SAGE: A Tool for Intuitive Design of Shape Grammars

Mariana Coutinho Guerreiro

Master in Computer Engineering

Supervisor:
PhD Pedro Figueiredo Santana, Associate Professor,
Iscte – Instituto Universitário de Lisboa

Supervisor:
PhD Filipe Alexandre Azinhais dos Santos, Assistant Professor,
Iscte – Instituto Universitário de Lisboa

October, 2025

TECHNOLOGY
AND ARCHITECTURE

Department of Information Science and Technology

SAGE: A Tool for Intuitive Design of Shape Grammars

Mariana Coutinho Guerreiro

Master in Computer Engineering

Supervisor:
PhD Pedro Figueiredo Santana, Associate Professor,
Iscte – Instituto Universitário de Lisboa

Supervisor:
PhD Filipe Alexandre Azinhais dos Santos, Assistant Professor,
Iscte – Instituto Universitário de Lisboa

October, 2025

I learned a lot, by the end of everything.

Acknowledgment

I would like to express my sincere gratitude to my supervisors, Pedro Santana, and Filipe Santos, for their continuous guidance, encouragement, support and for reminding me to stay pragmatic whenever I started to over complicate things.

I would also like to thank my friends for being there during the most difficult moments and for always believing in me.

To my boyfriend, thank you for celebrating every completed part of this thesis with me, and for your endless motivation and kindness throughout the entire process.

Finally, I am deeply grateful to my family for their unconditional love and unwavering support, which made it possible for me to reach this milestone.

Resumo

As Gramáticas de Formas são sistemas baseados em regras para a geração de composições espaciais, oferecendo uma abordagem formal e flexível para a criação de designs. No entanto, as ferramentas existentes frequentemente exigem conhecimentos avançados de sistemas formais e programação, o que limita a sua acessibilidade a um público mais amplo.

Esta dissertação apresenta a SAGE (ShApe Grammar assisted Environment), uma ferramenta visual desenvolvida com base na plataforma Alternative Shaper, concebida para tornar as Gramáticas de Formas mais intuitivas e acessíveis a utilizadores de diferentes perfis. Enquanto o Alternative Shaper fornece o motor generativo subjacente, SAGE centra-se no desenvolvimento de uma interface clara e orientada ao utilizador, que permite criar formas, definir regras de transformação e compor gramáticas procedurais através de um ambiente de programação visual integrado. Desenvolvido através de uma abordagem de prototipagem iterativa, o SAGE procura aproximar os sistemas generativos formais de modelos de interação mais intuitivos, promovendo a integração das Gramáticas de Formas em fluxos de trabalho de design interativo.

Uma avaliação sumativa envolvendo 30 participantes revelou uma pontuação média de 78.1 no System Usability Scale (SUS), correspondente à categoria "Bom–Excelente", indicando uma experiência de utilização positiva e confirmando que o SAGE reduz eficazmente a barreira de entrada às Gramáticas de Formas. Estes resultados demonstram o seu potencial como base para futuras aplicações educativas e profissionais no domínio do design generativo.

Palavras Chave: Gramáticas de Formas; Design Generativo; Sistemas baseados em Regras; Interação Humano-Computador

Abstract

Shape Grammars are rule-based systems for generating spatial compositions, offering a formal and flexible approach to design generation. However, existing tools often require advanced knowledge of formal systems and programming, limiting their accessibility to a broader audience.

This dissertation presents SAGE (ShApe Grammar assisted Environment), a visual tool built upon the Alternative Shaper platform to make Shape Grammars more intuitive and approachable to users of all backgrounds. While the Alternative Shaper provides the underlying generative engine, SAGE focuses on developing a clear, user-centred interface that allows users to create shapes, define transformation rules, and assemble procedural grammars through an integrated visual programming environment. Developed through an iterative prototyping approach, SAGE bridges the gap between formal generative systems and user-friendly interaction models, bringing Shape Grammar functionality closer to interactive design workflows.

A summative evaluation involving 30 participants yielded a mean System Usability Scale (SUS) score of 78.1 ("Good–Excellent"), indicating a positive user experience and confirming that SAGE effectively reduces the entry barrier to Shape Grammars. These results highlight its potential as a foundation for future educational and professional applications in generative design.

KEYWORDS: Shape Grammars; Generative Design; Rule-Based Systems; Human-Computer Interaction

Contents

List of Tables

List of Figures

List of Acronyms

DSR	Design Science Research
HCI	Human–Computer Interaction
SUS	System Usability Scale
UCD	User-Centred Design

CHAPTER 1

Introduction

1.1. Context and Motivation

Generative design has become an increasingly important paradigm in fields such as architecture, product design, and industrial design. By leveraging computational systems, designers are no longer constrained to manually exploring single alternatives, but can instead automate creative processes and efficiently generate, compare, and refine multiple design variations. This paradigm shift is particularly valuable in early design stages, where the exploration of a wide solution space can significantly influence the quality and innovation of the final outcome.

Within this domain, Shape Grammars, first introduced by Stiny and Gips in 1972 [1], stand out as a formalism capable of describing, analyzing, and generating designs through the recursive application of transformation rules. The flexibility of Shape Grammars allows them to represent both the syntactic and semantic aspects of a design, making them a powerful tool for capturing stylistic features, enforcing design constraints, and systematically exploring variations. Over the past decades, Shape Grammars have been applied to diverse areas, from architectural composition [2], [3], [4] to furniture and product design [5], [6], [7], and even art and decorative forms [8], [9], demonstrating their versatility as a generative system.

Despite their potential, Shape Grammars remain largely inaccessible to non-experts. The formalisms involved are often mathematically and conceptually complex, requiring significant prior knowledge to be applied effectively. Existing tools, such as DESIGNA [10], IM-sgi [11], and QShaper [12], provide valuable implementations and confirm the applicability of Shape Grammars in design practice. However, they also highlight an important limitation: most of these tools assume that users already possess technical knowledge of the underlying formalism, which restricts their use to a niche audience of researchers or highly specialized designers.

To overcome these barriers, there is a clear need for tools that democratize the use of Shape Grammars, translating their theoretical richness into designer-oriented workflows that balance formal rigor with usability.

Recent Human–Computer Interaction (HCI) research offers new perspectives on this challenge. For instance, Brickify [13] illustrates how direct manipulation of visual tokens can make generative systems more accessible and expressive. By allowing designers to convey intent through graphical interactions rather than abstract parameters or textual prompts, such approaches reconcile human creativity with computational formalism.

Inspired by these developments, this dissertation argues that Shape Grammars, when coupled with visual interaction and rule-based control, can provide a natural framework for guiding and constraining generative design.

Building upon this premise, the present work extends the Alternative Shaper [14], a pre-existing Shape Grammar interpreter and design generation engine, by developing SAGE (ShApe Grammar assisted Environment), a new user-centered interface layer designed to make such capabilities more transparent, learnable, and visually accessible. By integrating Blockly [15], a visual programming library, SAGE allows grammar composition through a drag-and-drop interface, thereby reducing the cognitive load typically associated with textual or symbolic representations.

Developed through an iterative, User-Centered Design (UCD) process, the proposed tool aims to support both novices, who may be encountering generative design concepts for the first time, and experts seeking a more efficient and streamlined workflow.

1.2. Research Questions

In light of the motivations outlined above, this dissertation seeks to address the following research questions:

- **RQ1:** Which graphical representations are most effective for supporting the visual design of Shape Grammars, particularly for novice users?
- **RQ2:** Which interaction metaphors best facilitate the intuitive creation, manipulation, and understanding of shapes, rules, and grammars within a visual environment?
- **RQ3:** How can iterative prototyping and user feedback inform and validate these representations and interaction models to enhance the accessibility, clarity, and usability of Shape Grammar-based design tools?

By addressing these questions, the dissertation contributes to bridging the gap between the formal expressiveness of Shape Grammars and their practical applicability in design contexts, demonstrating how Human–Computer Interaction (HCI) methodologies can enhance their adoption and usability.

1.3. Methodology

To address the research questions, this dissertation follows the Design Science Research (DSR) methodology [16], which is widely used in information systems and software engineering for the creation and evaluation of innovative artifacts. DSR is particularly suitable for this work because it emphasizes both the practical utility of the developed solution and its theoretical contribution.

The methodology unfolds through the following stages:

(1) **Problem Identification** — defining the challenges associated with existing Shape Grammar tools and identifying the gap in accessibility and usability.

(2) **Design & Development** — extending the Alternative Shaper platform by developing the SAGE interface, iteratively refining its visual components and interaction model based on user needs.
(3) **Demonstration** — applying the tool in representative design scenarios to illustrate its capabilities and potential.
(4) **Evaluation** — assessing the effectiveness of the tool through usability testing, comparing it to existing solutions, and analyzing user feedback.
(5) **Communication** — disseminating the results and discussing the implications of the work for the fields of generative design, computational creativity, and HCI.

This structured approach ensures that the proposed solution is not only technically sound but also aligned with the needs of its intended users, ultimately contributing to both academic knowledge and practical design practice.

1.4. Objectives

Based on the DSR methodology described above, this dissertation defines a set of concrete objectives that operationalize its research questions and guide the development of the proposed artifact. These objectives ensure that the methodological framework is effectively applied toward the creation, implementation, and validation of a usable and educational Shape Grammar environment.

Specifically, the objectives are as follows:

(1) To analyze the limitations of existing Shape Grammar tools in terms of accessibility, usability, and integration with contemporary design workflows;
(2) To build upon the Alternative Shaper platform by designing a new user interface layer (SAGE) focused on clarity, interactivity, and user experience, while maintaining compatibility with its generative engine;
(3) To prototype and implement the proposed tool through an iterative, user-centered process integrating visual programming via Blockly;
(4) To evaluate the usability, clarity, and educational value of the developed prototype through user testing and feedback analysis;
(5) To identify potential directions for future development and refinement, bringing the prototype closer to a deployable and extensible solution for design education and practice.

Together, these objectives align the methodological process with the research goals, ensuring that both theoretical and practical contributions support the broader aim of making Shape Grammars more accessible and usable in creative design contexts.

1.5. Dissemination

The conceptualization and early-stage development of the visual Shape Grammar tool were presented in a paper submitted to the International Conference on Graphics and Interaction (ICGI 2025). The paper focused on the design rationale, interface conceptualization, and low-fidelity prototype created in Figma, outlining the foundations of the project and its intended interaction model. The submission was accepted as a full paper for oral presentation at the conference, which will take place on November 13–14, 2025.

While this publication disseminates the initial phase of the project, the present dissertation extends the work by describing the implementation of the full application and presenting the results of a summative evaluation, conducted to assess its usability and effectiveness.

1.6. Document Structure

Following this introductory chapter, the dissertation is organized into more five main chapters.

Chapter 2 reviews the theoretical and practical foundations underpinning this study. It begins by outlining the adopted literature review methodology and explores key concepts across the fields of Computer-Aided Design (CAD), Generative Systems, Generative Design, and Shape Grammars, highlighting existing tools and identifying current gaps and opportunities.

Chapter 3 presents the design and development of SAGE. It first introduces the overall structure and core features of the final system, and then retraces the iterative prototyping process through which the interface and interaction model evolved. This chapter establishes the design rationale that guided implementation, demonstrating how feedback from successive evaluation rounds shaped the final tool.

Chapter 4 details the development phase, including the chosen environment, system architecture, and technical integration of the Blockly framework. It demonstrates how the conceptual ideas were translated into a functional application.

Chapter 5 reports on the summative evaluation conducted to assess the usability, clarity, and effectiveness of the final system. It explains the evaluation goals, methodology, and applied metrics, followed by both quantitative and qualitative analyses of participant feedback.

Chapter 6 summarizes the main findings of the study, revisits the research questions, and discusses future directions for improvement and further development.

CHAPTER 2

Related Work

The development of a design generation tool based on Shape Grammars builds upon an evolving interdisciplinary landscape that intersects computational design theory, generative systems, and HCI. From the early adoption of Computer-Aided Design (CAD) to the rise of parametric and generative design paradigms, the trajectory of digital design tools reflects a gradual shift from manual geometric control toward computationally assisted creativity. As these tools became increasingly interactive and user-oriented [17], [18], the relationship between designer and computer evolved from one of control to one of collaboration.

Within this continuum, Shape Grammars represent a foundational formalism capable of expressing design languages through transformation rules. Despite their theoretical richness, however, the practical adoption of Shape Grammars in design practice remains limited, largely due to usability challenges, a lack of intuitive visual environments, and difficulties integrating grammar-based logic with interactive workflows.

