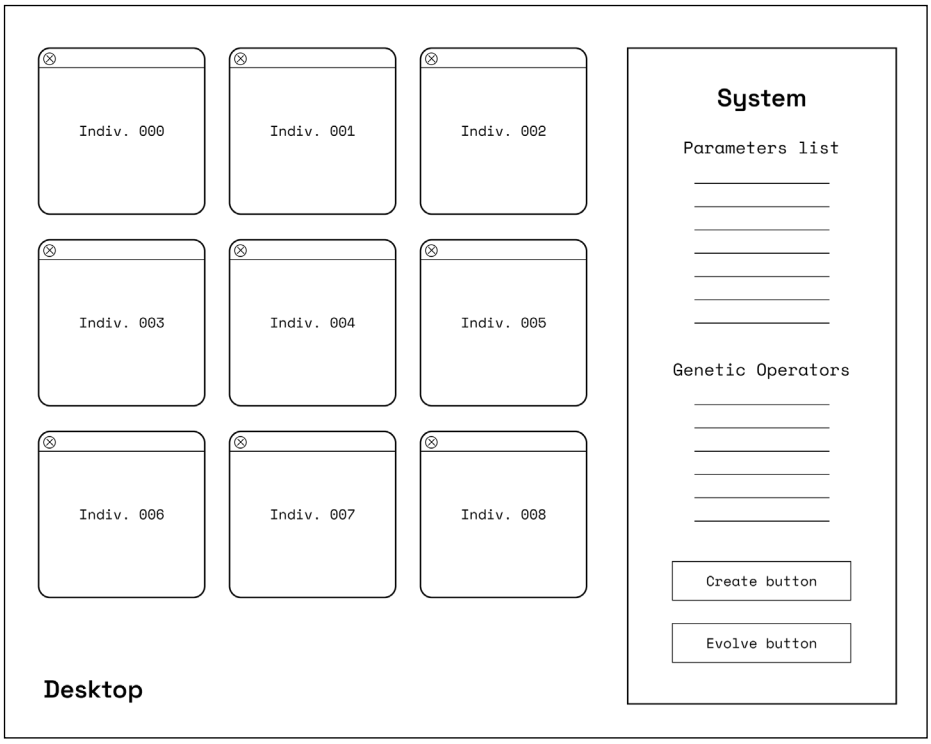


Figure 4.2.4.
System's interface wireframe.

4.3 User flow diagram

Figure 4.3.1. illustrates user navigation from sketch upload to evolution. Rounded rectangles mark the start/end of the user flow. Straight rectangles represent steps to take, and diamond shapes indicate decision points. Underlined tasks constitute critical steps. At the bottom is a parallel view locating the framework's modules in time during execution. This diagram will be helpful for interface design and evaluation.

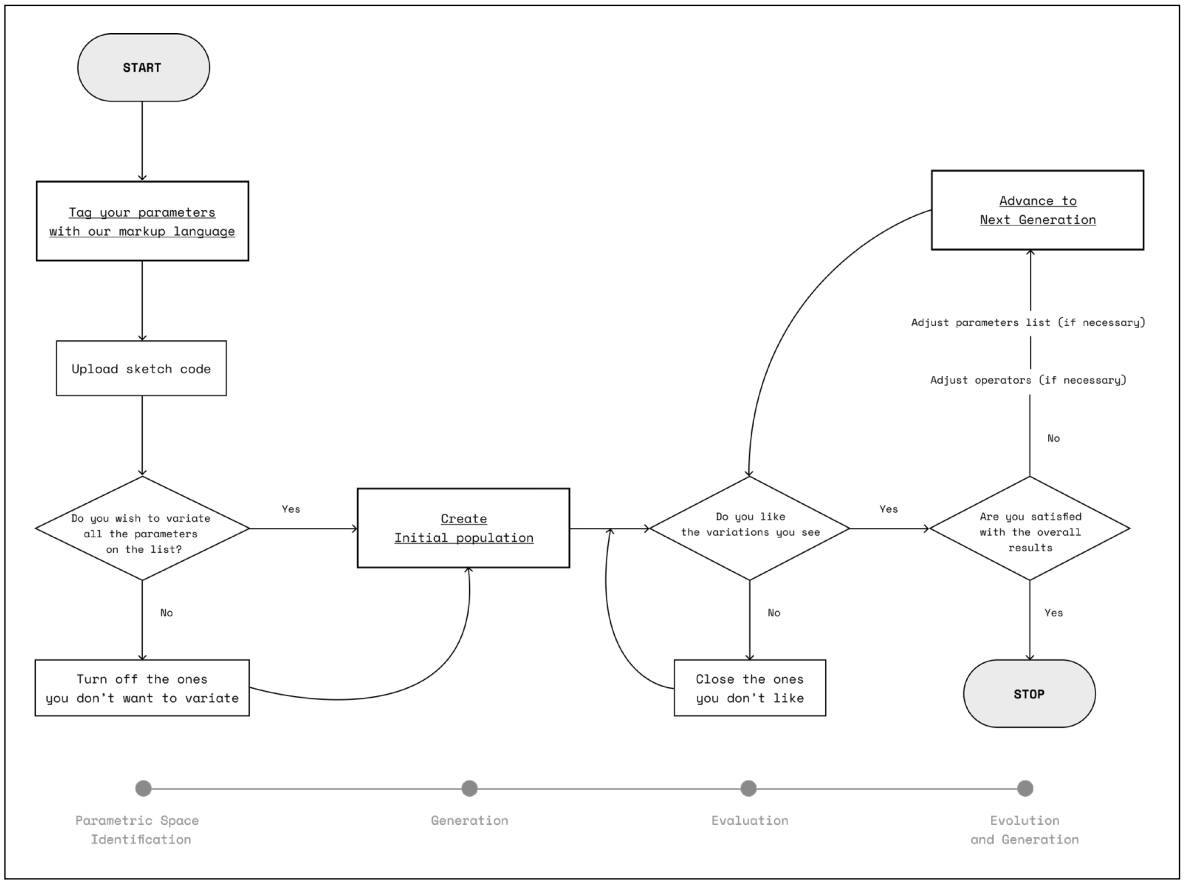


Figure 4.3.1.

User flow diagram depicting the central steps of the execution process.

4.4 Development structure

We will use Processing, a widely-used software in the design community (as mentioned in the Literature Review chapter), to develop our system. By doing so, the framework ensures compatibility with the standard programming language used by our target audience. This gives us the possibility to test and evaluate the system in real scenarios.

The development process has two stages: creating a proof of concept prototype and building the actual application. The idea behind creating an early prototype is to prove the framework's potential and the viability of critical aspects, such as the markup language assignment, the Parameter identification, and the generation of multiple instances of the given sketch code. This early endeavour also helps to initiate some components regarding user interaction, counting control panel and communication between windows and the system. Once the concept is proven, system development starts where the prototype left off. This will involve integrating the evolutionary component, as well as refining the interface and adding secondary features. During these stages, we expect to produce distinct results to gain empirical insights and strengthen our proposal.

CHAPTER 5.

PROOF OF CONCEPT

This chapter gives an overview of the first stage of system development. The objectives include assigning the markup language, uploading sketch codes, extracting and parsing parameters, generating variations, rendering, exportation, and interface early design. Here, we cover technical details, early results, and considerations since the Framework Proposal already describes the conceptual aspects.

5.1 Environment

The prototype was developed as a Processing application (Fry & Reas, n.d). In addition to Processing's integrated programming language, we added some segments in native Java for specific purposes, like parts of the socket communication and sketches rendering.

5.2 Notes from input to parameters extraction

I. Parsing.

The first technical challenge was related to parameter identification in the given sketch code. The chosen approach was to work with String indexes. To explain this method, let us return to the example of a properly marked parameter: *float __radius = 10; //min: 5 max: 30*. Since we want to extract and distinguish the information in this line, we must provide the system with the necessary tools to do it. We reflected on which characters were always present in a variable declaration (inspired by Alan Turing's pattern comparison (Raikar, n.d).). We know our markup tags will be there, but we also know that in Java, there will be an attribution key (=) and a semicolon (;) to close each declaration. With this knowledge, we retrieve the length of the code line (trimmed to avoid errors) and the characters index in it. Then, we define intervals between those indexes to extract the content inside them into an array. This allows us to filter the declaration and parse the valuable information into a table. The method is precise, so if any identified keys are missing, the parameter is considered an outlier (for our purposes). Reversibly, we use the same index information not to extract but to inject new values in the sketch code and produce a variation.

II. Data types analysis.

During this phase, it was crucial to differentiate between mutable and immutable data types. Any data type associated with numbers (byte, short, int, long, float and double) may be altered, as it is possible to define a range of exploration and assign new values within that range. This includes boolean variables (essentially a choice between two states: 0 or 1) but excludes non-primitive types like Strings, Arrays and Classes. While being primitive data types, char variables may store letters of the alphabet. For that reason, we also excluded them. It is relevant to notice that our system's input and output is text (encoded as Strings), so the variable type is only considered to manipulate the parameter's assigned value. Parameters life cycle always begins and ends as a String.

III. Uploading refinement.

During the first development iterations, we had to manually input the source sketch location path in the directory structure. To solve this, we added an interactive action using Processing's selectInput() method to open a file chooser dialogue for selecting input files. The system then stores the selected file in an array of Strings.

5.3 How to render and export

Exportation and Rendering are crucial for our intentions, so they needed to be solved early. Both steps are related since the sketches exported are the ones we render on screen.

I. Exportation.

Exporting the variations we create as runnable sketches has benefits. We may visually review each variation and provide the designer with sketch files for later use. Exploring design solutions is only relevant if they remain accessible after system execution.

This process involves injecting the list of new values and communication methods (see Generation Module segment from Chapter 4) into the source sketch encoded array of Strings. The array is saved as a Processing file in a " Variations " folder. Each execution implies exporting multiple files that will be rendered, so we used a structured indexation method to assign filenames. For that, we use two counters, one for each generation and one for each sketch within that generation. As we knew we intended to implement a genetic algorithm, we applied its semantics. The tag "pop" (from a population) precedes each generation index, and "indiv" (from an individual) precedes each sketch index (the initial name was "modified", but we found it inaccurate). This is an example of a pseudo path leading to the eighth variated sketch from the first population: *system/variations/pop_000/indiv_008/indiv_008.pde*.

II. Rendering.

After completing the export process, Processing's processing-java extension automatically executes sketches without requiring the user to open a terminal (as would be the usual procedure). This extension sends "-run" commands using the same counters that created the sketch paths (predicting that sketches with a certain filename will be in the "Variations" folder), all within the system's task flow. That's how we render exported populations.

An important aspect of this process is window positioning on-screen, as we want to avoid overlap (by default, Processing centres the sketch). We designed a grid layout of relative positions to present each variation. The grid calculates sketch positions by combining their index (indiv_000, 001, ...), population size and screen dimensions. The positions are then assigned with Processing's setLocation(). This property is part of the extra code injections done during exportation. We also considered randomly assigning positions to create a "hacked" desktop feeling. However, when tested, this approach was ineffective, making it difficult for the user to see and evaluate variations.

5.4 Initiating interface design

To begin considering the interface, we designed a simple, straightforward control panel with buttons that led users through a sequence of ordered tasks (to ensure each method worked correctly). The panel was placed on the right side of the desktop, leaving room for variation windows. We left empty space in it to accommodate future feature additions. It worked fine as a prototype, but we knew it would require elements hierarchy and clarity improvements. Figure 5.4.1. illustrates this first experiment. The system uploads a Processing sketch that draws balls in motion. The control panel allows us to (a) run the original input sketch, (b) generate variations, (c) run variations and (d) print the ones we like (explained in the upcoming segment).

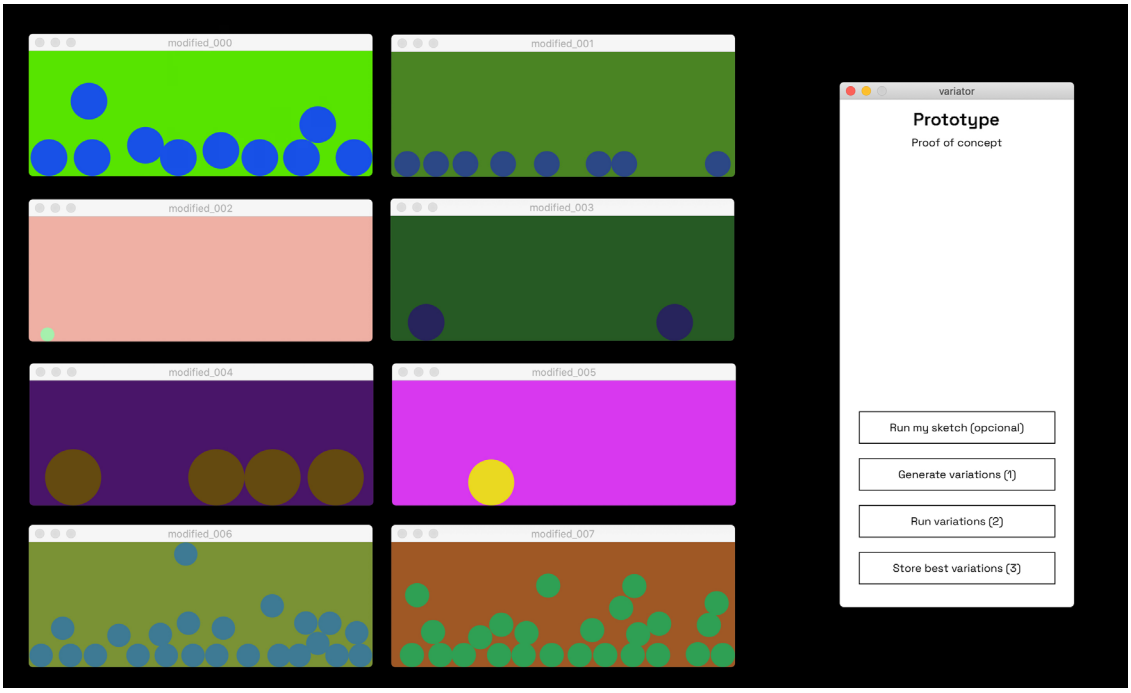


Figure 5.4.1.

Initial interface snapshot. On the left is an eight-variations rendered population of a sketch named “Bouncy Balls”. On the right, the control panel with simple ordered tasks (the user has already pressed the first three buttons at this point).

5.5 First foot on evaluation

We had time to initiate evaluation after meeting the predicted objectives for this stage. Even though the evolutionary implementation had not yet begun, our prototype was already displaying randomly generated variations on-screen, so it was possible to access their quality. We started designing the communication between windows and the system. This functionality fits in the socket-based architecture described in the framework. We used the networking tools provided by Processing, with local ports and client-server assignments for each variation. Once a line of communication is established, each sketch reports a value of 1 to the system as long as it stays open. When the window is closed, it reports a value of 0 before the sketch shuts down. The system prints a list of open sketches using this binary approach, identifying those with a remaining value of 1, which allows for the assessment of preferred sketches. This data will be crucial to breed new generations.

Figures 5.5.1. and 5.5.2. depict this process. A population of twelve sketches is rendered, the user closes the ones he doesn't like and presses the "Store best variations button".

Figure 5.5.1.

Twelve variations of sketch “Rectangles”.

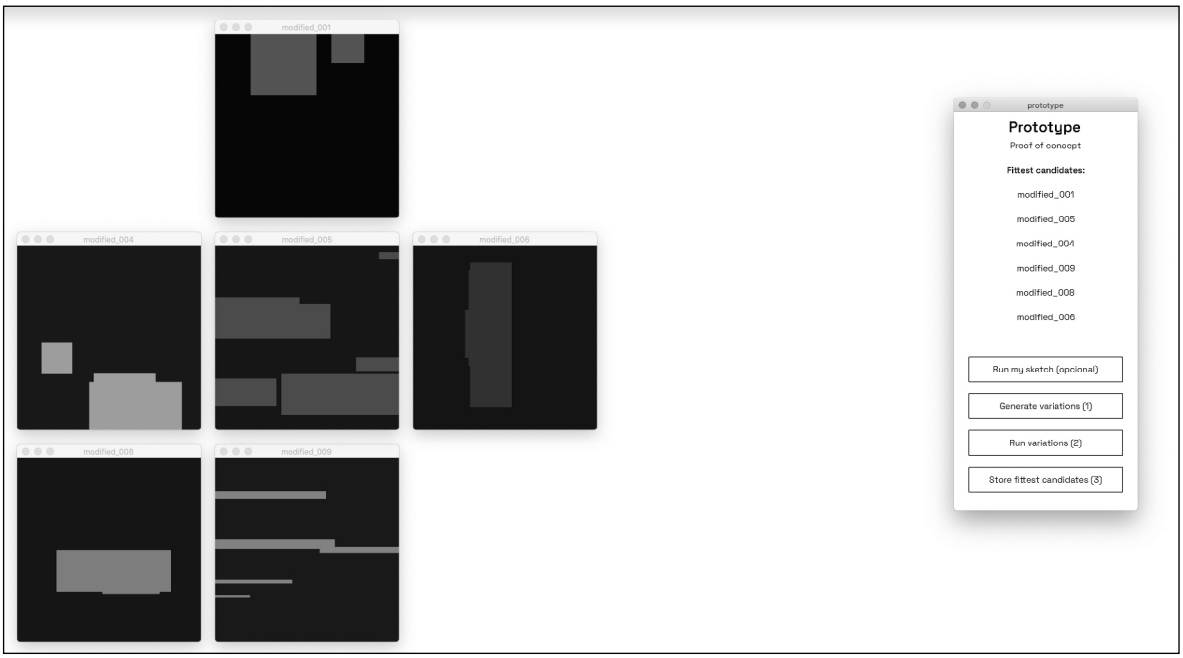


Figure 5.5.2.

The remaining six, after user interaction, are still opened and identified in the control panel.

5.6 Early results

Figures 5.6.1. to 5.6.3. are visual outcomes of experiments conducted with our prototype. All of them represent random variations from sketch codes with distinct characteristics. We already have a sense of a generic generative design tool. Figure 5.6.4. illustrates the problems random positioning may bring, even though it transmits a feeling of organic beauty.

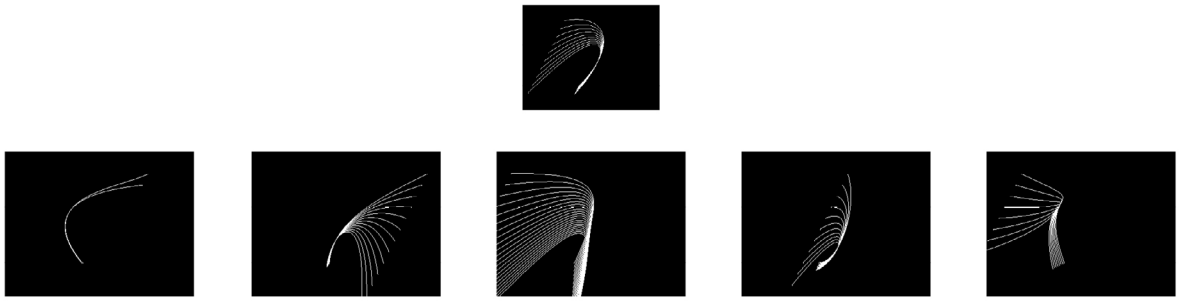


Figure 5.6.1.
Sketch “A Bezier Loop”,
original and selected variations.

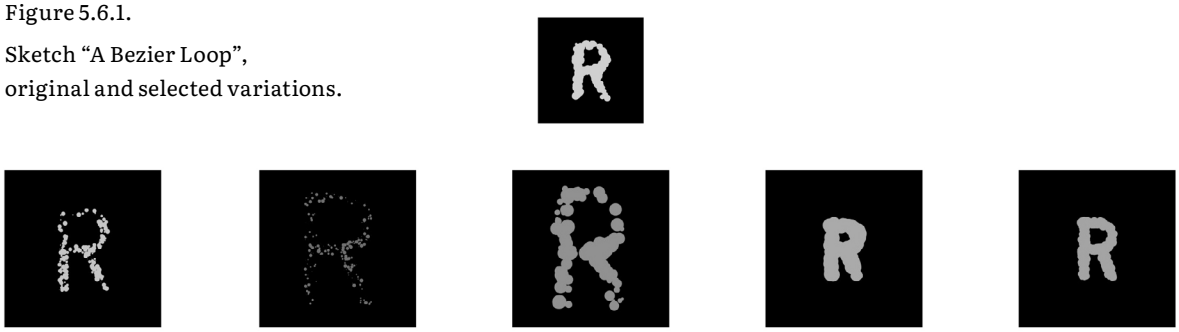


Figure 5.6.2.
Sketch “Type from Particles”,
original and selected variations.

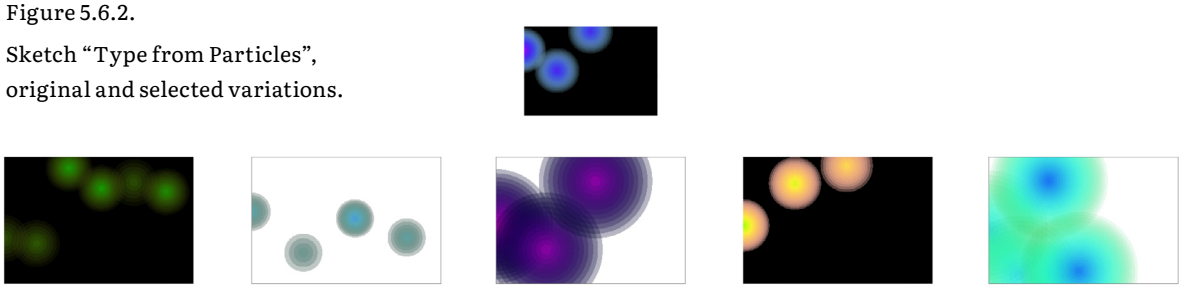


Figure 5.6.3.
Sketch “Gradient Circles”,
original and selected variations.

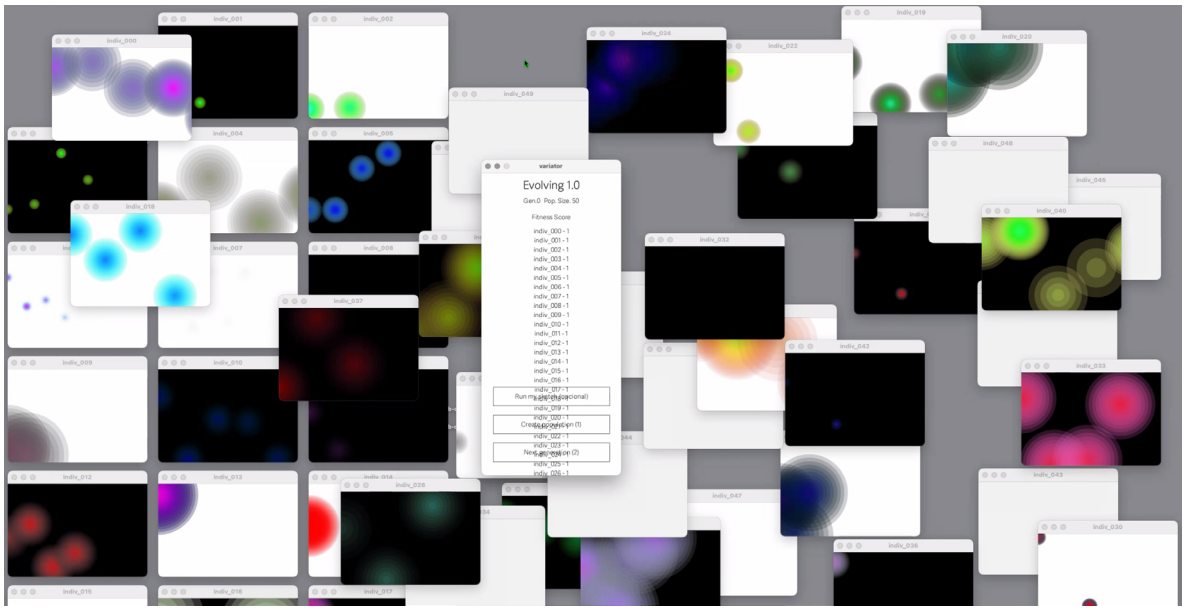


Figure 5.6.4.

Fifty variations of sketch “Gradient Circles”. Generative chaos.

5.7 Considerations

Before advancing to system development and refinement, we considered some aspects. (a) Random generation alone can be interesting. It allows us to visualize different solutions automatically, making this type of exploration helpful at the beginning of the generative process to provide the user with distinct design directions. (b) As we want to render multiple instances of a source sketch, its window dimensions must be considered. It is impossible to reduce width and height or zoom out the entire sketch, as it could affect the artefact. Sketches with larger dimensions may only be explored on high-resolution screens. (c) Rendering can be slow in some cases. The concurrent execution of multiple Java processes consumes considerable computational resources. In Java, this limitation may persist as the system requires a reasonably powerful machine. (d) On a positive note, the system runs on Linux, Windows, and macOS.

CHAPTER 6.

EVOPROTEUS

In this chapter, we delve into EvoProteus, the system we've developed to optimize the exploration of parametric spaces. Below, we discuss the integration of the evolutionary component, technical challenges, new features, interface enhancements, some results and considerations.

Figure 6.1.

Proteus by Jörg Breu the Elder (1475–1537).

EvoProteus is born.

We named our system EvoProteus. In Collins Dictionary (n.d), the term "protean" means "readily taking on various shapes or forms; variable" and comes from the Greek sea-god Proteus (Figure 6.1.), whose name also suggests the "first" (from Greek "prōtos")(The Editors of Encyclopaedia Britannica, n.d), proposing the idea of a primordial being from which many forms arise. This concept aligns with our intentions, so we combined it with "Evo" to create EvoProteus - a system that explores parametric spaces in an evolutionary, protean way.

6.1 The Evolutionary Leap

We successfully implemented the Parameters Identification, Generation, and Evaluation modules in the prototype's development. Here, we integrate the Evolution Module to generate successive populations of user-evaluated sketch variations.