This chapter reviews the evolution of digital design frameworks relevant to Shape Grammars, from CAD and generative systems to contemporary HCI paradigms. The review aims to identify conceptual and practical gaps that justify the development of an interface designed to make Shape Grammar-based design more transparent and approachable. In particular, it positions the Alternative Shaper as the most comprehensive existing interpreter of parametric Shape Grammars, whose procedural engine forms the computational foundation for the tool developed in this dissertation. The present work builds upon that foundation to design a new interface layer focused on learnability, feedback, and exploratory engagement.

2.1. Literature Review Methodology

A literature review was conducted with the aim of collecting academically relevant information regarding the foundations, applications, and evolution of Shape Grammars in design contexts. Because Shape Grammars are a mature concept, with origins in the 1970s [1], it was necessary to combine both classic works, which establish their theoretical and formal basis, and more recent publications focusing on usability, parametric extensions, and integration into generative design workflows.

The review followed a snowballing approach, starting from foundational Shape Grammar references and expanding to subsequent publications that cited or extended them. This strategy was adopted because the field of Shape Grammars, while seminal, is relatively niche compared to broader areas such as CAD or generative design, and therefore lacks the extensive indexing and standardized taxonomies typically required for systematic reviews.

Although not conducted under a formal PRISMA protocol, the review was guided by its general principles of transparency and traceability. Searches were performed manually using academic databases such as Google Scholar, Scopus, the ACM Digital Library, and the Portuguese B-On (Biblioteca do Conhecimento Online), focusing on peer-reviewed publications written in English. Foundational works by Stiny and Gips (1972) served as the starting point, from which relevant citations were iteratively collected and analyzed.

Through this process, a broad set of publications related to Shape Grammars, computational design, generative systems, and HCI was identified. To facilitate analysis, the reviewed sources were first grouped into five thematic domains reflecting the conceptual evolution of Shape Grammars within computational design:

(1) Computer-Aided Design (CAD): to contextualize how digital tools first entered the design process and how they established the groundwork for rule-based systems.
(2) Generative Systems: to examine how concepts of rule-based generation (e.g., cellular automata, L-systems, agent-based models) and emergence (where unexpected patterns arise from simple rules) influenced design thinking and computational creativity.
(3) Generative Design: to trace the evolution towards goal-oriented, optimization-driven systems where computation plays an active role in exploring solution spaces.
(4) Shape Grammars: to focus on the specific formalism at the core of this research, including its theoretical foundations, parametric extensions, and applied tools.
(5) Human-Computer Interaction: to examine how usability, interaction design, and user experience considerations shape the accessibility and practical adoption of Shape Grammar tools.

After this categorization, publications were filtered according to four general criteria:

- Relevance to Shape Grammars: Does the publication directly address Shape Grammars or their applications in design?
- Generative Context: Does it relate to rule-based or computational generation approaches that inform or parallel to Shape Grammars?
- Practical Application: Does it contribute to the understanding of usability, parametric control, or integration with design workflows and interaction design?
- Historical Importance: Is it a foundational work that defines or frames Shape Grammar?

Following this filtering process, the final corpus combined foundational theoretical works with more recent studies on usability, parametric extensions, and HCI integration. The snowballing approach was ultimately deemed appropriate given the historical and exploratory nature of the topic, ensuring comprehensive coverage of both classical and contemporary perspectives without imposing the rigid inclusion criteria typical of systematic reviews.

2.2. Computer-Aided Design (CAD)

As defined by Zeid [19], Computer-Aided Design (CAD) refers to “the use of computer systems to assist in the creation, modification, analysis, or optimization of a design”. CAD systems primarily act as digital extensions of manual drafting, enhancing precision and efficiency while preserving user control over the design process. Tools such as AutoCAD [20] have long served as the backbone of architectural and industrial workflows, standardizing representation and supporting interoperability with downstream processes.

However, as Kalay [21] and Sass and Oxman [22] note, traditional CAD systems remain fundamentally representational rather than generative, focusing on documentation rather than exploration. Their core purpose is to depict geometry that already exists in the designer’s mind, relying on explicit user commands to construct and modify every element. As such, CAD environments describe and record design intent but do not autonomously propose or derive new solutions.

From an HCI perspective, traditional CAD environments can be seen as early examples of direct manipulation interfaces, where control and precision are prioritized over exploratory or creative engagement. The introduction of Graphical User Interfaces (GUIs) [23] made such systems more accessible, but their underlying interaction paradigm remained task-oriented rather than exploratory. As interaction research emphasizes [18], systems that rely solely on user-driven modeling tend to support efficiency but hinder reflection, discovery, and reinterpretation: aspects that are central to creativity-oriented design.

The evolution towards parametric and procedural modeling, as exemplified by Grasshopper 3D [24], marked a turning point where geometry is defined not directly but through sets of rules and parameters [25]. This logic-based workflow introduced computational design thinking, enabling designers to explore families of solutions by manipulating input parameters and delegating part of the creative process to the computer.

Despite these advances, parametric CAD tools still rely heavily on explicit user input and technical expertise, offering precision but limited support for emergent creativity. As Shneiderman [17] argues, effective digital tools for design should not only optimize performance but also support exploration and creativity. This limitation has motivated the development of new forms of computational and generative systems that rethink the interaction between designer and computer.

2.3. Generative Systems

Generative systems extend beyond parametric control by introducing autonomous or semi-autonomous processes capable of producing complex results from relatively simple rule sets. Unlike CAD environments, where geometry is explicitly constructed and manipulated by the user, generative systems leverage computation to reveal emergent patterns and behaviors that may not be predictable from the initial conditions alone [26], [27].

Classical examples include cellular automata, such as Conway's Game of Life [28], where simple local rules lead to globally complex behavior, and L-systems [29], originally developed by Lindenmayer to simulate plant growth. Agent-based models such as Boids [30] further exemplify this principle, where individual entities following minimal behavioral rules generate sophisticated group dynamics. These approaches demonstrate how rule-based systems can produce rich visual and structural outcomes without explicit user modeling.

Generative principles have also influenced digital media and entertainment, as seen in Spore [31] and procedural city generation techniques [32], where computational rules drive large-scale diversity and complexity. In design contexts, such systems introduce the idea of emergence: the ability for unexpected configurations to arise from predefined logic, which is central to computational creativity [26], [33].

By partially shifting creative agency to the computer, generative systems promote a co-creative paradigm in which designers act as curators of process rather than direct modelers of geometry. This dynamic redefines the designer–computer relationship as an ongoing dialogue rather than a sequence of commands. As McCormack and d'Inverno [34] argue, such interaction represents a partnership of creative exploration, where each iteration reveals new possibilities within a defined rule space.

From an HCI standpoint, this transition reflects a shift from efficiency-centered design toward systems that prioritize interpretability, feedback, and cognitive engagement [18], [35]. When users can understand and influence generative behavior through visual interaction, computation becomes a medium for reflection and discovery rather than mere automation. This perspective underscores the need for interfaces that make the underlying generative logic transparent, allowing users to comprehend, predict, and creatively manipulate emergent outcomes: a principle that directly informs the rationale for Shape Grammar-based tools.

2.4. Generative Design

Generative Design formalizes the co-creative paradigm introduced by generative systems, shifting from open-ended emergence to goal-oriented exploration of design alternatives. Rather than merely producing emergent forms, generative design integrates computational algorithms, performance criteria, and user-defined constraints to guide the search for solutions [25], [36], [37]. The designer specifies objectives, parameters, and evaluation metrics, while the system autonomously generates and evaluates variations, simulating aspects of creativity and supporting decision-making in complex problem spaces [38], [39].

This paradigm has given rise to the notion of Computational Design Thinking, as described by Oxman and Terzidis [37], [40]: a conceptual shift in which designers no longer manipulate geometry directly but instead define the rules and logics through which form emerges. The role of the designer becomes one of guiding and interpreting algorithmic behavior, engaging in a dialogue with the computational process rather than exerting total control over it. In this sense, generative design merges the rigor of computation with the intuition of human judgment, allowing designers to explore wide design spaces interactively and iteratively.

Notable tools in this domain include Houdini [41], widely used for procedural modeling in digital media, and Autodesk's Generative Design [42] platforms, which combine optimization algorithms with performance-driven criteria. These systems exemplify how computation can expand the design search space, but they also reveal important challenges: As Celani and Vaz [43] point out, generative systems often remain opaque and technically complex, demanding specialized knowledge that limits accessibility for many designers.

From a HCI perspective, this opacity represents a significant barrier to adoption. Systems that prioritize algorithmic power over interpretability tend to alienate users who lack technical expertise, restricting creativity to those capable of coding or managing abstract parameters. As Rogers [18] and Hassenzahl [35] emphasize, creative digital tools must balance computational control with cognitive transparency, enabling users to understand and manipulate generative processes through intuitive interaction. In other words, usability becomes a creative enabler rather than a secondary concern.

Overall, Generative Design demonstrates how computational logic can extend human creativity while also highlighting the need for accessible and interpretable design systems, a concern that becomes central when considering Shape Grammars.

2.5. Shape Grammars

Within the generative design landscape, Shape Grammars occupy a distinctive and foundational position. Originally conceived as a formal language for describing architectural styles and design languages [1], [44], they provide an interpretable and structured way of encoding design intent. While early Shape Grammar applications were primarily exploratory and descriptive, more recent implementations have integrated them into generative design workflows that incorporate parametric control, optimization, and user-defined constraints [4], [45]. In this sense, Shape Grammars integrate the formal rigor of rule-based systems with the goal-oriented, performative nature of modern generative design.

Shape Grammars are formal systems that generate visual compositions through the recursive application of geometric transformation and substitution rules. They represent to spatial design what linguistic grammars represent to written language: a means of constructing structured, meaningful expressions from a basic alphabet. While linguistic grammars operate on strings of symbols, Shape Grammars manipulate spatial configurations of shapes directly. The formalism itself was inspired by linguistic generative grammars proposed by Chomsky [46] and by the production systems of Post [47], translating syntactic notions into spatial terms.

Shape Grammars are defined by three core components [1]:

(1) an alphabet of shapes, which specifies the basic visual elements available for composition;
(2) a set of rules, formulated as condition-action pairs, which describe how shapes can be added, removed, or transformed based on spatial relationships;
(3) an initial shape, from which the design generation process begins.

In later formulations, Stiny [8] introduced a fourth component, labels, to encode semantic or structural information, allowing more explicit control over the derivation process. Rules are applied recursively: if the pre-condition of a rule matches part of the current composition, the corresponding action is executed, modifying the design. Shapes can be added, removed, substituted or transformed using geometric operations, often defined algebraically. Because multiple rules may be applicable at any given time and can be applied in different orders, Shape Grammars naturally support the generation of multiple, stylistically consistent design alternatives.

Over time, extensions to the original formalism have addressed limitations in expressiveness and control. Among the most significant was the development of Parametric Shape Grammars [45]. These introduced variable parameters and algebraic constraints into rule definitions, enabling dynamic manipulation of dimensions, proportions, and relationships among shapes. This broadens the design space and allows families of solutions to be generated from a single grammar. This evolution not only increases the diversity of possible designs, but also aligns Shape Grammars more closely with modern generative design practices, where user-defined goals and constraints play a central role. Such advancements have been essential in bringing Shape Grammars closer to the needs of architectural and industrial design, as demonstrated in Duarte's Discursive Grammar for mass housing [4], and in subsequent works by Stouffs and Krishnamurti [48]. One notable implementation of Parametric Shape Grammars principles is the base of this dissertation, Alternative Shaper, a model for automatic design generation that integrates rule-based shape transformations with procedural execution, allowing algorithmic control over which rules are applied, in what order, and with what parameters. Originally developed for the automatic generation of architectural floor plans, the Alternative Shaper demonstrates the practical feasibility of implementing fully procedural and parametric Shape Grammars. Although focused on architectural floor plans, its architecture is domain-agnostic and can be adapted to other design contexts. It provides features that make it an ideal foundation for further development: an extensible grammar engine capable of managing rule sequencing, conditional execution, and parameterized transformations. However, the Alternative Shaper primarily focuses on the computational back-end rather than on user interaction. Its powerful rule interpreter lacks an accessible interface that allows designers to visually author, inspect, and manipulate grammars.

Another central theoretical concept that shapes both the expressive power and computational complexity of Shape Grammars is emergence. Knight [49] formalized the idea that new shapes may appear through perceptual inference rather than explicit rule definition. Emergent shapes embody the creative potential of Shape Grammars: they reflect the designer's ability to "see" new configurations and reinterpret the grammar dynamically during the design process. However, this same flexibility introduces challenges for computational interpretation and automation, as emergent perception often depends on subjective human recognition rather than deterministic computation.

Alongside these theoretical developments, numerous applications have demonstrated the potential of Shape Grammars in capturing architectural styles, generating design variations, and formalizing creative processes. Early examples include grammars describing Palladian villas [2], Queen Anne houses [3], or Mughul gardens [8]. These case studies validated Shape Grammars as analytical and generative frameworks, capable of encoding stylistic languages and producing new yet coherent designs.

Later, computational implementations, such as Tapia's GEdit [50] and Heisserman's Genesis [51], attempted to operationalize these principles into software interpreters, paving the way for subsequent tools such as DESIGNA [10], which integrates Shape Grammar derivation within modern CAD environments.

Despite their expressive potential, most Shape Grammar tools continue to face practical limitations. The manual definition of rules, lack of immediate visual feedback, and absence of accessible interfaces make Shape Grammars difficult to use outside academic or expert contexts. Users who are not already familiar with the theoretical foundations of Shape Grammars often struggle to understand their underlying logic or to formulate valid rules, which makes it challenging to engage with these systems meaningfully. This steep learning curve, combined with the lack of intuitive, real-time interaction, discourages the kind of exploratory experimentation that characterizes creative workflows. As highlighted in HCI-driven approaches such as IM-sgi [11], effective Shape Grammar environments must prioritize usability, interactivity, and cognitive transparency. In this context, being HCI-driven means that the design of the system is guided by user experience principles that align interaction, feedback, and learning processes with human cognitive capabilities rather than with purely computational logic. These principles form the foundation of the system developed in this research.

2.5.1. Gaps and Opportunities of Existing Tools

The evolution of Shape Grammar tools reflects a persistent tension between formal expressiveness, computational tractability, and practical usability in design workflows. Beyond the foundational case studies in architecture, such as the previously mentioned Palladian villas [2], Queen Anne houses [3], and Mughal gardens [8], the literature reports a broad spectrum of applications that include product form languages (e.g., coffee makers [6], automotive brand identity [7], [52], consumer packaging [9]), furniture and decorative arts [5], and urban form generation (e.g., Marrakech Medina [53]). These cases confirm the dual role of Shape Grammar as analytical (capturing existing styles) and generative (generating novel designs) formalisms.

From an implementation standpoint, early and influential interpreters illustrate complementary strengths and recurring limitations. GEdit [50] provided an accessible 2D environment with maximal lines and subshape detection (supporting emergence), while Genesis [51] progressed 3D solid descriptions and sparked industrial interest, though many technical details are not public. Subsequent systems explored different representational choices (visual, symbolic, and set grammars) balancing subshape matching, parametric control, and computational cost [54], [55], [56], [57], [58]. Later efforts integrated computer vision techniques to improve subshape recognition and rule matching (Qi [59]), or hybridized Shape Grammars with graph grammars (e.g., GRAPE [60]) to enhance spatial recognition and parametric manipulation. However, these improvements often came at the cost of increased system complexity and reduced computational performance.

Industry-facing tools, notably procedural city modeling (e.g., CityEngine [61]), demonstrate scalability in rule-based pipelines for urban massing and facade generation; however, their grammars are typically text-based, domain-specific, and tuned for production rather than for didactic exploration of Shape Grammars concepts, leaving limited support for visual rule authoring, stepwise derivation, or emergent-shape reasoning in early design.

In the broader landscape of Shape Grammar implementations, which spans both academic prototypes and industry-oriented systems, DESIGNA [10] advances a software architecture that bridges an interpreter with standard CAD tools via Rosetta, enabling labeled shapes, rule description as transition operators, and search strategies (depth-/breadth-first) for rule application. This improves portability and output continuity with AutoCAD/Rhino, but still assumes a technically proficient user and prioritizes the back-end engine over visual, guided authoring and immediate feedback interfaces.

Complementary HCI-oriented proposals, such as IM-sgi [11], foreground usability through Cognitive Walkthrough and differentiated user profiles (e.g., students, designers, experts), articulating ergonomic criteria for Shape Grammar interfaces. Yet, these contributions remain largely conceptual or disconnected from fully operational derivation engines, limiting their validation in complex, dynamic design scenarios.

Finally, lightweight visual front-ends such as QShaper [12] illustrate the advantages of explicitly organizing Shape Grammar components into three distinct sets (shapes, rules, and resulting designs), allowing stepwise visualization of the derivation process. However, these tools do not scale well to parametric grammars, constraint handling, or emergent-shape recognition, nor have they been systematically evaluated with professional users.

Across these lines of work, four cross-cutting limitations persist:

(1) Rule authoring and learnability remain significant barriers. Most existing interpreters rely on text-based or code-centric definitions, offering limited syntactic guidance, semantic validation, or visual feedback. This raises the entry threshold for non-expert users and restricts Shape Grammars to technically proficient audiences. Visual authoring environments, when they exist, tend to be partial or experimental, limiting their pedagogical potential.

(2) Emergence and subshape support remains computationally demanding. Consequently, many tools either constrain the representation (e.g., rectilinear lines) or sidestep emergence entirely, reducing exploratory power.

(3) Although parametric and procedural extensions broaden the generative scope of Shape Grammars, existing tools rarely provide intuitive ways to express, visualize, or debug procedural behaviors such as rule sequencing, search strategies, or conditional execution. These aspects are typically confined to back-end interpreters rather than integrated into the design environment itself.

(4) Interaction and feedback remain limited. Most systems offer little visibility into what rules matched, why they applied, or what alternatives were available. The absence of such interpretability restricts co-creative exploration, comparison of derivation paths, and learning through iteration, which are key qualities for both educational and professional use.

Taken together, these limitations underline the opportunity for a new kind of Shape Grammar environment that combines formal rigor with usability and pedagogy. Such a tool would: emphasize 2D visual interaction, to keep recognition and authoring tractable while supporting subshape reasoning; procedure-aware execution, making rule sequences and derivations explicit within the interface; and a didactic interaction model grounded in good HCI practices, designed for progressive disclosure of complexity, inline feedback, and visual debugging of derivations.

This direction explicitly builds upon the Alternative Shaper, whose computational core demonstrates the feasibility of fully parametric and procedurally controlled Shape Grammar derivations. The present work extends this foundation by introducing SAGE, a visual interface layer that exposes the Alternative Shaper's capabilities through direct manipulation, stepwise feedback, and intuitive interaction The goal is not to reinvent its computational core, but to make it accessible: to transform a technically advanced interpreter into an exploratory and didactic environment that bridges formal Shape Grammar theory and creative practice.

CHAPTER 3

Tool Design and Development

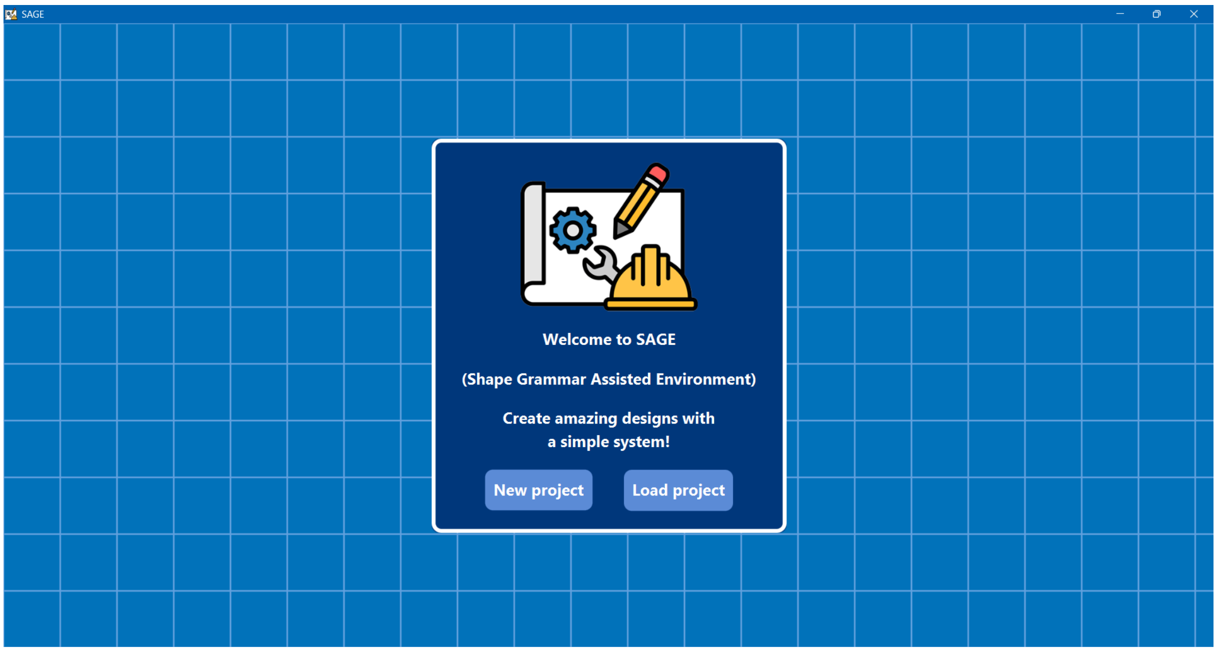
This chapter presents the design and development of SAGE (ShApe Grammar assisted Environment). It begins by outlining the structure and main features of the final system, followed by an overview of the iterative prototyping process through which its design evolved. The goal is to demonstrate how theoretical principles and iterative user feedback were progressively translated into a coherent and usable Shape Grammar environment.

3.1. Overview of SAGE

SAGE integrates three main functionalities: the creation of shapes, the definition of rules and procedural grammars, and the guided generation of designs. The system is organized into three core screens:

(1) the Home screen;
(2) the Main workspace;
(3) and the Build screen.

Each screen supports a specific stage of the user workflow. Upon starting the application, users begin at the Home screen, where they can initiate a new project or resume work on an existing one (Fig. 3.1).



Welcome to SAGE

(Shape Grammar Assisted Environment)

Create amazing designs with a simple system!

FIGURE 3.1. Snapshot of the Home screen in SAGE.

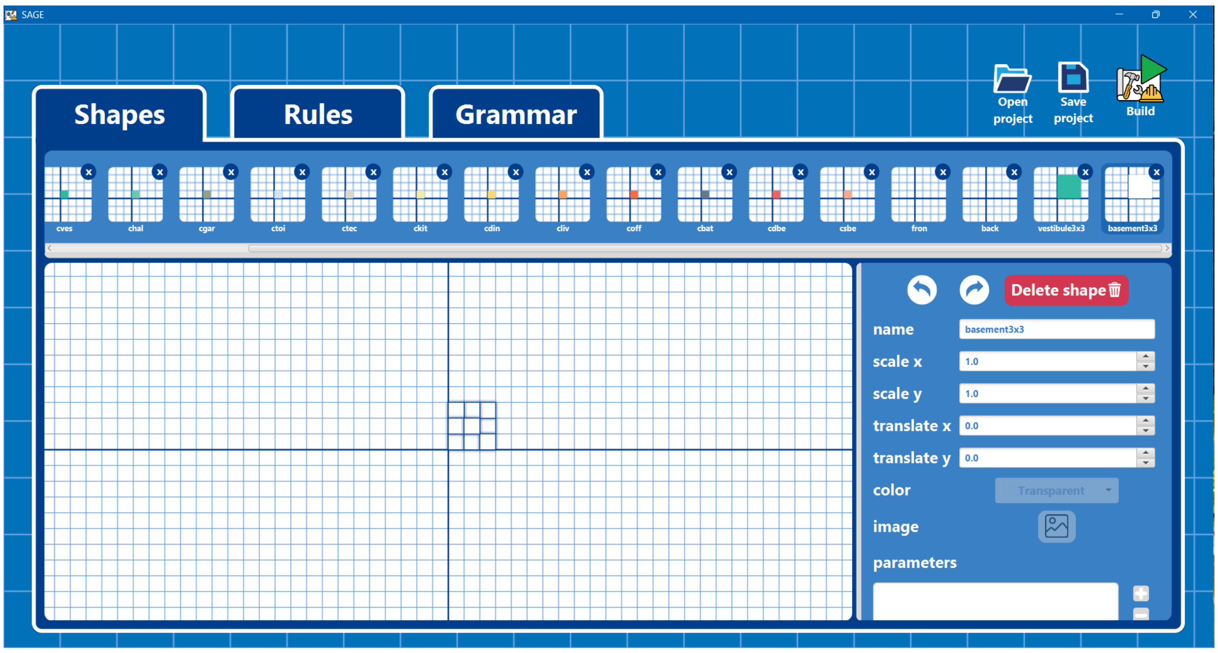
The Main workspace includes three core editors:

(1) the Shapes Editor (Fig. 3.2);
(2) the Rules Editor (Fig. 3.3);
(3) and the Grammar Editor (Fig. 3.4).