I. General Notes on Evolution.

(a) In order to integrate the genetic algorithm, we added two new classes to our code: "Genotype" and "Population". The "Genotype" class handles reproduction operators such as elite, crossover, and mutation. The "Population" class handles selection, sorts individuals by fitness and conducts the tournament. It also calls the rendering and exportation methods developed earlier. (b) The strings of random values we used to generate a prototype population in the previous chapter constitute, here, the first set of genotypes. Our Genotype is, therefore, a sequence of values assigned to each extracted parameter and optimized through an iterative process. Our Population is a group of sketch variations obtained by replacing the default values with evolved Genotype values. (c) The only time EvoProteus processes code is during the input sketch

upload. This means that it always operates on top of the original source code to export new individuals. Once the evolution process starts, the only input our system receives is a list of binary fitness scores it uses to breed new Genotypes.

II. Beware of Booleans.

Most variable types our system recognizes are mutated within a Gaussian distribution, but boolean variables have only two states, zero or one, challenging controlled mutations. Changing a boolean value involves going from one extreme to another, which can significantly affect the final visual outcome, such as changing a background from white to black.

III. Between Populations Rendering.

The evolutionary process requires killing one generation and breeding another. In EvoProteus, this implies exiting the ancestor's windows and running new ones in a single-panel instruction. To make this possible, we call our sockets component again. However, instead of listening, the system sends information to the clients (sketches). It opens a binary communication for parent sketches to receive instructions on when to end their execution. When a population is rendered, the system continuously prints the value one into each phenotype. When the user signals "Next Generation", this value changes to zero. For the population, zero means extinction, so the "exit()" command is triggered inside each individual. This method automates the execution, avoiding manually closed windows.

6.2 Debugs

This stage required overcoming two significant technical obstacles related to the entire execution that took us some time to analyse and solve.

I. Handling Temporary Files.

One issue was the creation of excessive temporary files during each execution, which would fill the computer disk space in the most severe cases. We thought this would be related to the rendering of multiple sketches via the command line but then traced this problem to the interface's font file uploading. The system was creating precautionary temporary files due to mistrust of font file paths. We excluded these external font loading and called the typefaces directly with Processing's PFont.list(), indicating the indexes of each desired font.

II. Concurrency Concerns.

We had another significant constraint. Sometimes, correctly exported variations failed to run during the rendering process. Although considered valid by the system, these variations could not be evaluated, resulting in zombie individuals poisoning the offspring with genetic information the user was unaware of. The problem was due to Java Race Condition (Lutkevich, n.d), a concurrency bug. Some sketch windows would crash when the system attempted to perform several operations (run sketches) simultaneously. We tried to add a delay between each variation execution, but that did not work. As it was practically impossible to solve this occasional issue, we worked on a debug button. Each sketch rendering is associated with a Java process/thread ID. When a sketch runs adequately, we use the socket communication to print its ID to the system. We then employ a Java method to store all the existing processes in an array and compare them with the identified healthy sketch IDs. Any outlier constitutes a zombie process (stuck process). The debug button outputs a Java command to terminate it and prompts to rerun the sketch.

Translated to user interaction, if one or more sketches from the current population fail to run, the user may press an "Indiv. missing? Press here" button to complete the rendering process and evaluate the results. It's worth noticing that this bug is unpredictable and, most times, unexisting, especially in powerful computers.

6.3 Additional features

We wanted EvoProteus to be both functional and visually appealing. After fixing bugs and completing the framework modules, we added extra features to enhance user experience.

I. Exploration.

Our first significant improvement was adding toggle buttons to each extracted parameter. By default, all parameters are toggled on at the beginning, which means they are equally subjected to the evolution process, getting consecutive new values. During the execution, users do not need to evolve all variables at the same time. They may freeze the exploration of a given parameter and proceed to evolve the remaining set. This allows much more precise guidance of the evolution process (see segment 6.6). Technically, when a parameter is paused, we look at its assigned values in the current generation and keep those values intact in the following ones. Paused parameters may be activated later.

II. Restart generation.

Another feature we had was a restart button. Users can now restart the evolution process with a single button press. This saves time compared to previous versions, where the program had to be shut down and the input uploaded again.

III. Fitness assignment.

We attempted to enhance evaluation by implementing another form of fitness assessment, a complementary method that would produce more distinct scores. We were particularly interested in the screen's edges, as we believed a sketch dragged out of the edges would likely be less attractive. With our socket component, we developed a method that prints the location of each sketch window on-screen. EvoProteus compares that location with the screen's dimensions. Windows entirely inside the screen maintain a score of 1. If part of a window overflows one edge, we map that proportion to a value between 0 and 1, meaning a window half out and half in receives a fitness score of 0.5. We tested this extra assignment but found it less intuitive than our regular classification based on open-closed windows, which remains the primary evaluation method.

IV. Favourites folder.

We added an all-time favourite sketches folder to the system. It is not evident that the last population generated will correspond to the user's preferred individuals. In between, there may be pleasing solutions. Even though all the rendered individuals are stored in the "Variations" folder, locating the best ones within an entire population is challenging. EvoProteus allows users to press a window with an attractive solution to address this difficulty, storing it in a special folder for later appreciation. Sketches saved to this folder are named by date and time, making it easy to identify them. For example, a favourite sketch might have a path like this: *system/favourites/2023_6_27/sketch_16h_2m_58s/sketch_16h_2m_58s.pde*.

6.4 Interface refinements

The quality of user interaction relies a lot on interface design. Our control panel underwent several upgrades, enhancing its visuals, hierarchy and usability. As part of these improvements, we sought to integrate the previously mentioned features. The following segments describe our design process over time (Figures 6.4.1. to 6.4.9). Before EvoProteus, the system was given many different names.

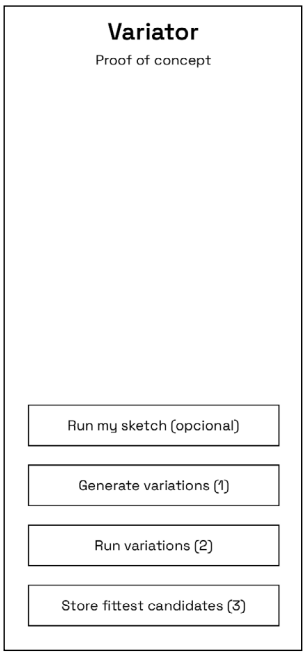


Figure 6.4.1.

Version 1.

The control panel when we started the second phase of development.

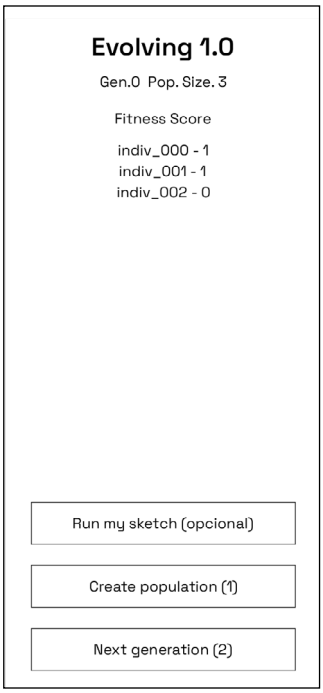
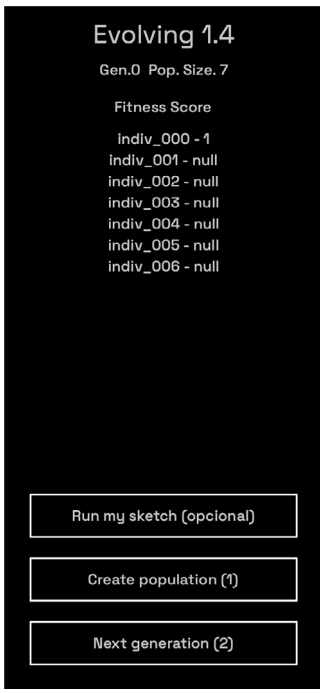


Figure 6.4.2.

Version 2.

Feedback on the number of generations and population size.

The button for viewing the fittest candidates was removed and replaced with a real-time display of scores per individual.



(a)

(b)

Figure 6.4.3.

Version 3.

As we wanted to include genetic settings, we tested the possibility of splitting the control panel into two windows but abandoned the idea due to higher resource consumption and impracticality, favouring a single-panel solution. (a) Main panel, (b) side panel.

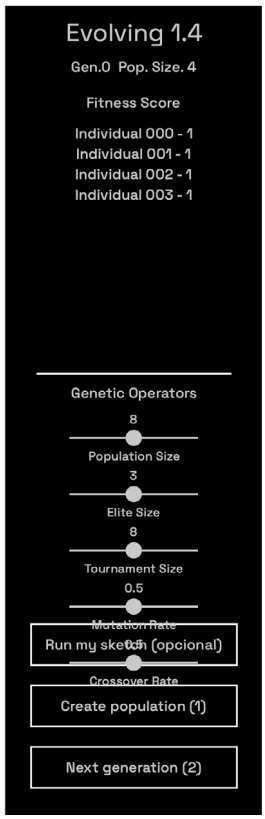


Figure 6.4.4.

Version 4.

This is the first attempt at incorporating the genetic operators into the interface: population size, elite size, tournament size, crossover and mutation rates.

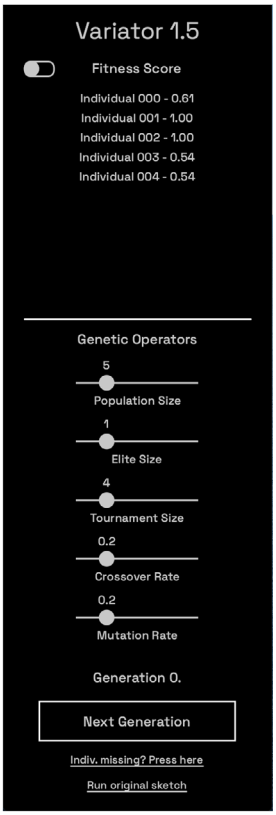


Figure 6.4.5.

Version 5.

We restructured the buttons scheme, creating a main button that exports and renders populations with a single command.
The “run input” button was resized for visual hierarchy reasons.

We added an extra button for concurrency debugging.

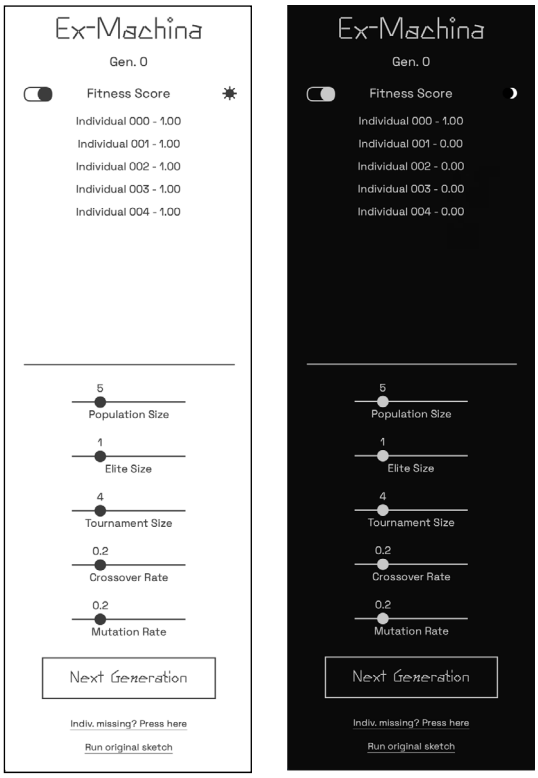


Figure 6.4.6.

Version 6.

Light (a)
and dark (b) mode.

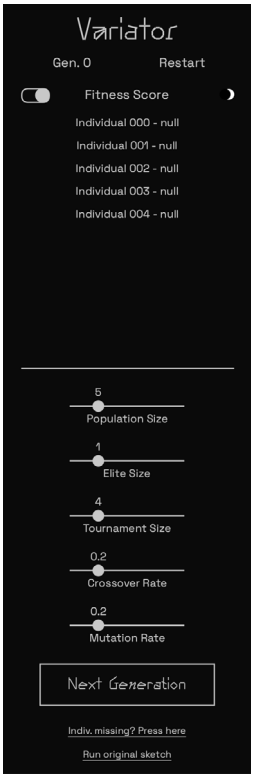


Figure 6.4.7.

Version 7.

Restart button added.

Users may now reinitiate
the execution at any time.

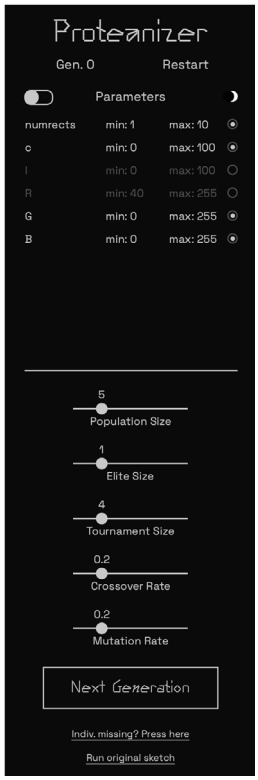


Figure 6.4.8.

Version 8.

The possibility to pause the evolution of specific parameters is now fully integrated with the parameters table component.