This workspace also provides project-level functionality, including saving, loading, and accessing the design generation interface.

In the Shapes Editor (see Fig. 3.2), users can create and edit either Simple or Composition shapes on a grid-based canvas. When adding a new shape, the user must specify whether it will be a Simple shape, representing a single editable entity, or a Composition shape, which acts as a container for multiple shapes via pop-up.

For Simple shapes, only one element can exist on the canvas at a time, and it can be selected and edited through the properties panel. In Composition shapes, multiple elements can coexist, allowing users to add other shapes by dragging their thumbnails from the shape list onto the canvas. Once added, each constituent shape can be selected and edited individually. When no specific shape is selected, the tool assumes interaction with the entire Composition, enabling global transformations such as scale (scaleX, scaleY) and translation (translateX, translateY) that affect all constituent shapes. Additionally, Composition shapes include a parameter list that can be used for further customization or integration with procedural grammars. For Simple shapes, editable properties include name, position (x, y), size (width, height), color, associated image, and parameter list. The properties panel supports standard interactions, including undo, redo, and delete. A thumbnail list at the top of the editor allows users to quickly view, select, and manage



name: basement3x3
scale x: 1.0
scale y: 1.0
translate x: 0.0
translate y: 0.0
color: Transparent
image
parameters

FIGURE 3.2. Snapshot of the Main workspace with Shape Editor selected in SAGE.

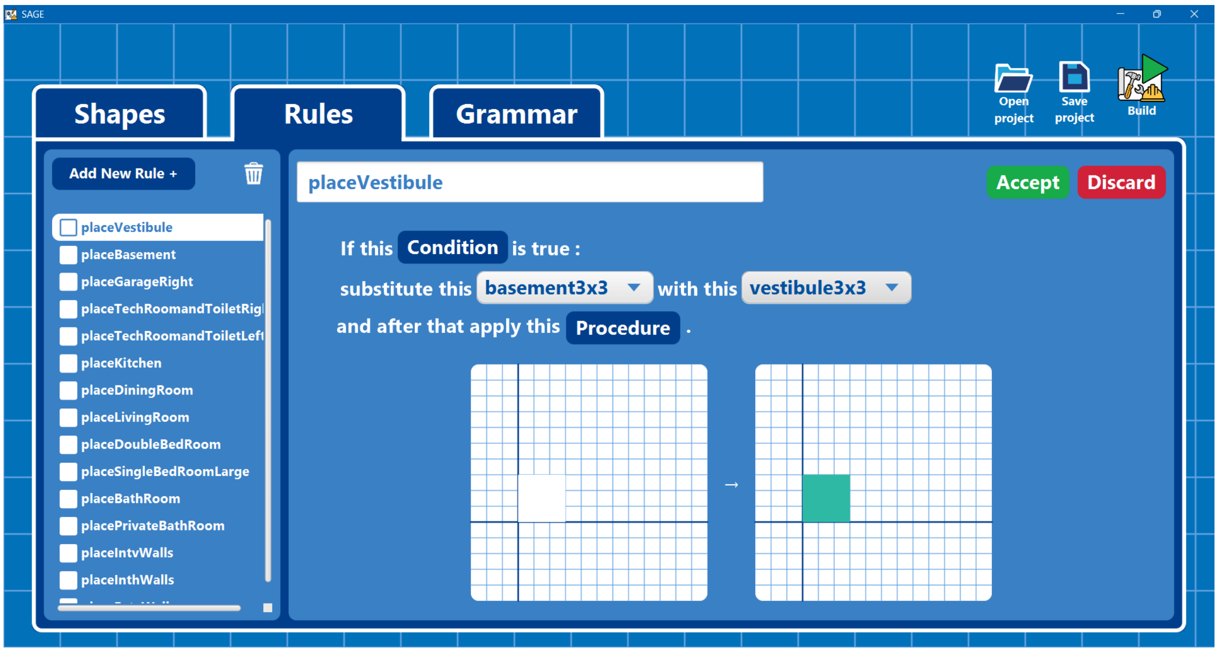


FIGURE 3.3. Snapshot of the Main workspace with Rule Editor selected in SAGE.

shapes for editing, including the option to delete them directly from the list, ensuring flexibility and efficient workspace organization.

The Rules Editor (Fig. 3.3) provides an overview of all transformation rules defined by the user. Each rule specifies an initial ("before") and resulting ("after") shape, selected via drop-down menus listing all previously created shapes. The editor extends the traditional Shape Grammar model by allowing the optional addition of conditions and procedures, offering finer control over rule application. These are defined through dedicated buttons that open Blockly-based visual editors, where users can create logic blocks representing constraints or post-rule actions. This dynamic mirrors the logic of the Alternative Shaper [14], where each rule can include associated conditions and procedural steps to define its behavior more precisely. In SAGE, this hybrid approach maintains the simplicity of rule-based transformations while enabling more expressive and modular design behaviors, allowing users to embed local logic directly within individual rules when needed.

The Grammar Editor (Fig. 3.4) integrates a Blockly-based programming environment where users can define high-level procedures that orchestrate the application of rules and shapes. Within this environment, users may also create new shapes and rules as part of procedural workflows, extending the expressive potential of the grammar. However, in the current implementation, this functionality is not yet fully synchronized with the main editors, meaning that elements created directly in Blockly are not automatically reflected in the shape and rule lists of the visual workspace. Despite this limitation, the Grammar Editor provides an accessible and powerful way to model generative logic without the need for textual programming, bridging procedural reasoning and visual authoring.

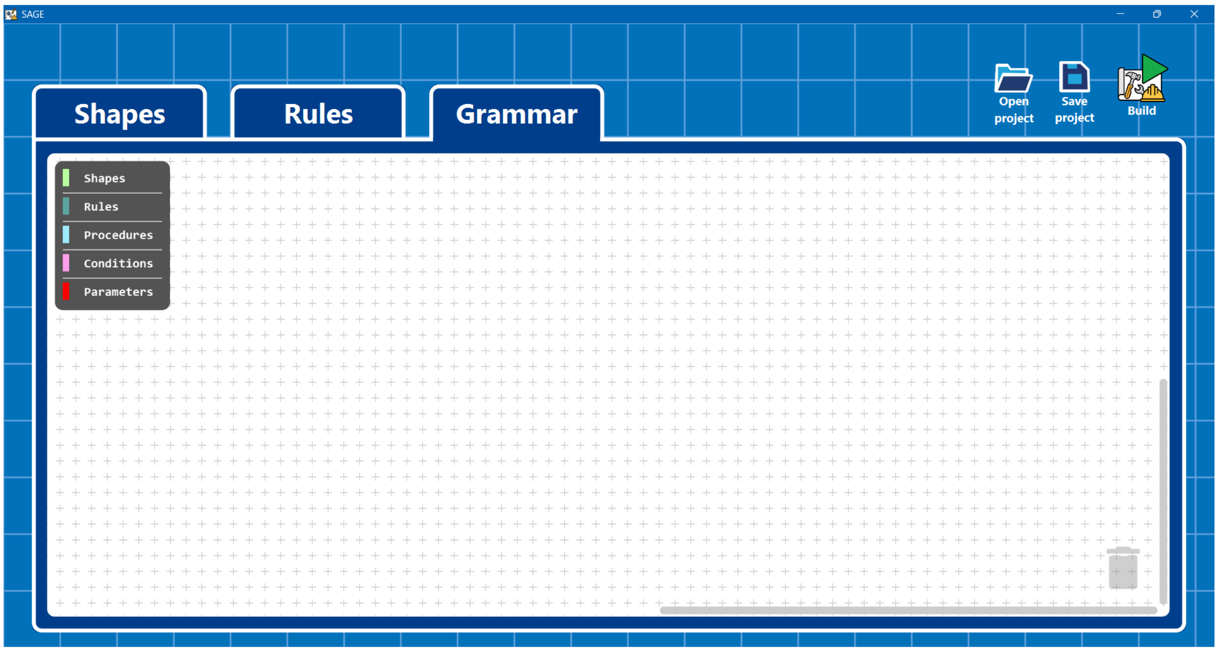


FIGURE 3.4. Snapshot of the Main workspace, with Grammar Editor selected in SAGE.

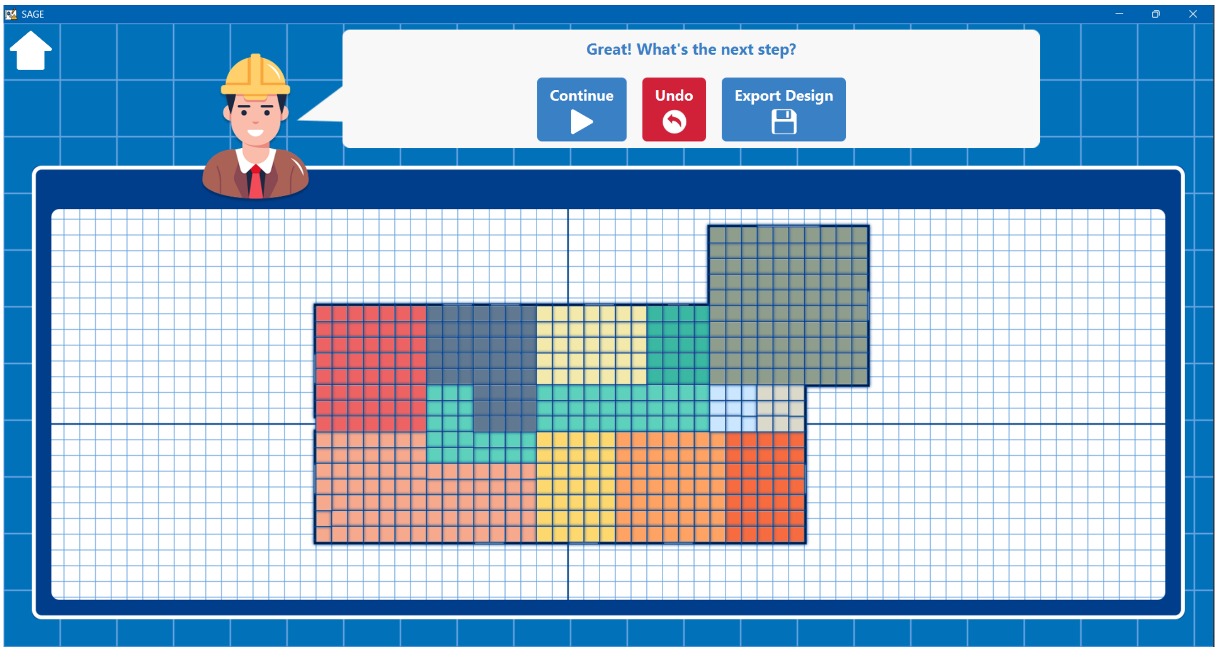


FIGURE 3.5. Snapshot of the Build screen in SAGE, showing the design of a complete home generated by Alternative Shaper.

When users are ready to generate designs, they move to the Build screen (Fig. 3.5), where the process is presented as an interactive dialogue with a virtual builder mascot. This mascot acts as a guide through the generation process, helping configure parameters and presenting design alternatives in a conversational format. The underlying generation logic is powered by the Alternative Shaper's Prolog engine, which executes the grammar rules and procedures defined in that model. To initiate a design, users must select a

Reference	Feature	Implementation in SAGE
Unity [62]	Property inspector panels	Adopted for the shape Properties panel to ensure clarity and consistency in parameter editing.
GameMaker [63]	Thumbnail-based object selection	Used in the shape list to allow quick visual access and editing of existing shapes.
Blender [64]	Dynamic "+" and "–" parameter controls	Implemented for adding and removing parameters within shapes and rules.
Scratch[65]/ Blockly [15]	Block-based visual programming	Used in the Grammar Editor to enable logic definition without textual coding.
Architectural Blueprints	Visual metaphor/ aesthetic	Inspired the blue-toned palette and grid background, evoking the look of a design workspace.

Table 3.1. Interface inspirations and their influence on SAGE's design.

procedure, if multiple outcomes are possible, the interface allows them to be iterated between them. This loop of procedure selection continues until the user is satisfied with the result. The final design can then be exported for later use.

With all these concepts in place, SAGE is conceptually structured into two complementary environments, one dedicated to grammar creation, where users define shapes, rules, and procedures, and another focused on design generation using that grammar. This separation allows users to move from structural definition to creative exploration in a clear and accessible way. Fig. 3.6 illustrates this workflow, highlighting the key user interactions in each phase of the process.