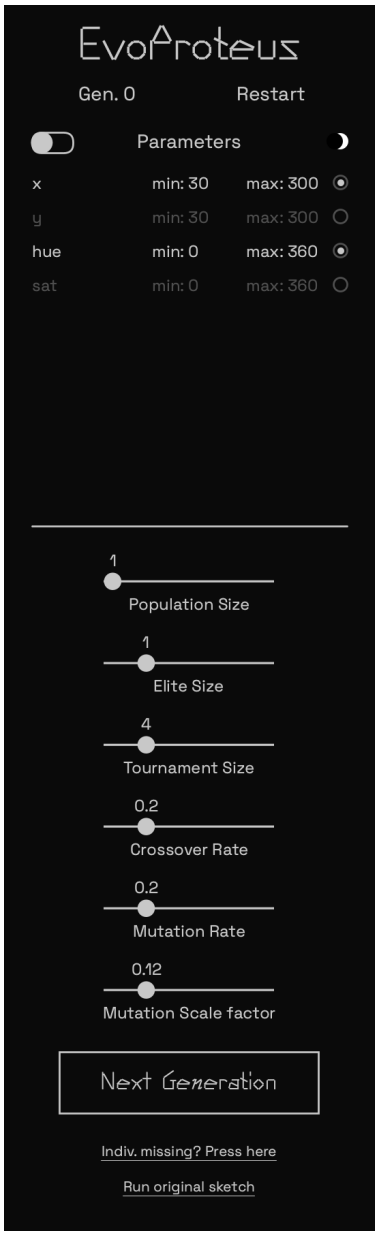


Figure 6.4.9.

Version 9.

Panel's current state.

Typefaces: Anthony by Sun Young Oh; Space Grotesk by Florian Karsten.

We completed the list of operators with a new setting: a mutation scale factor that allows for greater control over the impact mutations have on genes.

6.5 Reflections on the Input

Based on the experiments conducted so far, we share a few thoughts regarding the input sketch to provide insights into the system's most efficient exploration. (a) Simple inputs are preferable. By "simple," we mean sketch codes without too many rules, methods or constructors. Complex sketches are designed for specific goals with clear visual objectives, while simpler code pieces with broader/unclear intentions may profit more from exploration. It does not mean designers who create complex artworks cannot use EvoProteus. In those cases, it might be interesting to evolve separate code segments and search through solutions for each design component.

(b) Less (parameters) is more. Halford et al. (2005, p.75) conclude that "decision-making must entail the processing of no more than four variables in any one cognitive step". We share this vision. Exploring fewer parameters each time (switching to others later) decreases the processing load. Fewer variables allow for a better understanding of how parametrization affects the outcome, which may be complex to perceive with too many parameters at stake.

(c) Dimensions should be taken into account. We mentioned this question previously: The sketch dimensioning must consider the screen resolution. Running multiple instances on small screens requires smaller window sizes for an effective interaction.

CHAPTER 7.

EVALUATION AND RESULTS

This chapter describes different levels of system evaluation. It is divided into four sections: (a) assessing broadness and (b) relevance, (c) initiating user evaluation and (d) experimentation in a different environment. The results obtained during this phase were used to refine the framework's conception and system quality.

7.1 Assessing broadness

Throughout the report, we have argued for the generic nature of our system and how it can extract parameters and produce variations from any Processing sketch. In this segment, we try to prove it. OpenProcessing[7] is an online coding community for sharing artworks made in p5.js[8] (a Processing web adaptation). We came up with the idea of using sketches from this platform to assess EvoProteus's broadness. One advantage of this approach is that it exposes the system to external and unfamiliar inputs.

While exploring OpenProcessing, we came across sketches that caught our attention. To try them out, we had to accomplish two tasks. First, we translated the source code into Processing, which was quite simple since both languages are similar. Then, we assigned the required tags to parameter identification. Figures 7.1.1. and 7.1.2. present two community artworks and their respective executions in EvoProteus. Both tests demonstrate the amplitude of our system and the impact genetic operations may have. In each run, we started with a highly diverse set of potential solutions, and over time, we converged towards similar solutions that met our personal tastes. Figure 7.1.3. depicts another experiment with a selection of the most beautiful outcomes. We tested with lots of sketches, and every time, the system performed efficiently.

7. **OpenProcessing**, openprocessing.org

8. **p5.js**, p5js.org/

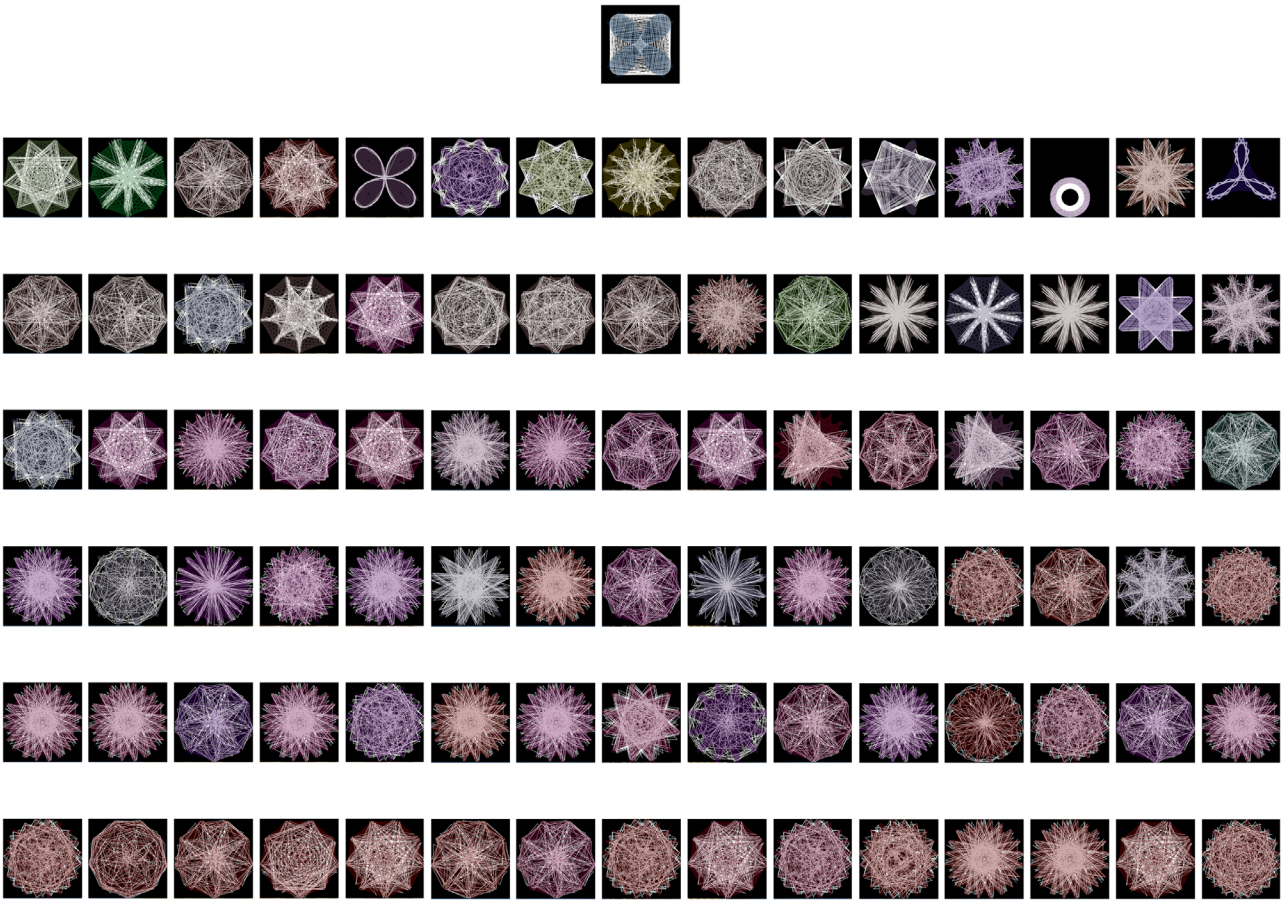


Figure 7.1.1.

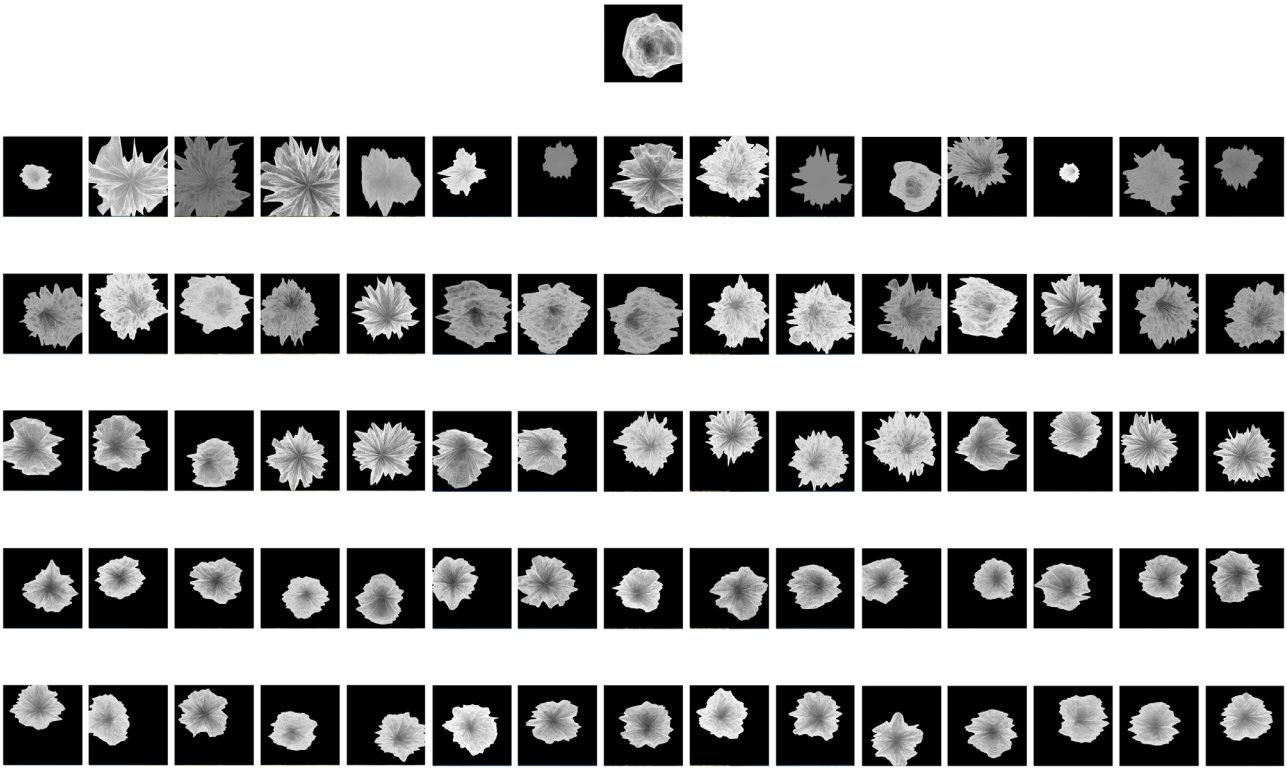
Evolution of “Maurer Roses”, created by Stefan Nicov.
Population size: 15; Number of Generations: 40.

This test confirms the performance of genetic operations,
we start with a random set and soon converge to similar solutions.

Figure 7.1.2.

Evolution of “Arp”, created by Aaron Reuland.
Population size: 15; Number of Generations: 23.

Each variation is an animation.