From a visual and interaction design perspective, SAGE incorporates several interface patterns and conceptual references drawn from both established design tools and broader visual metaphors. This strategy aimed to leverage familiar paradigms (whether from software environments or design practice) to improve usability, evoke domain familiarity, and reduce cognitive load for first-time users. Table 3.1 summarizes the main references and the specific interface aspects adapted from each.

3.2. Prototyping and Iterative Design Process

While the previous section presented the final structure and visual identity of SAGE, the next sections retraces the iterative design process through which the tool's interface and functionality were refined. The prototyping phase played a central role in translating theoretical concepts into practical interaction models, ensuring that the resulting system aligned with both Shape Grammar principles and user experience requirements.

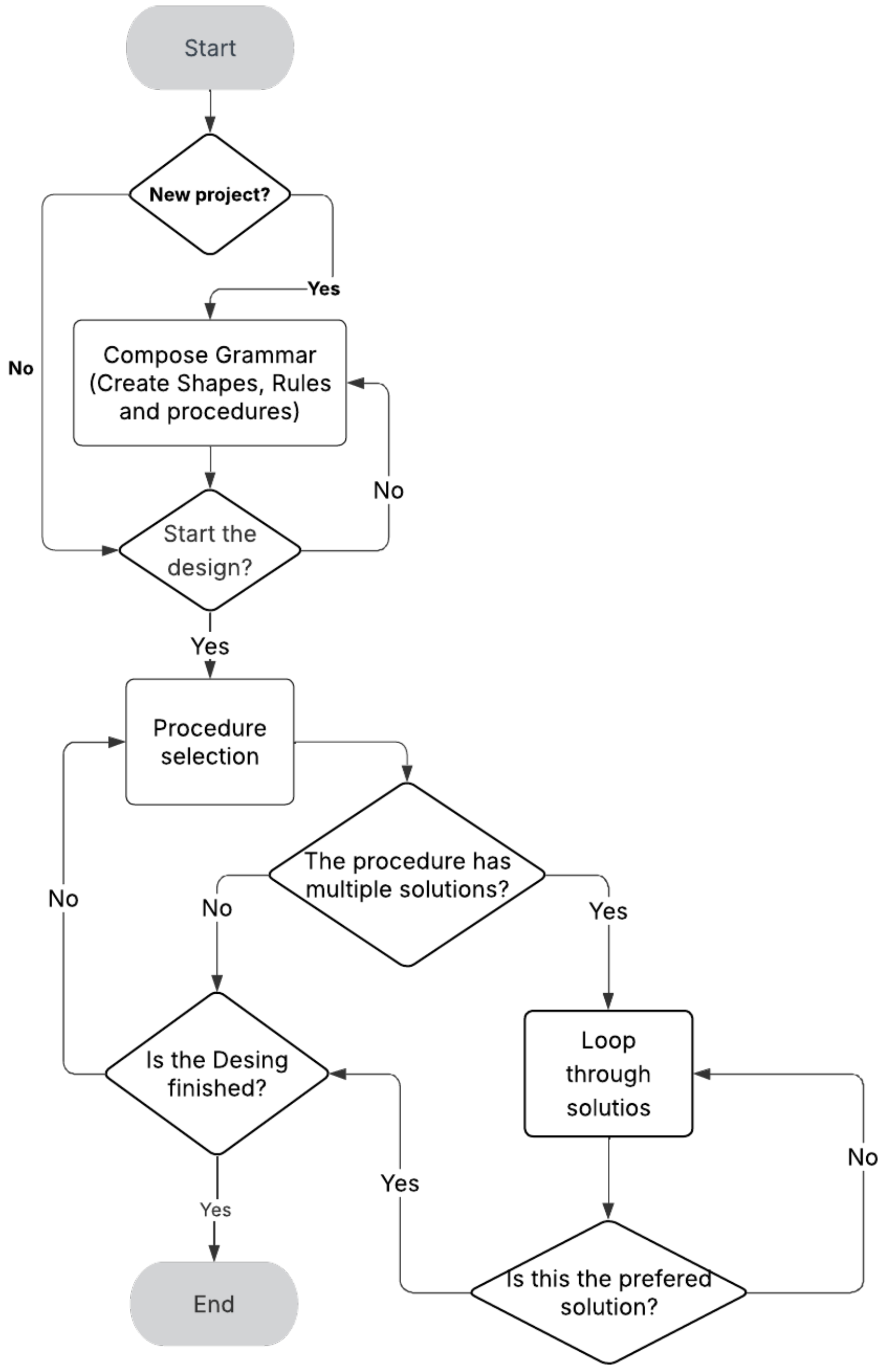


Figure 3.6. User workflow across editing and build phases. Diamonds represent user decisions and rectangles denote screen states.

In line with a User-Centred Design (UCD) approach, the design of SAGE progressed through multiple stages of medium-fidelity prototyping. Each iteration aimed to test specific interaction hypotheses, validate usability decisions, and progressively refine the interface based on user feedback. Three main prototypes were developed using Figma [66], each followed by evaluation sessions combining task-based observation and qualitative feedback.

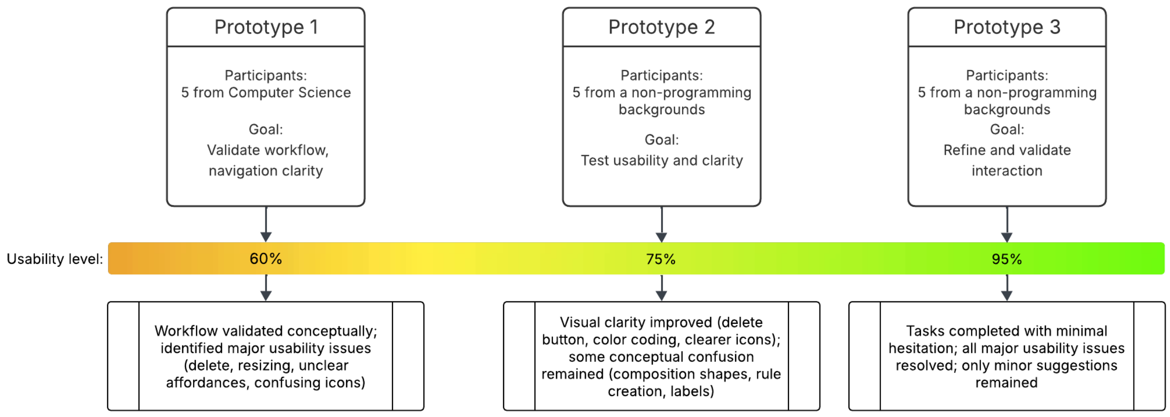

Prototype 1
Participants:
5 from Computer Science
Goal:
Validate workflow, navigation clarity
Prototype 2
Participants:
5 from a non-programming backgrounds
Goal:
Test usability and clarity
Prototype 3
Participants:
5 from a non-programming backgrounds
Goal:
Refine and validate interaction
Usability level:
60%
75%
95%
Workflow validated conceptually; identified major usability issues (delete, resizing, unclear affordances, confusing icons)
Visual clarity improved (delete button, color coding, clearer icons); some conceptual confusion remained (composition shapes, rule creation, labels)
Tasks completed with minimal hesitation; all major usability issues resolved; only minor suggestions remained

FIGURE 3.7. Summary of the three Figma prototype iterations, showing participant profiles, evaluation goals, and the progressive increase in usability across the design refinement process.

Figure 3.7 provides an overview of the three iterations, summarizing their goals, participant profiles, and the gradual improvements in usability and interaction design achieved throughout the process.

The prototyping process followed early-stage evaluation methodologies [67], emphasizing lightweight and iterative testing to identify usability issues efficiently and with minimal development overhead. Following principles of HCI [68], [69], each prototype iteration sought to minimize the gap between users' mental models and the intended system logic. Think-aloud protocols [70] were employed during all sessions to capture participants' reasoning, difficulties, and interpretation of visual cues in real time.

Prototypes were tested through a predefined task set designed to simulate a complete Shape Grammar workflow, from shape creation to design generation. The same task set was reused across all iterations, with only minor adjustments, to ensure comparability of results and to systematically assess the impact of each refinement:

(1) Create two simple shapes ("quarto" and "wc") and modify their size, name, and color;
(2) Create a composition shape ("apartamento") using the previously created simple shapes;
(3) Add and edit a parameter ("door") in a shape;
(4) Delete a shape;
(5) Define a basic substitution rule ("Random Rule");
(6) Delete an existing rule;
(7) Initiate a new design;
(8) Apply a procedure capable of generating multiple design solutions and review the generated alternatives.

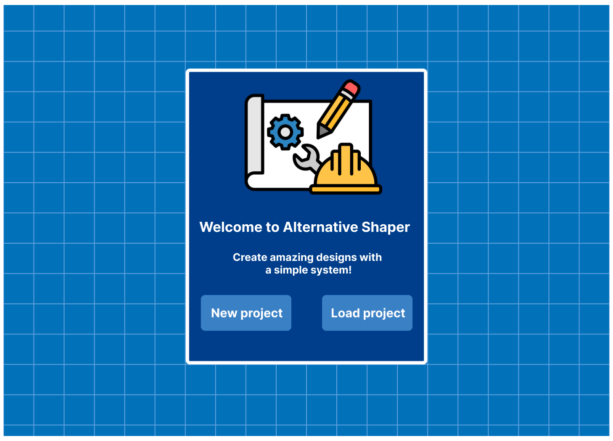

Welcome to Alternative Shaper
Create amazing designs with a simple system!
New project
Load project

FIGURE 3.8. Initial screen of the first SAGE prototype.

Because Figma prototypes cannot execute procedural logic, the Blockly-based programming layer was not directly testable at this stage. Participants were therefore instructed to assume that the necessary procedures were already defined. This limitation did not compromise the evaluation goals, which focused primarily on interface clarity, interaction flow, and conceptual understanding of the workflow.

The following sections describe each prototype iteration in detail, outlining their main design changes, evaluation procedures, and the insights that guided the progressive refinement of SAGE's interface.

3.3. First Prototype

The first prototype[1] aimed to validate the proposed workflow, from shape creation to design generation, and to assess the clarity of navigation and visual hierarchy prior to implementation. This evaluation round involved five participants with backgrounds in computer science, whose familiarity with programming environments, game engines, and platforms such as Scratch (conceptually similar to Blockly) ensured that the study could focus primarily on interface design rather than functional comprehension. Although the procedural layer was not directly testable in Figma, participants were able to infer how Blockly's logic would operate based on its visual organization and category structure.

This iteration marked the first definition and testing of SAGE's conceptual workflow. The interface was organized into sequential screens that reflected the intended design logic: a New/Load Project screen (Fig. 3.8) for project management, a main workspace divided into three editors (Shapes, Rules, and Grammar) and a final design generation screen. At this stage, the prototype still displayed the placeholder title "Welcome to Alternative Shaper" on the home screen, as the project name had not yet been finalized. This organization established the foundation for the navigation model adopted in subsequent iterations.

Within the Shapes Editor, this prototype introduced the main interaction paradigms that would later define SAGE's visual and functional identity. Core principles such as

[1]Interactive Figma prototype of the first iteration.

thumbnail-based shape organization, property-based parameter editing, and the use of "+" and "–" buttons for parameter management were established here, drawing on conventions from established design tools to promote familiarity and reduce cognitive load. When a new shape was created, a default 1 × 1 placeholder was automatically instantiated, as shown in Figure 3.9, requiring users to delete it before composing more complex configurations. Deleting the placeholder required selecting the shape on the canvas, which opened a contextual pop-up offering the option "Delete Shape" for confirmation.

In the Rules Editor, the same "+" and "–" buttons were used to manage rules, ensuring consistency across the system. During this phase, its design focused solely on representing a transformation between a "before" and an "after" shape, both selectable from dropdown menus (Fig. 3.10). No guiding textual elements were yet included, as this explanatory layer would only be introduced in later iterations.

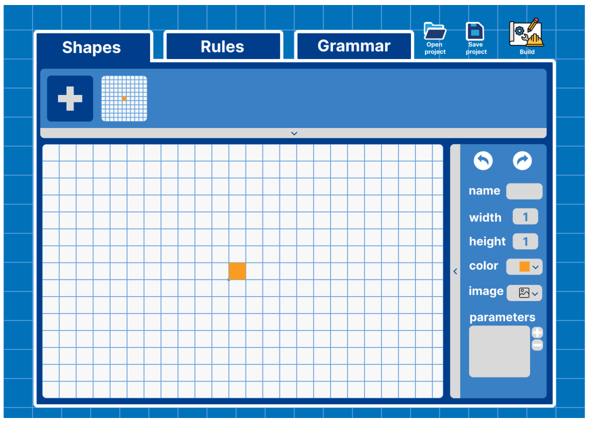


FIGURE 3.9. Shapes editor screen with a default shape added in the first SAGE prototype.

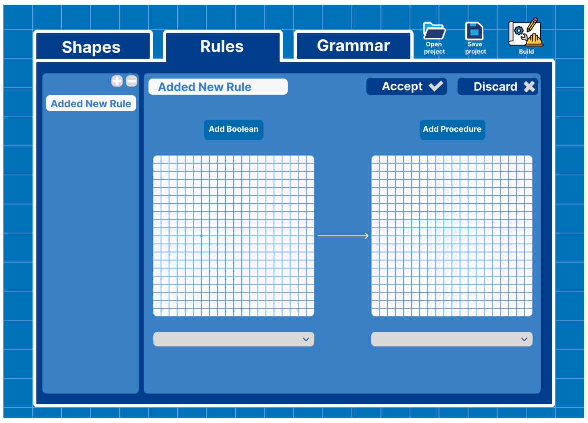


FIGURE 3.10. Rules editor screen with a new Rule in the first SAGE prototype.

Regarding the Grammar Editor, this prototype included a static image of the pre-existing Blockly module from the Alternative Shaper project , used purely as a conceptual placeholder (Fig. 3.11).

The conversational workflow between user and mascot to generate designs was first established in this prototype. The dialogue began with a prompt to either start a new design or load a previous one (Fig. 3.12). Once a new design was initiated, the system entered the iterative loop of procedure selection (previously illustrated in Fig. 3.6), where users selected both a shape and a procedure (Fig 3.13), a structure inspired by the Alternative Shaper. In subsequent iterations, explicit shape selection was removed: the "selected shape" is implicitly the current state of the design (i.e., the shape composed so far in the Build workspace), and procedures operate relative to that evolving state.

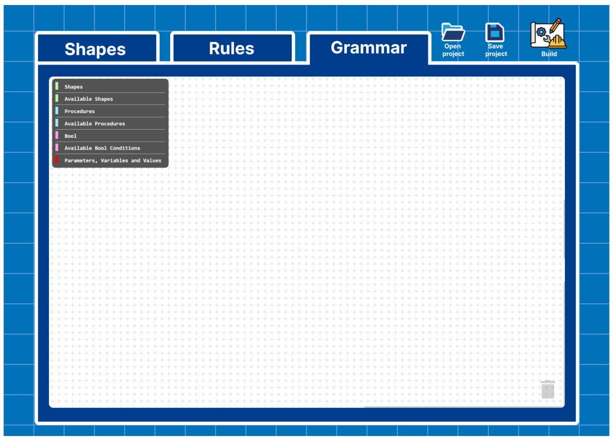


FIGURE 3.11. Grammar editor (Blockly) in the first SAGE prototype.

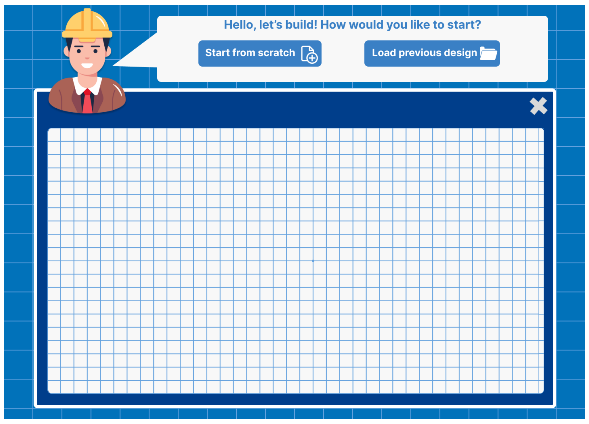


FIGURE 3.12. Initial screen of the design generation phase in the first SAGE prototype.

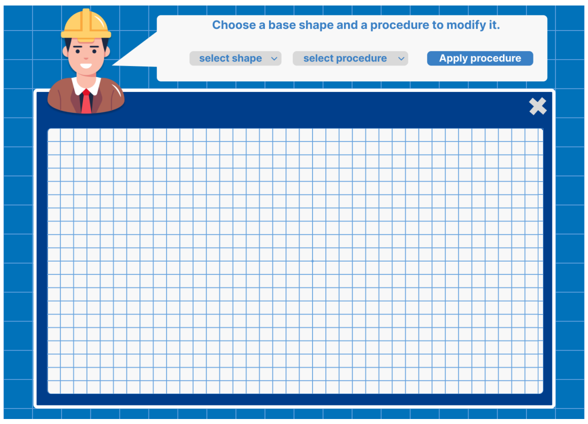


FIGURE 3.13. Procedure and Shape selection in the first SAGE prototype.

After the first procedure was selected, the mascot prompted users to confirm or adjust the origin point of the design, ensuring spatial coherence for subsequent transformations (Fig. 3.14). Each time a procedure was executed, the mascot presented the resulting solution on screen. When a procedure generated multiple outcomes, users could browse through the alternatives until identifying a preferred one (Fig. 3.15) . Upon accepting a result, or discarding the current procedure, the system asked whether to save the generated outcome as a new shape. If confirmed, this resulting composition shape became available in the Shape Editor for future use in rules or procedures (Fig. 3.16).

After each confirmation of solution, the dialogue returned to the main loop, prompting the user to choose whether to apply another procedure, discard the last step, or save and finish the design (Fig. 3.17). This cyclical process of shape and procedure selection continued until the user selected the "Save the build" option, which finalized the design and triggered the mascot to confirm its storage. Navigation back to the editing environment was available at any time through an “X” button located in the upper-right corner of the canvas, a temporary mechanism later refined for greater clarity and consistency.

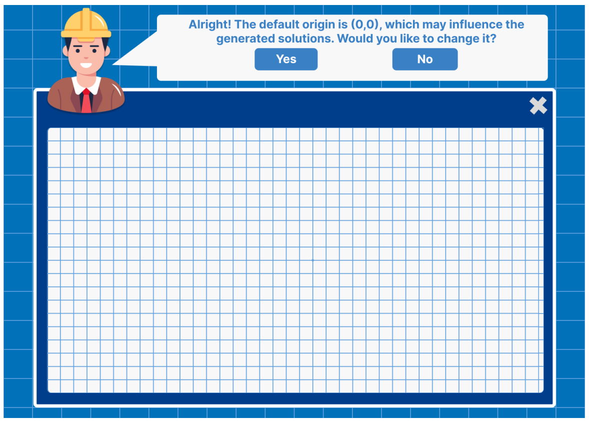

Alright! The default origin is (0,0), which may influence the generated solutions. Would you like to change it?

FIGURE 3.14. Mascot prompting origin point confirmation in the first SAGE prototype.

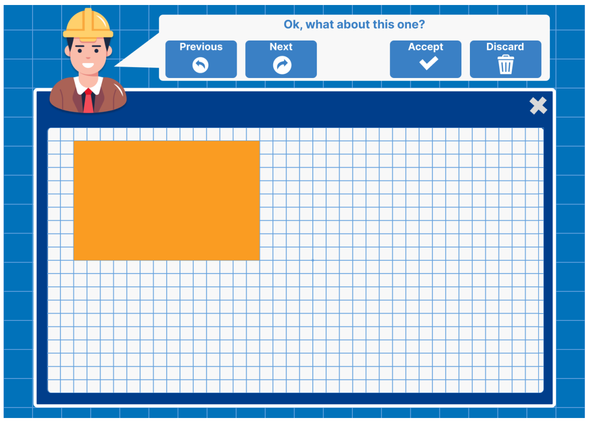

Ok, what about this one?

FIGURE 3.15. Design alternative generated after executing a procedure in the first SAGE prototype.

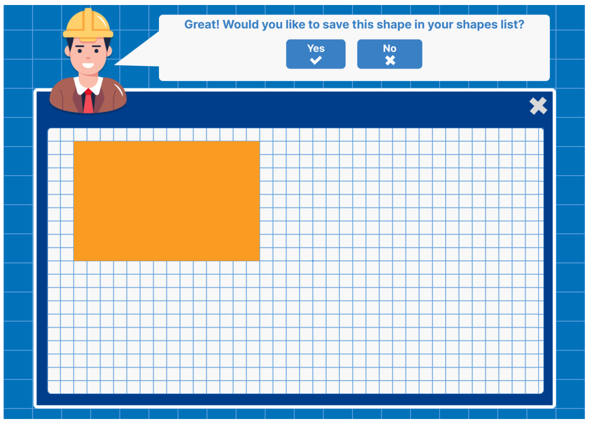

Great! Would you like to save this shape in your shapes list?

FIGURE 3.16. Prompt asking the user whether to save the accepted solution as a new shape in the Shape Editor in the first SAGE prototype.

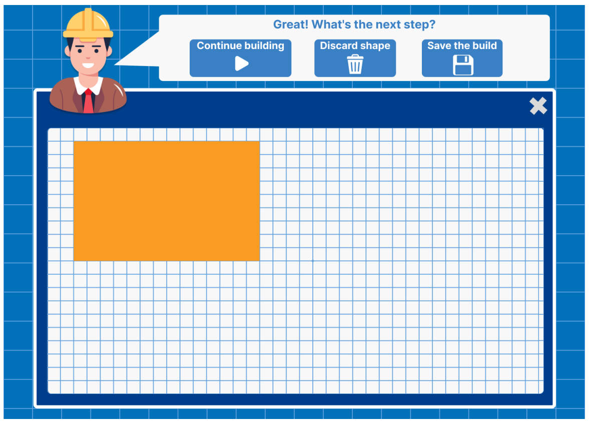

Great! What's the next step?

FIGURE 3.17. Dialogue stage prompting the user to choose the next action.

Additional snapshots from this prototype, including the shapes and rules created by participants as well as interface states from the shape editing and design generation phases, are provided in Appendix A.

Besides the tasks previously mentioned, participants were prompted to provide feedback on visual and interaction design elements.This included: the evaluation of different icon options for the Let's Build button; the use of color to indicate actions (e.g., accept, discard, add, delete), with green and red tones discussed only if the participant considered color appropriate in that context; and whether the origin indicator was sufficiently visible or required additional cues such as axes or color differentiation. These alternative design options are presented in Appendix B.

3.3.1. Feedback and Insights

Participants provided feedback on various aspects of the interface, interaction, and visual design. In the Shape Editor, participants suggested the addition of a full-screen mode to focus solely on the grid. The Delete Shape action was also reported as insufficiently visible; users recommended integrating it directly into the shape properties panel for greater accessibility. Adjustments to parameter controls were proposed as well, for instance, enabling parameter creation by clicking empty areas and enabling their removal through direct interaction rather than relying on the "+" and "–" buttons. Some participants also experienced difficulties when creating composition shapes, as they did not initially realize that the default shape needed to be deleted before constructing the composition. Furthermore, resizing shapes by dragging was not immediately intuitive, prompting suggestions to include visible handles or vertex indicators.

In the Rule Editor, participants similarly requested the addition of a dedicated Delete Rule button within the editor itself. The existing mechanism, based on the "+" and "–" buttons at the top of the rules list, was perceived as not intuitive and inconsistent with standard interaction patterns.

Regarding the Blockly interface, several users expressed confusion about the organization of blocks, category naming, and color coding. They suggested that shape availability and labeling be made clearer to support easier navigation and block identification.

Finally, in the design generation stage, the Let's Build button was frequently misinterpreted as a static logo. Participants recommended either changing the icon or adding a visual outline to clarify that it functions as an interactive element. Among the proposed alternatives, the most preferred was an icon resembling the application's logo but incorporating a play symbol, which participants considered both intuitive and visually consistent with the system's identity. Similarly, the "X" used to exit the Let's Build screen was often misunderstood; users proposed replacing it with a back arrow positioned in the top-left corner to improve navigation clarity. The use of green and red tones to represent confirmation and cancellation actions was positively received overall, although some participants cautioned that color should only be used when semantically appropriate. Additionally, the visibility of the origin indicator was questioned, with suggestions to make it more prominent, possibly through axes or stronger color differentiation.

These observations provided essential guidance for the refinement of interaction patterns and the restructuring of several interface components, which were addressed in the development of the second prototype.

3.4. Second Prototype

Based on the feedback gathered from the first iteration, several interface and interaction improvements were implemented in the second prototype[2].

In the Shape Editor, the Delete Shape action was made more prominent through the addition of a dedicated button within the properties panel, improving both visibility and accessibility. This design solution was well-received and was therefore retained in the final version of SAGE.