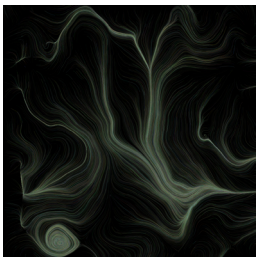
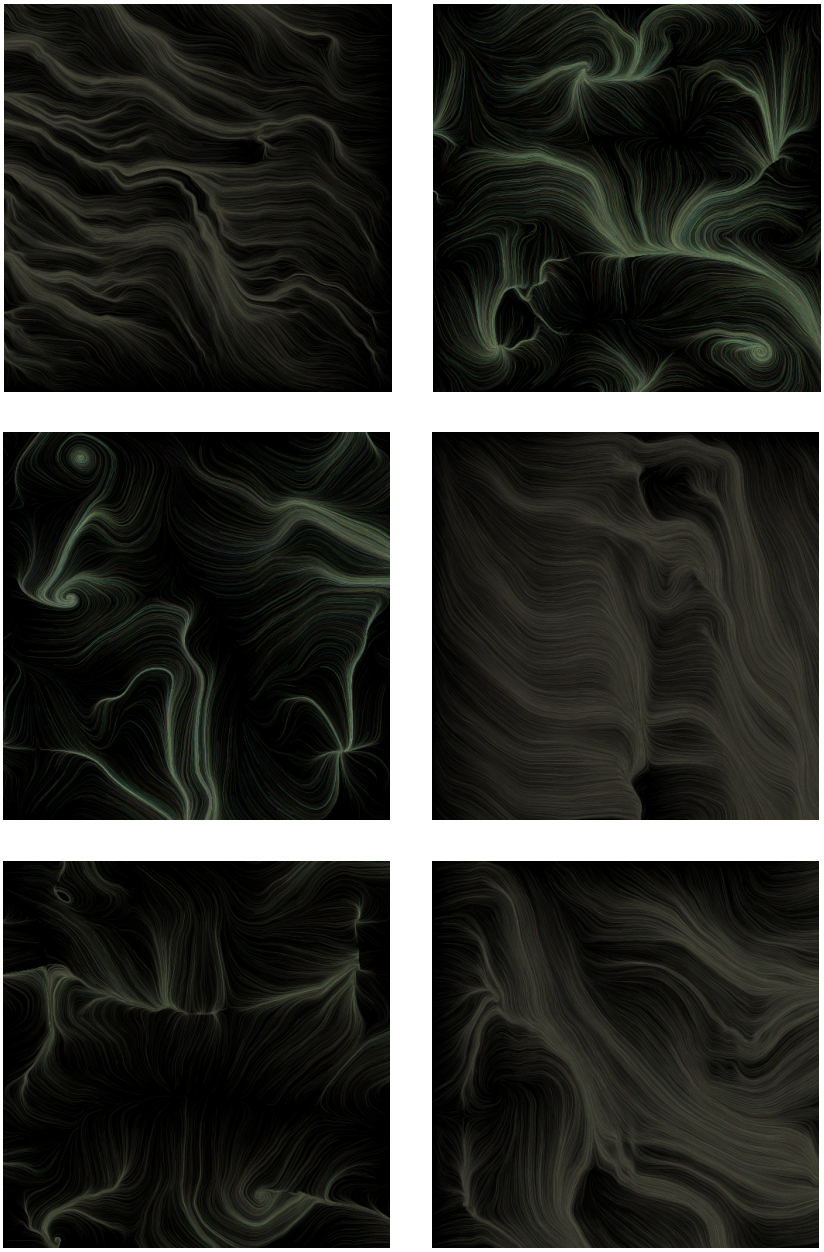


Figure 7.1.3.

Evolution of “Waves”, created by Teng Robin.
Original and selected variations.



7.2 Assessing relevance

Our framework is built explicitly for graphic designers who have programming skills. In this section, we focused on making sure EvoProteus can be used as a tool, assisting in design decisions, from colour choice to typography and composition. To achieve this goal, we created, imported and evolved artworks that fit into the typical design work. In Figure 7.2.1, we experimented with typographic distortion. Figures 7.2.2. and 7.2.3 present some of the most pleasant results from this process. In Figure 7.2.4., we show examples of gradient combinations, which can be helpful when deciding on a visual identity's colour scheme. Figure 7.2.5. demonstrates how the system may provide insights into the composition of an exhibition poster. These examples result from the combination of parametric exploration of core design elements with an evolutionary component that optimizes outcomes based on human evaluation. We are pleased to see that EvoProteus is relevant to the design field.

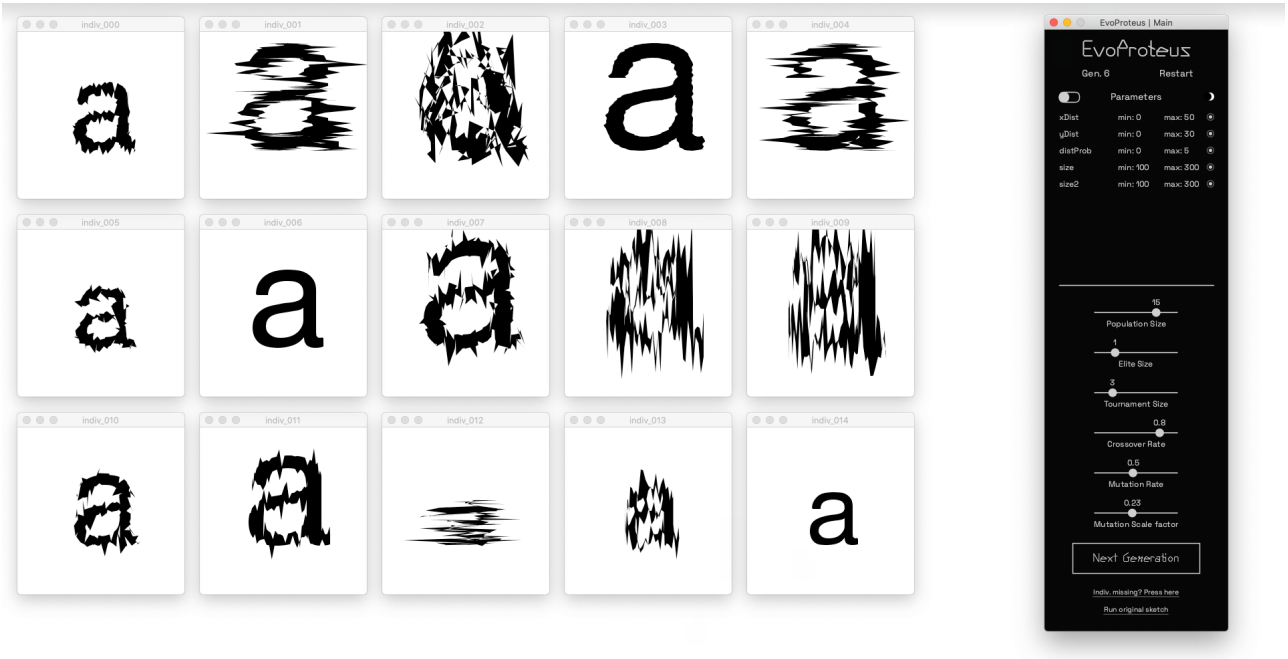


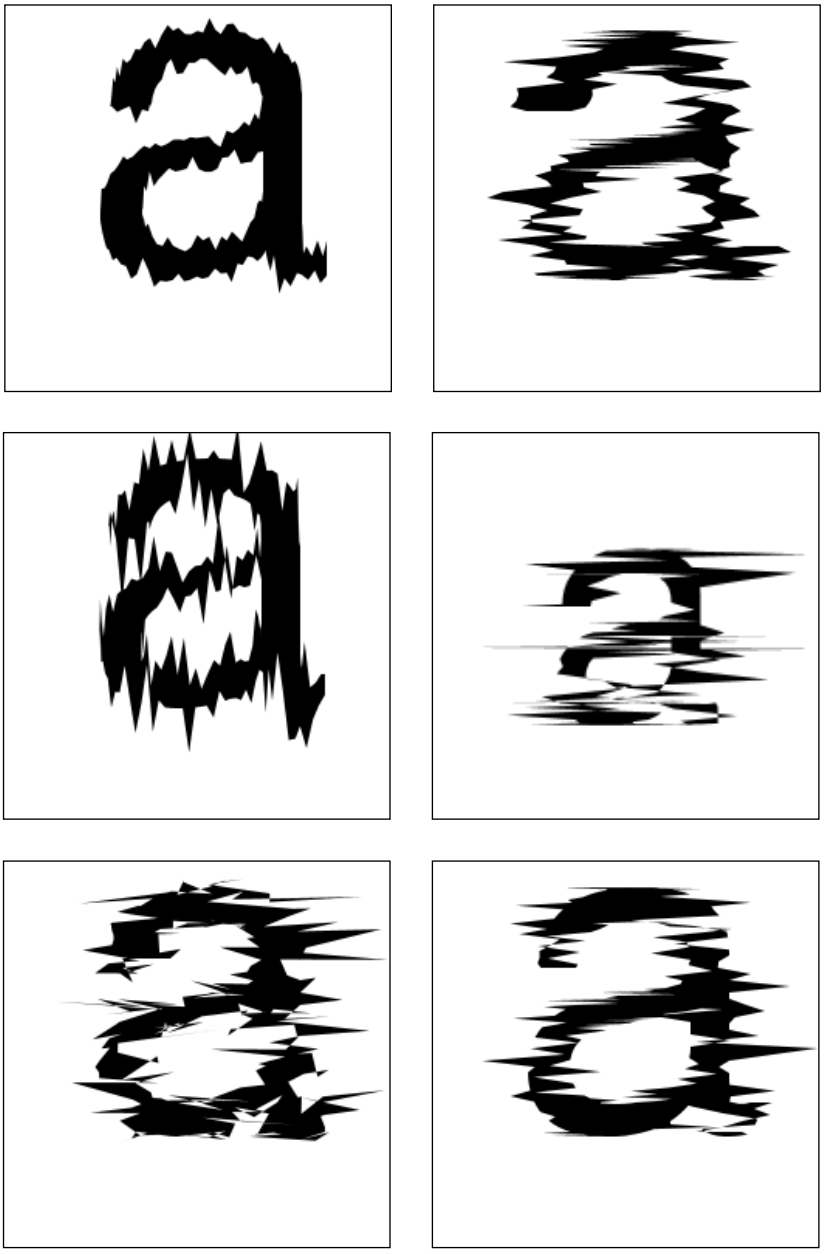
Figure 7.2.1.

Snapshot. Evolution of "glitched_type". Exploring distortions on the letter 'a'. Typeface: Helvetica Neue Regular.

Figure 7.2.2.

Evolving “glitched_type”, letter ‘a’.
Original and selected variations.

(Typography)



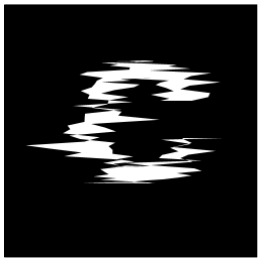
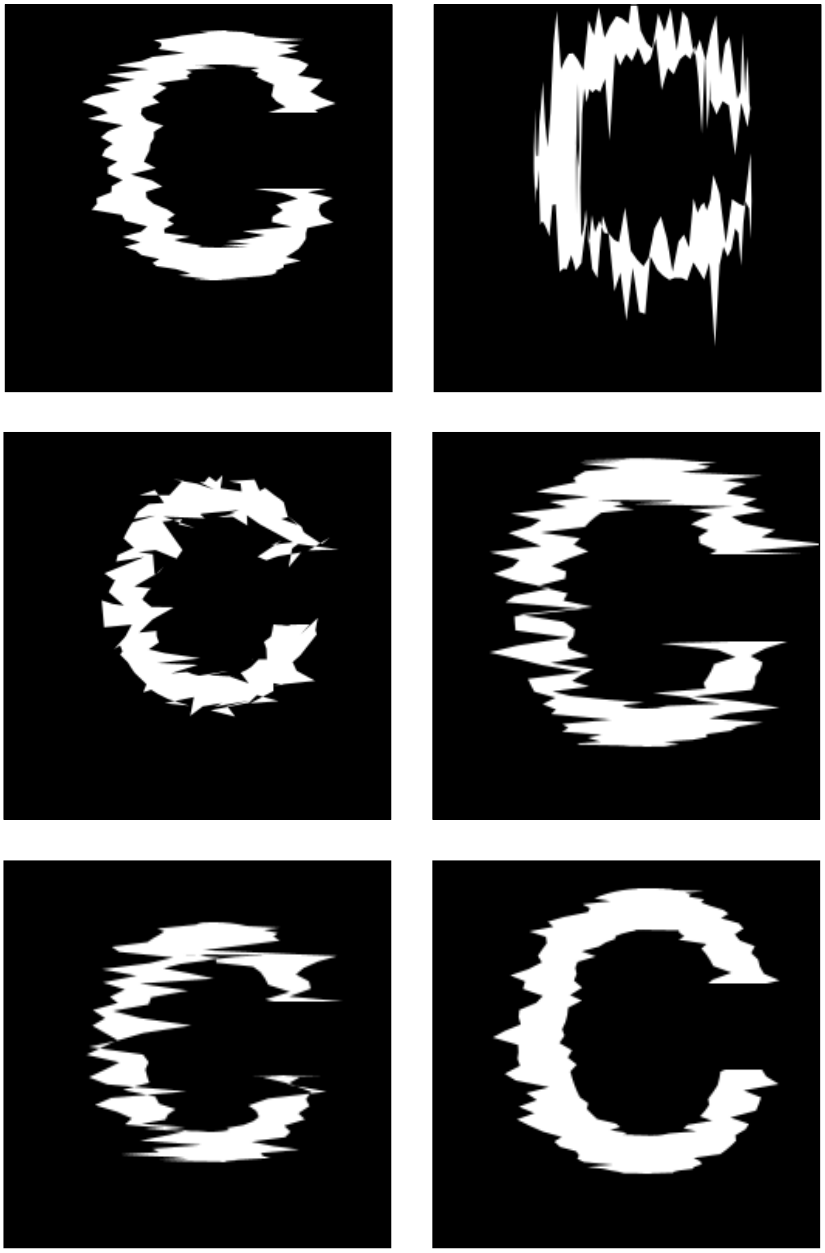


Figure 7.2.3.

Evolving “glitched_type”, letter ‘c’. Original and selected variations.

(Typography)



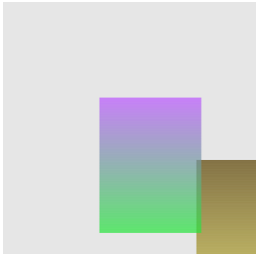


Figure 7.2.4.

Evolving “gradients”.
Original and selected variations.

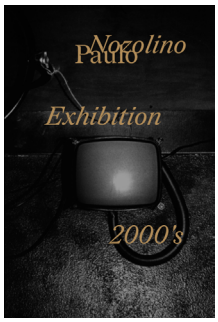
(Colours)

Figure 7.2.5.

Evolving “Nozolino exhibit”. Original and selected variations.

(Composition)

This sketch was made with a photograph by Paulo Nozolino Lisboa (2013).



In case the designer has a general idea of the poster he wants to create but does not know exactly how to position the elements and which colours to choose.

7.3 User evaluation

User testing is a valuable tool for designers to optimize a system's performance, elevate efficiency, and enhance overall satisfaction. It helps uncover behaviours, preferences, and pain points for a more user-centric and successful product (Damyanov, 2023). Stéfani Diniz, a Brazilian student, attended the University of Coimbra's Summer program in Portugal. Our paths crossed when she enrolled in an immersive research experience with us in the Department of Informatics Engineering. After briefly explaining our system's conception, we discussed the need to test it with users. She agreed to work on Processing sketches and feed EvoProteus. We intended to evaluate the system's performance and define the proper set of instructions to request from a designer before exploration. First, she was presented with this list of commands:

1. Create a sketch in Processing;
2. Choose the set of parameters you want to evolve;
3. Tag those parameters with our markup language;
Example:
Original variable → int radius = 0;
Tagged variable → int __radius = 10; //min:0 max:50
(The system only reads initializations.)
4. Run EvoProteus and upload your sketch (make sure to gather all code in a single file);
5. Feel free to calibrate the operators as you wish. In case you don't know anything about them, we suggest these values to begin:
Population size - 15 / *Elite size* - 1 / *Tournament size* - 3 / *Crossover rate* - 0.9 / *Mutation Rate* - 0.3 / *Mutation scale factor* - 0.05
6. Explore.

Notes:
(a) All variated sketches will be stored in the 'variations' folder;
(b) Your favourite ones will be stored in the 'favourites' folder;
(c) To save a sketch as a favourite, click on the respective window.

With this guide, Stéfani created thirteen distinct sketches. EvoProteus correctly imported them all, identified the search space and exported the outcomes. These experiments emphasized the necessity of developing the paused parameter component, which involves restricting the number of parameters evolved simultaneously and allows for greater control over the evolutionary process. We also identified the need to restart execution in case of unsatisfactory results. Stéfani expressed satisfaction with EvoProteus' performance and the diversity of achieved results. We plan to expand our system's evaluation, involving more designers in the future. Below is a selection of some sketches Stefáni made and explored with EvoProteus (Figures 7.3.1 to 7.3.3).

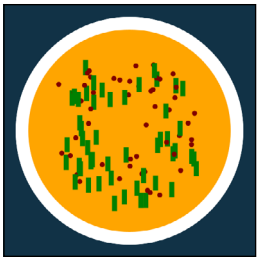
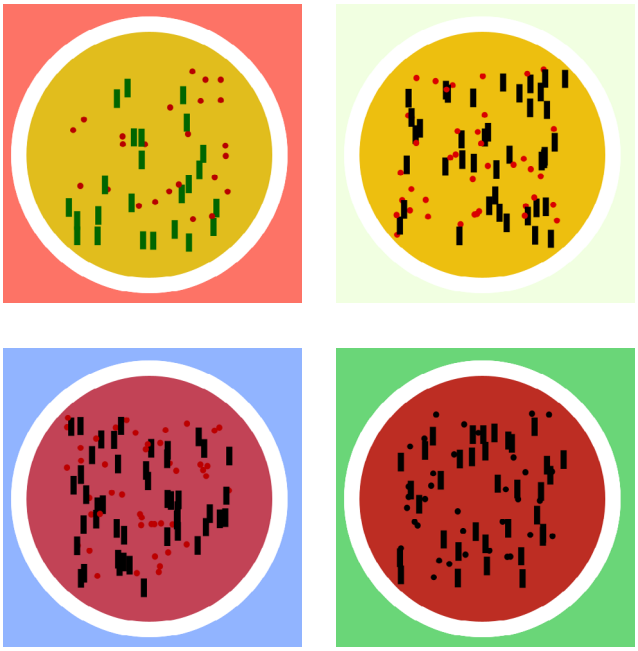


Figure 7.3.1.

Evolving "Soup" (after the University of Coimbra Canteen's soup) by Stéfani Diniz. Original and selected variations

A possible transition from soup to dessert.



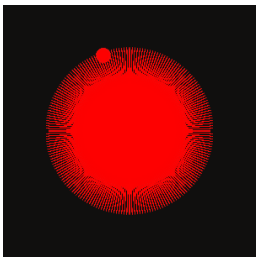


Figure 7.3.2.

Evolving “Constellations”
(after M13 Star Cluster Constellation)
by Stéfani Diniz
Original and selected variations.

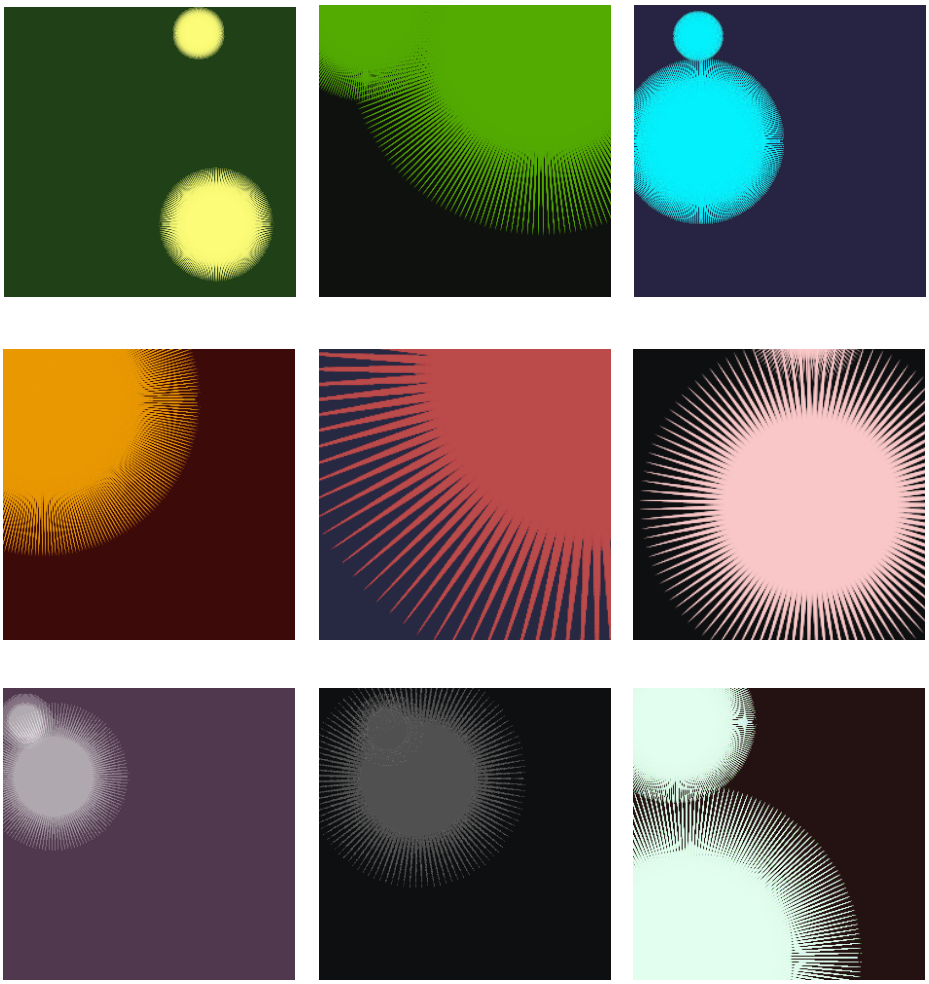
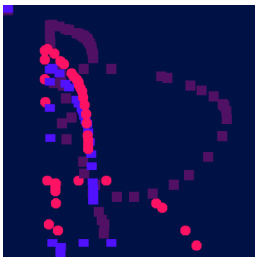
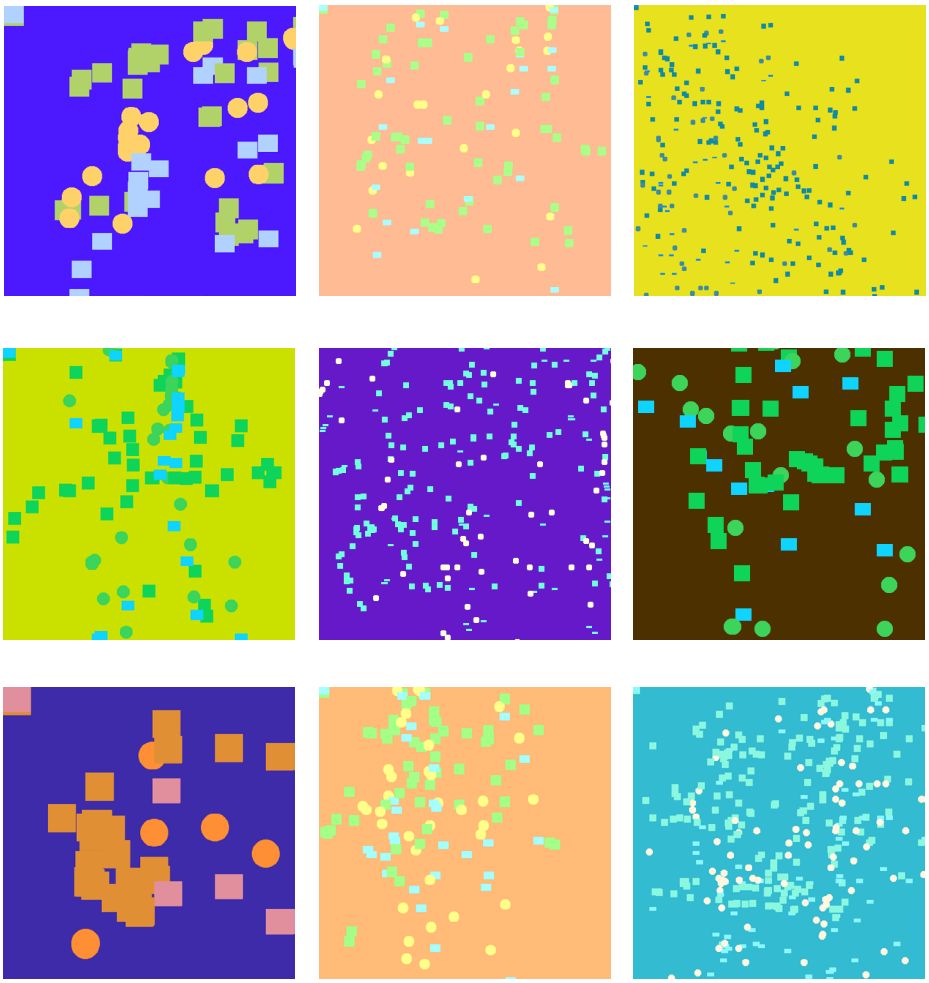


Figure 7.3.3.

Evolving “Disorder of objects” (about clutter) by Stéfani Diniz
Original and selected variations.





7.4 Experiments with FeedNPlay

Interaction is a crucial aspect of our system. As we got results, we began to consider other potential applications beyond typical desktop screen executions and looked for interactive approaches. Fortunately, the Department of Informatics Engineering (University of Coimbra) has a resource called FeedNPlay, one computer with a series of nine LCDs located along the busiest corridor after the entrance. These displays provide a dynamic digital platform that showcases a variety of content, such as images, videos, animations, text, information visualization projects, moving typography, and even live interactive artworks, installations, and media experiences. We started planning an EvoProteus application that could work within this context. "Galápagos" (Figure 7.4.1.) is an interactive media installation created by Karl Sims and first installed at the NTT InterCommunication Center in Tokyo from 1997 to 2000 (Sims, 1997).

> Sims (1997) describes "Galápagos" as an *interactive Darwinian evolution of virtual "organisms". Twelve computers simulate the growth and behaviors of a population of abstract animated forms and display them on twelve screens arranged in an arc. The viewers participate in this exhibit by selecting which organisms they find most aesthetically interesting and standing on step sensors in front of those displays* (p.1).

We were drawn to this project for two reasons. (a) It aligns with EvoProteus as a system that evolves visuals based on user feedback. (b) There are conceptual similarities between the platform used for Tokyo and FeedNPlay. We also found the pedal-based evaluation particularly interesting. It is simple, elegant and requires minimal effort. We adapted Sim's installation, sketching an initial plan for our own. Figure 7.4.2. depicts the nine screens available on FeedNPlay, each connected to one pedal. We decided on a fixed population size of nine individuals, with an additional pedal to generate a new population.