Additional interface snapshots illustrating this new deletion flow and the shapes created by participants during this iteration are presented in Appendix C.

Additionally, visual handles were added at each vertex of a shape, following the approach used in Microsoft Word [71], to indicate that the vertices can be dragged, making the resizing functionality more intuitive and immediately discoverable. The visibility of the origin was also enhanced by adding axes and clearer reference markers (Figure 3.18), a feature that remained unchanged in the final application due to its effectiveness in spatial orientation.

In the Rule Editor, the visibility and usability of the Add Condition and Add Procedure buttons were enhanced, and a trash icon was added to allow rule deletion. This also enabled observation of user preference between the new trash icon and the existing "–" button for removing rules (Figure 3.19).

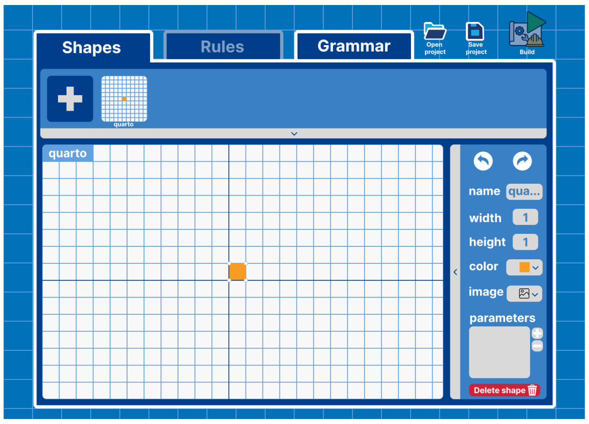


FIGURE 3.18. Shape editor showing the "quarto" shape being resized in the second SAGE prototype.

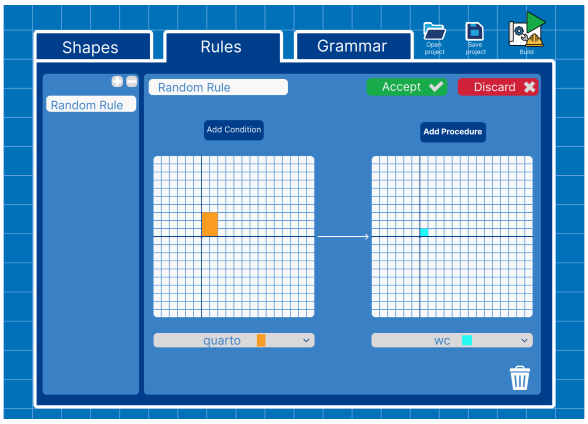


FIGURE 3.19. Rule editor showing the "Random Rule" selected in the second SAGE prototype.

Furthermore, the Blockly interface underwent a structural revision in this iteration, including clearer tab labels, improved legends, and revised block colors to facilitate block identification and categorization (Fig. 3.20).

[2]Interactive Figma prototype of the second iteration.

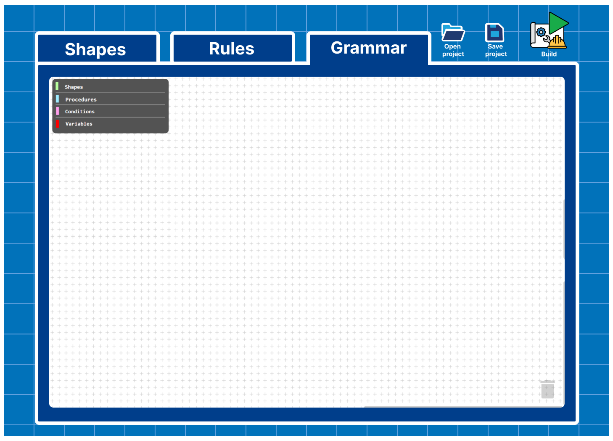


FIGURE 3.20. Grammar editor (Blockly) in the second SAGE prototype.

In the Design Generation screen, the Let's Build icon was redesigned to clearly indicate its function as a clickable button, following user feedback that suggested incorporating a play symbol to enhance affordance and recognizability. Similarly, the previous "X" used to exit the design generation phase was replaced with a back arrow, positioned in the top-left corner, signaling a return to the editing environment more intuitively (Fig. 3.21).

At a broader interface level, color coding was also introduced (green for confirmation and red for cancellation) to reinforce semantic meaning and maintain visual consistency across the system. This color strategy was validated in this iteration and preserved in the final implementation of SAGE. Furthermore, a dynamic feedback mechanism was added to guide users whenever certain functionalities were temporarily unavailable. For instance, when launching the application with an empty project, the Grammar tab remained disabled until at least one shape had been created, ensuring that users could only define procedures once the necessary elements were in place. Likewise, the Rules Editor became active only after at least two shapes existed, reflecting the logical dependency of rules on a "before" and "after" pair. The Let's Build button was similarly disabled until

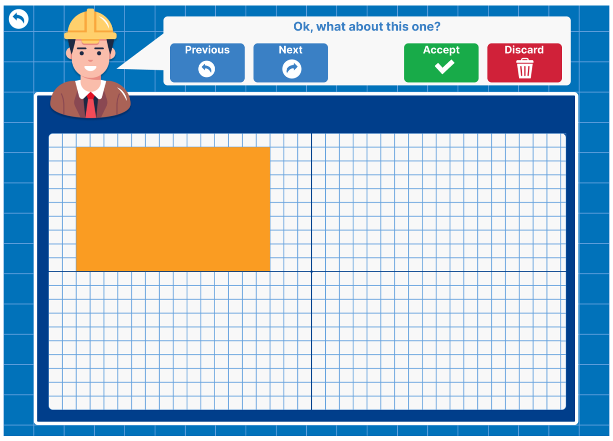


FIGURE 3.21. Screen showing the selection of a solution for the procedure "placeVestibule" in the design generation phase of the second SAGE prototype.

at least one procedure had been defined, reinforcing the sequential logic of the workflow. This incremental activation of interface elements helped prevent errors and clarified the expected workflow for first-time users.

This evaluation round involved participants without a computer science background (e.g., from the social sciences and psychology), allowing assessment of whether non-expert users could understand and interact with the tool without prior knowledge of Shape Grammars or visual programming concepts. The same task set and testing conditions as in the first iteration were maintained, enabling direct comparison of usability findings between iterations.

3.4.1. Feedback and Insights

During the evaluation sessions, several participants experienced difficulties creating composition shapes correctly. Many attempted to do so through the Rules Editor, while others failed to follow the required multi-step process of creating a new shape, deleting the default one, and then dragging shapes together. Additionally, the buttons and visual handles for resizing and moving shapes were not sufficiently prominent, causing some participants to overlook these functionalities entirely.

The workflow for adding conditions and procedures to Rules was also unclear. Several participants perceived the corresponding buttons as section headers rather than actionable elements. Likewise, the "–" button used for deletions was frequently ignored, suggesting that this approach was ineffective and that a dedicated trash icon was the more intuitive option.

Some participants continued to misinterpret the Let's Build entry point as either a logo or a settings menu, further supporting the suggestion to remove the gear icon previously associated with it.

In the Design Generation process, certain labels and prompts were also found to be confusing. For example, participants found the phrasing "Start from scratch" and "Load" ambiguous, suggesting that "New design" and "Load design" would be clearer alternatives. Similarly, the question regarding the design's origin caused misinterpretation: when asked "Would you like to change it?", participants associated "Yes" with maintaining the current option and "No" with changing it. In the solution acceptance stage, the options "Continue building," "Discard," and "Save the build" led to confusion. Participants assumed that "Discard" would delete the entire design rather than just the previous step, suggesting that a more explicit Undo option should be provided. Likewise, the label "Save the build" was interpreted as saving the project within the application rather than exporting it as a separate file.

Finally, issues also emerged in the last screen of the process, where the message "Your design is now saved! Feel free to explore or start a new one whenever you're ready." appeared alongside a button labeled "Start new one." Because some participants tended not to read the accompanying text, they misunderstood this option as restarting the entire project (including the definition of shapes, rules, and procedures) rather than simply beginning a new design instance.

3.5. Third Prototype

The second iteration confirmed the need for continued testing, particularly due to the persistent difficulties participants experienced when creating composition shapes and understanding the rule creation process. A new prototype[3] was therefore developed to address these issues and refine the interaction flow across the main areas of the system.

In the Shape Editor, the process of creating composition shapes was redesigned to be more explicit and guided. A dedicated pop-up dialog with the options of "Add Simple Shape" or “Add Composition Shape” was introduced to structure this process, ensuring that users could clearly distinguish between creating Simple and Composite entities. Additionally, the parameters "x" and "y", which are essential for defining a shape’s position, had been overlooked in previous iterations and were introduced in this version. Visual feedback mechanisms were also improved: selected shapes were now highlighted in dark blue, while vertex handles were enlarged and rendered in the same color to clearly indicate that they could be dragged for resizing. Moreover, the parameter system for composition shapes was refined to better reflect their collective nature. Rather than using the individual parameters "width", "height", "x", and "y" applied to Simple shapes, Composition shapes now include "scaleX", "scaleY", "translateX", and "translateY", enabling transformations that affect all constituent shapes simultaneously. Color and image options were intentionally disabled for Composition shapes, as these attributes are not meaningful at the composite level and would conflict with the visual properties of the individual components. Figure 3.22 illustrates these refinements, showing the Shape Editor with a composite shape "apartamento" selected.

All these changes proved effective during evaluation and were incorporated without major alteration into the final version of SAGE, defining the interaction model currently used in the implemented application.

In the Rules Editor, the interface was streamlined to reduce visual clutter and improve clarity. The generic “+” button was replaced with a clearly labeled “Add Rule” button, while the “–” button was substituted with a trash icon that also supported multiple selection for bulk deletion. When hovering over a rule in the list, a checkbox appeared, allowing users to select one or multiple rules before clicking the trash icon to remove them simultaneously. This approach was retained in the final version of SAGE, as it offered a clearer and more efficient interaction pattern for rule management. The Accept, Discard, and Rule Name elements were retained, but the rest of the editor was restructured into a guided, text-based format to help users understand the underlying logic. Up until that point, the Rule Editor, as shown in Figure 3.10 or Figure 3.19, presented only two grids representing the before and after states, with the condition and procedure fields placed above each grid, respectively. The new format, as illustrated in Figure 3.23, presented

[3]Interactive Figma prototype of the third iteration.

the structure “Substitute this shape with this shape if condition is true. Then, apply this procedure.” Dropdown menus allowed users to select the before and after shapes, while dedicated buttons provided access to Blockly for defining conditions and procedures. Smaller reference grids remained visible below to support visual comprehension. This textual, step-by-step structure effectively clarified the logic of rule creation and was maintained, with only minor stylistic refinements, in the final implementation of SAGE.

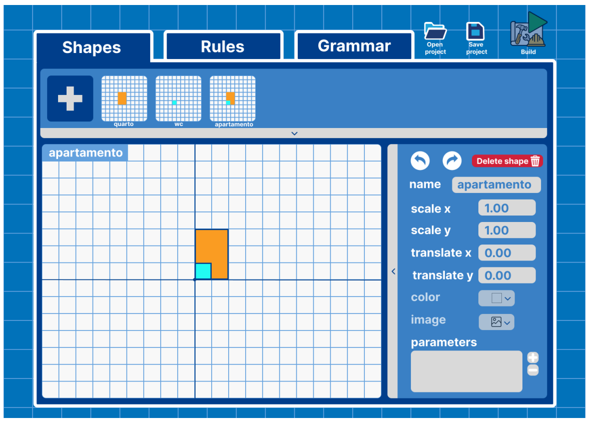
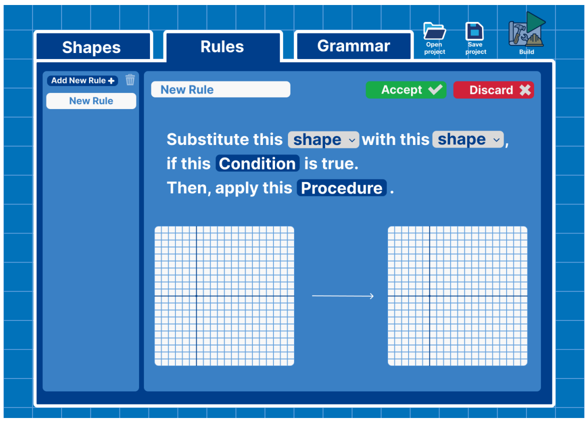


FIGURE 3.22. Shape editor screen with the “apartamento” shape selected in the third SAGE prototype.