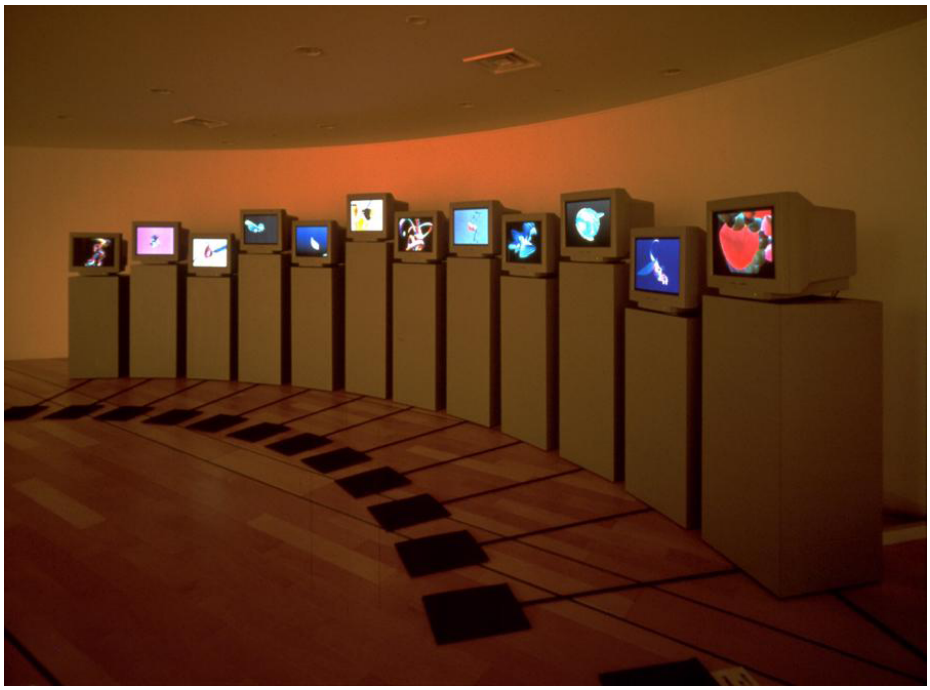


Figure 7.4.1.
"Galápagos" installation by Karl Sims at the ICC in Tokyo (1997).

With this plan in mind, we started preparing our installation. In terms of technical aspects, there are some key points to note. (a) We reduced the control panel usage to the minimum number of actions: input upload, parameter and genetic settings. From there, it stays hidden till the execution stops. (b) We integrated the pedals into FeedNPlay with an Arduino that establishes communication. Whenever a user presses a pedal, this device sends a signal to the system identifying the pedal's number. With that information, the system sends a signal to the sketch window associated with that number, prompting the execution process to stop (using our socket component). The window closes, and the system recognizes the solution as unsatisfactory. Pedal "ten" is centrally positioned and generates a new population. The next pages illustrate the whole process from preparation to experimentation and results (Figures 7.4.3 to 7.4.12).

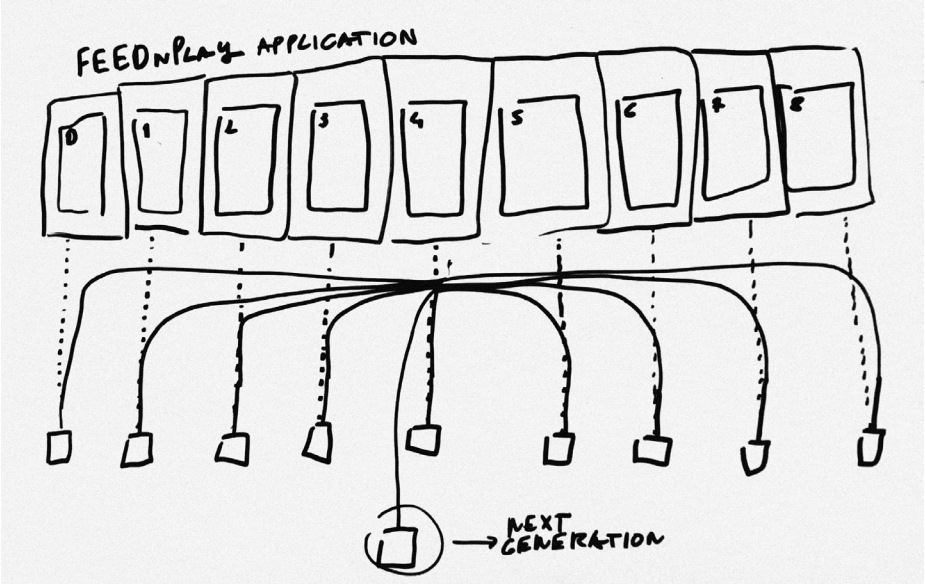


Figure 7.4.2.

Plans for an EvoProteus installation with FeedNPlay. Each rectangle represents a screen, the numbers correspond to individuals indexes from a sized-nine population.

Figure 7.4.3.

Preparation phase. Software adjustments

Figure 7.4.4.

Preparation phase. Testing pedals.

Figure 7.4.5.

Preparation phase. Pedals arrangement.

Figure 7.4.6. →

Experimentation with sketch “Protean poster“.

Input uploading and generation of variations.

↓ Figure 7.4.7.

Experimentation with sketch “Protean poster“.

Closing the worst by pressing its assigned pedal.

↓ Figure 7.4.8.

Experimentation with sketch “Protean poster“.

A rendered population.

↓ Figure 7.4.9.

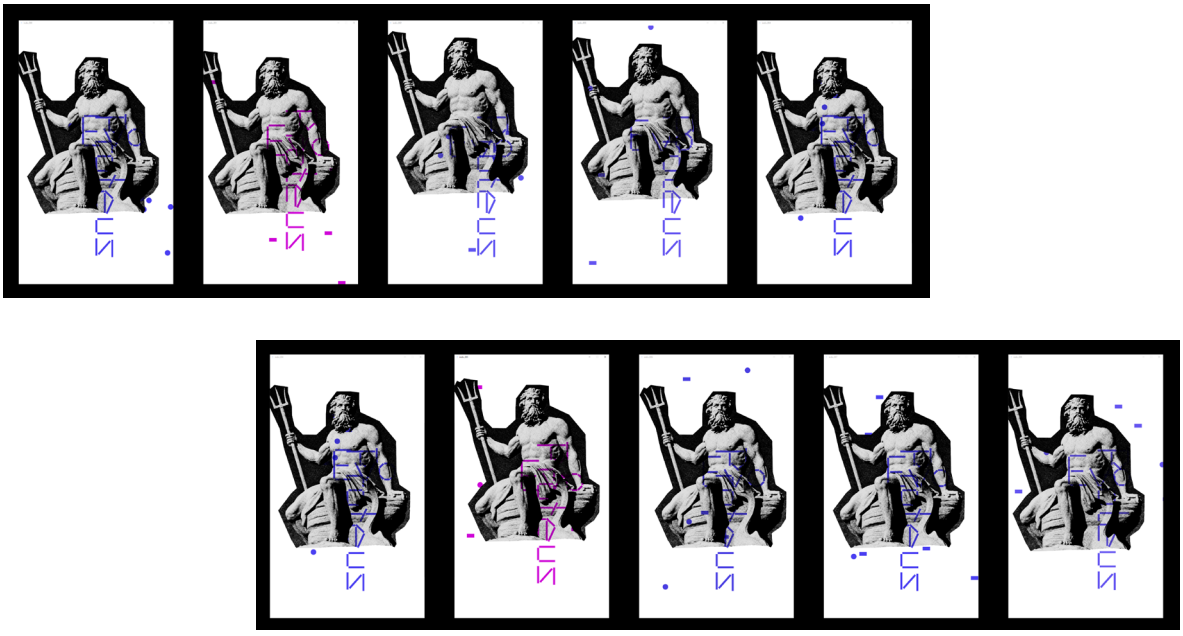
Experimentation with sketch “Protean poster“.

Evolving populations.

Our experiment has shown that the system can function in various environments. We have observed that different platforms encompass distinct experiences. Using EvoProteus on a desktop is more efficient, allowing for quick evaluation, diverse genetic schemes, and real-time manipulation of operators and parameters. Using it in a more physical environment within a specific context may also produce interesting results, including multiple users collectively evaluating solutions and evolutionary processes that run through an entire day. The choice of platform will always depend on the intention, whether to prioritize interactive experiences or the system's performance as a working tool.

Figure 7.4.10.

Snapshot. An evolved population of posters, created with FeedNPlay.



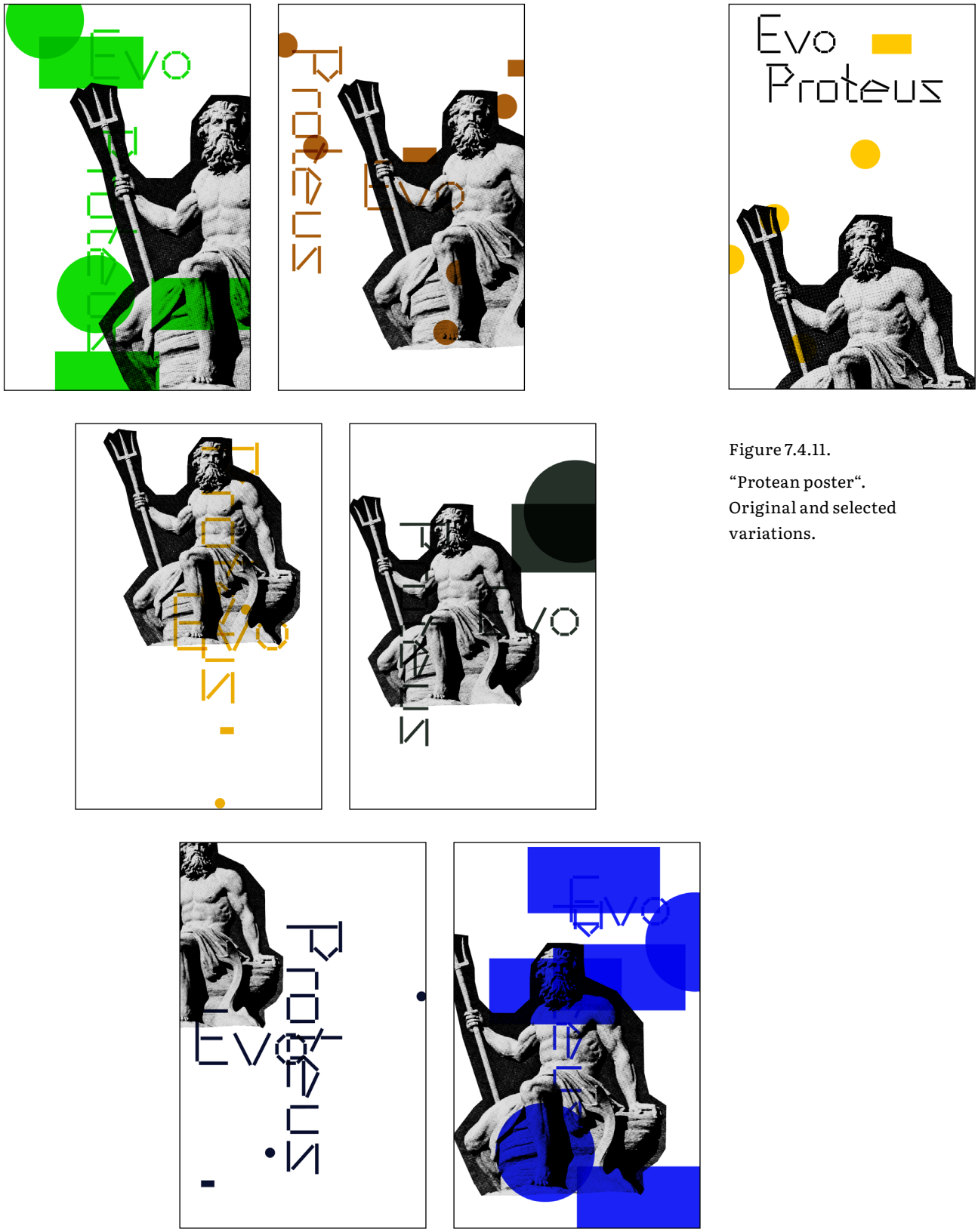


Figure 7.4.11.

"Protean poster". Original and selected variations.

CHAPTER 8.

FUTURE WORK

EvoProteus has the potential for further exploration beyond the scope of this dissertation. Below, we outline some directions for future research.

8.1 Automatic Fitness

Early in development, we considered automatic fitness. Due to time limitations, it was not possible. However, it is the next logical step in development. By automatic fitness, we mean a non-user-guided evaluation of candidate solutions. That way, it becomes possible to run larger populations (for now, large pops. may cause cognitive distress) and evolve individuals without interruption. Given the system's generic nature, the most interesting path for us is the combination with prompts. Prompts are textual instructions or descriptions written to get a desired result from a language-based model (Martins et al., 2023, pp.182). The user uploads a sketch and prompts the desired outcome. With this information, the system generates different variations, captures an image of each one, and compares it to a reference image of the desired result. How closely the proposed solution matches the example gives the fitness score. Variations far from the reference receive lower ratings, while those close receive higher ones. With this model, a user could create a sketch, upload it to the system, specify a desired outcome, take a break, and return later to find that the system had generated new solutions visually aligned with his intentions. This scenario would save time and effort for designers.

8.2 Other ideas

I.Multicast architeture.

As previously stated, our socket architecture operates using unicast packets. This means a unique server must be opened for each client to exchange data with the system. While practical, this solution lacks elegance as several servers may need to be opened for each execution. With IP multicast, data can be transmitted to multiple recipient clients through a single data stream (IONOS, 2023). In practice, only one server must be opened from the system's side to communicate with the entire

population. However, this approach presents technical challenges, such as dealing with concurrency when different clients attempt to send information simultaneously. Despite these challenges, multicast could be an efficient, low-resource approach.

II. Processing embed tool.

Our code is written in Processing, which has a range of valuable tools that can be used to attend specific design purposes. Exploring ways to integrate EvoProteus into the Processing Environment's available tools menu would make it even more convenient and efficient for users.

III. Web application.

The system's local development limits access for designers around the globe. EvoProteus as a web application would be an interesting transition. Even though it would require a significant reconfiguration of the interface, particularly the window architecture, integrating the system into the web would increase accessibility and open up possibilities to work with inputs written in p5.js.

CHAPTER 9.

DISCUSSION AND CONCLUSION

In this dissertation, we realise parametric design is quickly becoming a standard in the graphic design field. The designer defines a set of rules, variables and relationships between those variables to generate visual artefacts using computational tools. This mechanism allows for the generation of multiple parameter-based design solutions. However, most designers only produce one artefact at a time. They must constantly change the values assigned to each variable to obtain a different result from the previous one. What we have tried to prove in this work is that it is possible to automate this process of exploring design possibilities if we load the algorithm written by the designer into a machine that can identify the parameters in question and produce numerous solutions much more quickly and with greater diversity. We also realised that using an evolutionary algorithm could grant this machine the ability to evolve groups of solutions focusing on the designer's expectations. After some theoretical framework and schematisation, we developed a machine called EvoProteus.

Through a series of tests, we obtained surprising results that demonstrated the potential of our concept. Using our system, we often converged on good solutions that could not be generated manually due to human time constraints and potential cognitive limitations to assign values with high degrees of specificity (for example, values with numerous decimal places). We also experimented with different ways of interaction with our tool, which allowed us to realize its versatility in various environments. In addition to automating solution searches, our machine can promote collaborative design experiments involving multiple players in exploring an artwork. Further testing in different contexts may reveal additional uses beyond what we already perceived in this project. The issue of a user-centred evaluation of the solutions' quality is the system's main limitation to date. Exploring ways to automate this process should be the next step in development. Such an advance could establish the computer as an even more helpful design partner. It could improve and speed up design processes, such as producing distinct pieces for a visual identity. While many avenues may be explored with EvoProteus, we believe the system already possesses many advantages, and we have successfully met our objectives. This dissertation has taken significant steps towards what, in the future, can become a parametric, automated generative design paradigm.

REFERENCES

Barasch, M. (1997). The Language of Art: studies in interpretation. NY, USA: New York University Press. ISBN 0-8147-1255-X.

Bentley, P. J. (1999). An Introduction to Evolutionary Design by Computers. In P. Bentley (ed.), *Evolutionary Design by Computers*, *San Francisco*: Morgan Kaufmann, pp. 1–73.

Burnier, A. (n.d). CIDDIC. https://www.andreburnier.com/project/ciddic

Collins Dictionary (n.d). protean. In *collinsdictionary.com*. Retrieved July 15, 2023, from https://www.collinsdictionary.com/dictionary/english/protean

Cross, N. (2000). *Engineering design methods: Strategies for product design* (3rd ed.). John Wiley & Sons.

Damyanov, M. (2023). Guide to user testing: learn what users really want. In *Dovetail.com*. Retrieved May 12, 2023, from https://dovetail.com/ux/user-testing/

Dawkins, R. (1986). The Blind Watchmaker, New York: W. W. Norton & Company.

IONOS. (2023, March 21). What is Multicast? In *IONOS Digital Guide*. Retrieved August 11, 2023, from https://www.ionos.com/digitalguide/server/know-how/multicast/

Fry, B. and Reas, C. (2022). *Welcome to Processing!* (n.d.). In *Processing.org*. Retrieved November 14, 2023, from https://processing.org/

Galanter, P. (2003). What is Generative Art? Complexity theory as a context for art theory. New York, NY, USA: Interactive Telecommunications Program, New York University. http://philipgalanter.com/

Galvan, M. (2020, October 29). A brief history of graphic Design. In *Medium.com*. Retrieved September 21, 2022, from https://uxdesign.cc/a-brief-history-of-graphic-design-90eb5e1b5632

Glez-Morcillo, C., Martin, V. J., Vallejo, D.,Castro-Schez, J. J. and Albusac, J. (2010). Gaudii: An Automated Graphic Design Expert System. In *Proceedings of the Twenty-Second Innovative Applications of Artificial Intelligence Conference* (IAAI-10), pp.1775-1780.

Gradišar, L.; Klinc, R.; Turk,Ž.; Dolenc, M (2022). Generative Design Methodology and Framework Exploiting Designer-Algorithm Synergies. In *Buildings* 2022, 12(12), 2194; https://doi.org/10.3390/buildings12122194

Halford, G.S.; Baker, R.; McCredden, J.E.; Bain, J.D (2005). How Many Variables Can Humans Process? In Psychol. Sci. 2005,16, 70–76. DOI:[10.1111/j.0956-7976.2005.00782.x] (http://dx.doi.org/10.1111/j.0956-7976.2005.00782.x)

Hewitt, J. (2008, February 13). Flexible Consistency, Consistent Flexibility. Speak Up. In Retrieved March 10, 2023, from https://www.underconsideration.com/speakup/archives/004431.html

Holland, J. H. (1992). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. The MIT Press

Hollis, R. (1994). Graphic Design: a concise history. In Internet Archive. New York: Thames and Hudson.

Janssen, P., Frazer, J. and Ming-Xi, T. (2002). Evolutionary Design Systems and Generative Processes. In *Applied Intelligence 16*, 119–128. Kluwer Academic Publishers. The Netherlands.

Jebari, K.; Madiafi, M. (2013). Selection methods for genetic algorithms. In. *J. Emerg. Sci.* 2013, 3, 333–344.

Katoch, S., Chauhan, S.S. & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. In *Multimed Tools Appl* 80, 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

Kour, H., Sharma, P., and Abrol, P. (2015). Analysis of Fitness Function in Genetic Algorithms. In *International Journal of Scientific and Technical Advancements*, 1(3), 87-90.

Krish, S. (2011). A practical generative design method. In *Computer-Aided Design*, *43*(1), 88–100. https://doi.org/10.1016/j.cad.2010.09.009

Levanier, J. (2022). What is postmodern design: how the reigning style of the late 20th century works. In *99designs*. Retrieved January 19, 2023, from https://99designs.com/blog/design-history-movements/postmodern-design/

Lewis, M. (2008). Evolutionary Visual Art and Design. In Romero, J., Machado, P. (eds) *The Art of Artificial Evolution*. Natural Computing Series. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-72877-1_1

Li, S. (2019) Generative design. In *Medium.com*. Retrieved November, 12, 2022, from https://medium.com/@sixuanli/generative-design-61cdb7fa89fb

Lopes, D., Correia, J.N. and Machado, P. (2020). Adea – Evolving Glyphs for Aiding Creativity in Typeface Design. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, New York, NY, USA, 2020, pp. 97-98.

Lutkevich, B. (n.d). race condition. In *TechTarget*. Retrieved May 15, 2023, from https://www.techtarget.com/searchstorage/definition/race-condition

Manaranche, A. (2014, July 26). Lust – Posterwall. In *Index Grafik*. Retrieved November 29, 2022, from http://indexgrafik.fr/lust-posterwall/

Martins, T. (2021). Automated Evolution For Design. Doctoral thesis, University of Coimbra.

Martins, T., Cunha, J. C., Correia, J. and Machado, P. (2023). Towards the Evolution of Prompts with MetaPrompter. In *Artificial Intelligence in Music, Sound, Art and Design: 12th International Conference, EvoMUSART 2023, Held as Part of EvoStar 2023, Brno, Czech Republic*, April 12–14, 2023, Proceedings, Apr 2023, Pages 180–195. https://doi.org/10.1007/978-3-031-29956-8_12

McCormack, J., Dorin, A. and Innocent, T. (2004). Generative Design: a paradigm for design research. In Redmond, J. [et al](http://et.al/). (eds) *Proceedings of Futureground*, Design Research Society, Melbourne.

McNeil, P. and Muir H. (2017). Cover Process In *Eye* 94, pp. 104–105.

Meggs, P. B. (1992). Type & Image. In *New York: John Wiley & Sons, Inc.*

Meggs, P. B., Purvis A. (2006). Meggs' History of Graphic Design (4th e.). John Wiley & Sons.

Munken Paper Mill and Hübner P.(2022). The Munken Creator. *Munken*. Retrieved December 05, 2022, from https://colab.munken.com/about-munken-creator

Önduygu, D. C. (2010). Graphagos: Evolutionary Algorithm as a model for the creative process and as a tool to create graphic design products. M.A, Visual Arts and Visual Communication Design. Supervisor: Elif Ayiter. Spring 2010.

Pereira, F. A., Martins, T., Rebelo, S. and Bicker J. (2019). Generative Type Design: Creating Glyphs from Typographical Skeletons. In *Artech 2019, 9th International Conference on Digital and Interactive Arts*, October 23–25, 2019, Braga, Portugal. 8 pages. https://doi.org/10.1145/3359852.3359866

Peterson, H. P. (1965). The Digital Mona Lisa. In *Computers and Automation*. Newtonville, Mass. (USA): Berkley Enterprise, Inc.

Quiroz, J. C., Banerjee, A., Louis, S. J. and Dascalu, S. M. (2009). Document Design with Interactive Evolution. In *Berlin, Heidelberg: Springer Berlin Heidelberg*, 2009, pp.309–319. isbn:978-3-642-02937-0.doi:10.1007/978-3-642-02937-0_28.

Raikar, S. Pai (n.d). Bombe. In *Encyclopedia Britannica*. Retrieved January 20, 2023, from https://www.britannica.com/topic/Bombe

Rebelo, S., Bicker, J. and Machado, P. (2020). Evolutionary Experiments in Typesetting of Letterpress-Inspired Posters. In *Proceedings of the 11th International Conference on Computational Creativity* (ICCC'20), 110-113. Association for Computational Creativity (ACC).

Rodenbröker, T. (2022, June 21). What is Creative Coding? In *timrodenbroeker.de*. Retrieved October 20, 2022, from https://timrodenbroeker.de/what-is-creative-coding/

Samara, T. (2007). Design Elements: A Graphic Style Manual. Rockport Publishers. ISBN: 978-1-592-53261-2.

Shacklett, M. E., Novotny, A. & Kate, G. (n.d.). What is TCP/IP? In *TechTarget*. Retrieved March 10, 2023, from https://www.techtarget.com/searchnetworking/definition/TCP-IP

Shim, K. (2020, February 20). Computational Approach to Graphic Design. In *The International Journal of Visual Design*, 14 (1): 1-9. doi:10.18848/2325-1581/CGP/v14i01/1-9.

Silva-Jetter, J. (2012). Designing through the loop: programming as a tool for aesthetic creation in the field of graphic Design. In Farias, Priscila Lena; Calvera, Anna; Braga, Marcos da Costa & Schincariol, Zuleica (Eds.). *Design frontiers: territories, concepts, technologies*. São Paulo: Blucher. DOI 10.5151/design-icdhs-068**Sims, K. (1991).** Artificial Evolution for Computer Graphics. In *Computer Graphics*, 25(4), July 1991, pp. 319-328. (ACM SIGGRAPH '91 Conference Proceedings, Las Vegas, Nevada, July 1991.)

Sims, K. (1997). Galápagos. In *karlsims.com*. Retrieved july 04, 2023, from karlsims.com/galapagos/

Singh V., Gu. N. (2012). Towards an integrated generative design framework. In *Design Studies*, 33(2), 185-207. https://doi.org/10.1016/j.destud.2011.06.001

Slowik, A. and Kwasnicka, H. (2020). Evolutionary algorithms and their applications to engineering problems. In *Neural Comput & Applic* 32, 12363–12379 (2020). https://doi.org/10.1007/s00521-020-04832-8

The Editors of Encyclopaedia Britannica (n.d). Proteus. In *Encyclopedia Britannica*. Retrieved May, 29, 2023 from https://www.britannica.com/topic/Proteus-Greek-mythology

Woodbury, R. (2010). Elements of Parametric Design. NY: USA New York: Routledge. ISBN: 978-0-415-77987-6.

APPENDIX

Figure 11.1.

The process is the project. Experimentations with Sketch “Gradients”.

Figure 11.2.

The process is the project. Experimentations with Sketch “Bauhaus”.

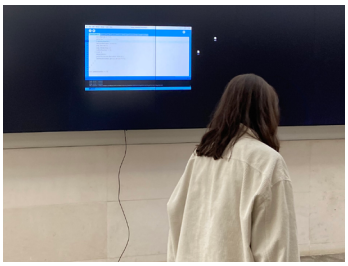


Figure 11.3.

Snapshots of the FeedNPlay experiment.

Figure 11.4.

The process is the project. Experimentations with Sketch “Protean poster” (in FeedNPlay).

Figure 11.5.

Sketch “Protean poster”, other outcomes.

This document was composed
with Literata typeface.

Designed by Irene Vlachou, Veronika Burian,
Vera Evstafieva and José Scaglione.
(2020)