Substitute this shape with this shape,
if this Condition is true.
Then, apply this Procedure .

FIGURE 3.23. Rules editor screen with a new Rule in the third SAGE prototype.

In the Build area, terminology and feedback messages were revised to eliminate ambiguities identified in previous sessions. The options “Start from Scratch” and “Load Previous Design” were replaced by “New Design” and “Load Design,” respectively, making their purpose clearer. Similarly, the Yes/No prompts used when changing a design’s origin were rephrased as “Change” and “Don’t Change” to prevent misinterpretation. After accepting a generated configuration, users were now presented with the options “Continue,” “Undo,” and “Export Design,” replacing the previous “Discard” button and its trash icon with a more intuitive "Undo" function, and substituting “Save Design” with “Export Design” for greater conceptual accuracy.

A confirmation pop-up was also introduced to verify the user’s intention to discard an entire design when opting to go back without saving. Instructional text was refined to clarify the status of saved designs, now reading: “Your design is now saved! Feel free to go back and create new rules, shapes, and procedures, or start a new design whenever you’re ready.” This ensured that messages explicitly described the next available actions and minimized confusion regarding the application state.

Further interface snapshots from this prototype, including the new Composition shape dynamics and rule management interactions, are provided in Appendix D.

The same task set from the previous iteration was used to ensure comparability of results, with one additional task introduced: “Select the apartment shape.” This task was specifically designed to evaluate how participants would attempt to select a composition shape in its entirety, revealing which interaction strategy felt most intuitive.

3.5.1. Feedback and Insights

Feedback from this iteration was overwhelmingly positive, confirming the effectiveness of the latest interface refinements and the overall clarity of the system's workflow. Participants completed the tasks with minimal hesitation, demonstrating a clear understanding of the concepts of composition shapes, rule creation, and design generation. Only a few minor suggestions were raised, such as adding the instructional text "Drag shapes here" within the composition workspace to further guide users, and introducing a confirmation pop-up when deleting rules to prevent accidental removal.

Regarding the new task designed to assess the selection of a composition shape, participants consistently attempted to select the entire composition by clicking outside the currently selected shape (e.g., clicking outside the "bathroom" shape to select the overall "apartment"). These small refinements were incorporated into the final implementation of SAGE and later evaluated in the summative usability test.

With all major usability issues resolved, the prototyping phase was concluded, paving the way for the implementation of SAGE.

CHAPTER 4

Implementation Details

This chapter details the technical implementation of SAGE, describing how the conceptual framework and interface design presented in previous chapter were translated into a functional system. It describes the development environment, and then analyses the three core layers of the application, which aligns with the Model–View–Controller (MVC) pattern, highlighting how each contributes to the overall interaction between the user, the grammar logic, and the Prolog execution engine.

4.1. Development Environment

SAGE was implemented as a Java desktop application, extending the Alternative Shaper codebase with a redesigned interaction layer and Blockly-driven procedural authoring.[1] It was developed using JavaFX for the user interface, Maven for dependency management, and integrates the SWI-Prolog inference engine via the JPL bridge. While SAGE introduces a new visual and user-centered workflow, it directly employs the Alternative Shaper's Prolog engine for design generation, ensuring complete compatibility between both systems.

The interface structure is defined through FXML layouts, with styles encapsulated in CSS files, allowing a clear separation between presentation, logic, and behavior. The FXML markup describes static interface components such as the navigation tabs and grid canvases, while their dynamic interactions are implemented in dedicated controller classes.

The project's dependencies include:

- JavaFX (org.openjfx) – for UI rendering and WebView embedding;
- Jackson Databind – for JSON serialization and deserialization of grammar data;
- JUnit 5 – for local unit testing of logic and bindings;
- JPL (org.jpl7) – for the Java–Prolog interface enabling query execution and design generation.

SAGE follows a MVC pattern (Figure 4.1), ensuring modularity and clear separation of concerns. The View defines the user interface and visual feedback, the Controller manages user interaction and communication with the logic layer and the Model encapsulates the data structures representing Shape Grammar concepts.

[1]The complete source code is available at `https://github.com/marianaiscte/SAGE`.

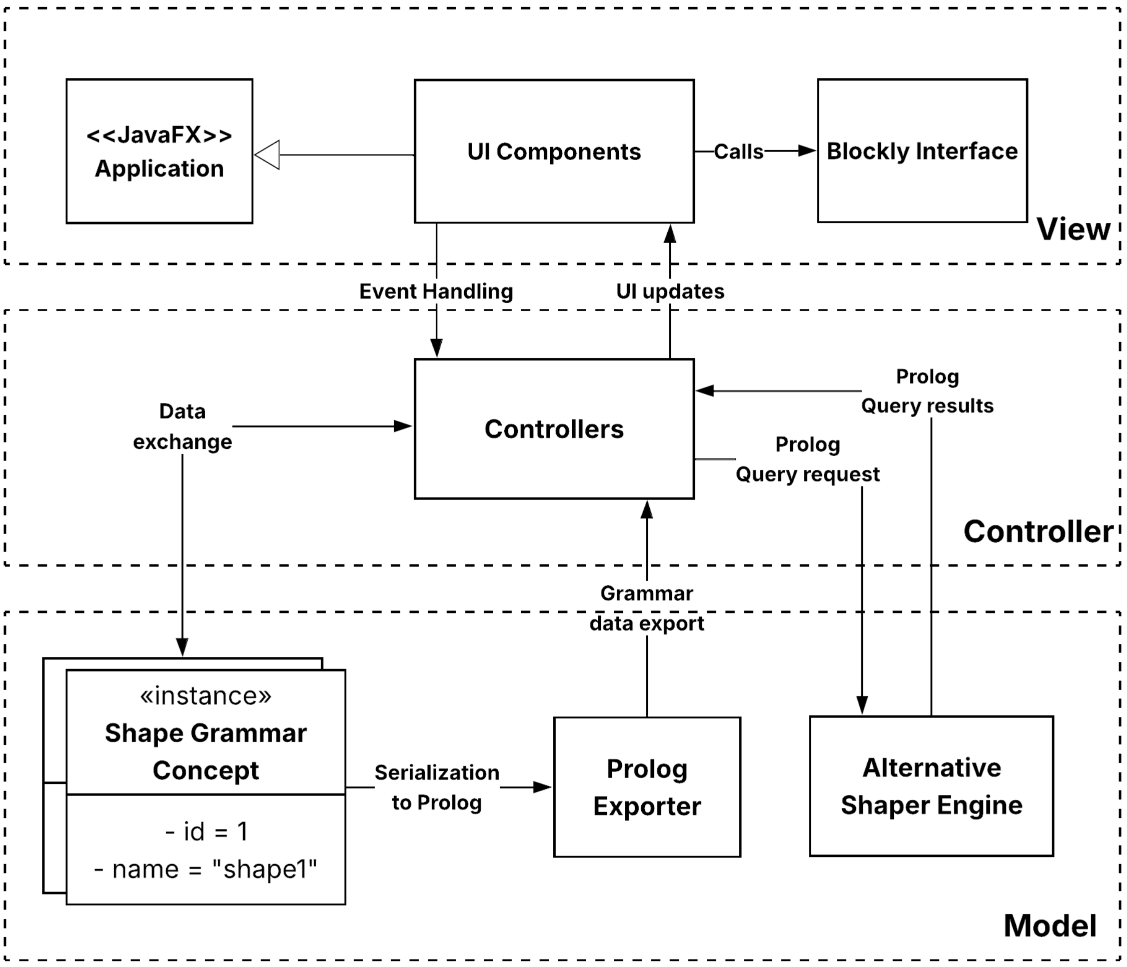


FIGURE 4.1. Model–view–controller architecture.

4.2. View Layer

The View layer defines the graphical interface of SAGE, implemented in JavaFX through FXML layouts and CSS stylesheets. It is composed of multiple scenes that guide the user throughout the workflow:

- Launch Scene — managed by the LaunchController, it displays the welcome screen, allows users to start new projects or load existing ones, and transitions to the main workspace (Figure 3.1);
- Main Scene — managed by the MainController, containing the "Shapes," "Rules," and "Grammar" tabs, where the user defines visual grammars (e.g. Figure 3.2);
- Design Generation Scene — managed by the DesignController, where the Prolog-based generation process is executed and visualized (Figure 3.5).

The Blockly environment embedded in SAGE was originally developed under the research project "Bolsa de Iniciação Científica" UIDP/04466/2020_04, within the broader Alternative Shaper initiative at ISTAR. This module was designed to provide a visual programming interface for constructing logical and procedural expressions through block-based manipulation, simplifying the interaction with Prolog predicates such as `ruleProcedure/2` and `shapeRule/2`.

Within SAGE, Blockly is integrated as a local HTML resource displayed inside a JavaFX `WebView`. It enables users to visually compose procedures and conditional logic through predefined block structures that mirror the syntax of the Alternative Shaper's grammar engine. In the final version of SAGE, used for the final summative evaluation, a simplified subset of Blockly blocks was employed, focusing on the essential procedural logic rather than the full expressive range of the original module. This streamlined configuration was intended to make the environment more accessible to participants unfamiliar with programming while still illustrating the connection between visual logic and Prolog execution. Additionally, a new category of "Rule" blocks was introduced to visually represent grammar transformations, allowing users to reference or trigger rules directly within the procedural flow, an extension not present in earlier prototype iterations.

However, since SAGE reuses the Alternative Shaper's Prolog engine, Blockly in this implementation operates as a design and learning interface rather than a persistent authoring environment. The visual logic created by users is not stored or executed directly as new procedures but instead reflects the structure of those already defined in the underlying engine.

This integration transforms formal Prolog logic into an accessible visual language, bridging textual programming and interactive design.

4.3. Controller Layer

Controllers coordinate user input, update the view, and synchronize data between the user interface and the Model layer. Each functional area of SAGE is handled by a specific controller or by a dedicated manager module associated with it, promoting modularity, separation of concerns, and ease of maintenance.

- LaunchController — manages the application's entry point, including the launch screen, project initialization, and navigation to the main workspace. It handles the creation of new projects and the loading of existing ones, establishing the initial session context before the main interface is displayed.
- MainController — serves as the core controller of the application. It manages global navigation within the Grammar composition workflow (creation of shapes, rules, and procedures), project state, and UI updates across tabs. It also coordinates several specialized managers responsible for visual editing and interaction logic.
- DesignController — coordinates Prolog communication with the Alternative Shaper engine during the design generation phase and visualizes the resulting compositions. Similar to the main workspace, this controller also integrates a grid environment and therefore instantiates its own `GridManager` for rendering and placement of generated shapes.

- RulesManager — assists the `MainController` by handling the creation and editing of transformation rules, connecting “before” and “after” shapes within the visual editor.
- GridManager — provides the logic for grid rendering, alignment, and shape placement. It is used both in the `MainController` (for editing and composing shapes) and in the `DesignController` (for displaying generated layouts).
- ThumbnailManager — manages the generation, display, and updating of shape thumbnails used for visual browsing. In SAGE, a new functionality was introduced allowing users to delete shapes directly from the thumbnail view via a small “×” button positioned in the upper-right corner of each thumbnail, a feature not present in earlier prototypes.
- ShapeBindingManager — synchronizes property bindings between the interface and the underlying model, supporting undo and redo operations to facilitate controlled editing.

These controllers and managers collectively implement SAGE’s interaction logic: user actions in the interface trigger updates in the Shape Grammar Concept, which are then serialized or converted into Prolog code via the PrologExporter. The controllers also mediate communication with the Alternative Shaper engine, issuing Prolog queries for design generation and retrieving the resulting compositions. This modular organization ensures consistent synchronization between the user interface, the Java data structures, and the underlying Prolog logic execution.

4.4. Model Layer

The Model layer in SAGE encompasses the internal data representation, persistence mechanisms, and the logical connection to the Alternative Shaper inference engine. It defines not only the grammar entities that structure visual compositions, but also the processes through which these entities are stored, translated into Prolog predicates, and executed to generate new designs.

At the core of this layer, all grammar components inherit from the abstract class `ShapeGrammarConcept`, which provides universal identifiers and names to ensure consistency and traceability across the system.

The abstract class `Shape` defines the geometric foundation of all visual elements, with properties such as `x`, `y`, `width`, and `height`. Two concrete subclasses extend this base:

- SimpleShape — represents a single geometric object with visual attributes such as color and thumbnail image;
- CompositionShape — aggregates multiple shapes, allowing hierarchical grouping and composite design structures.

In a composition, the coordinates (`x`, `y`) correspond to the bottom-left position of the entire structure, while the `width` and `height` describe the bounding box of the overall layout.

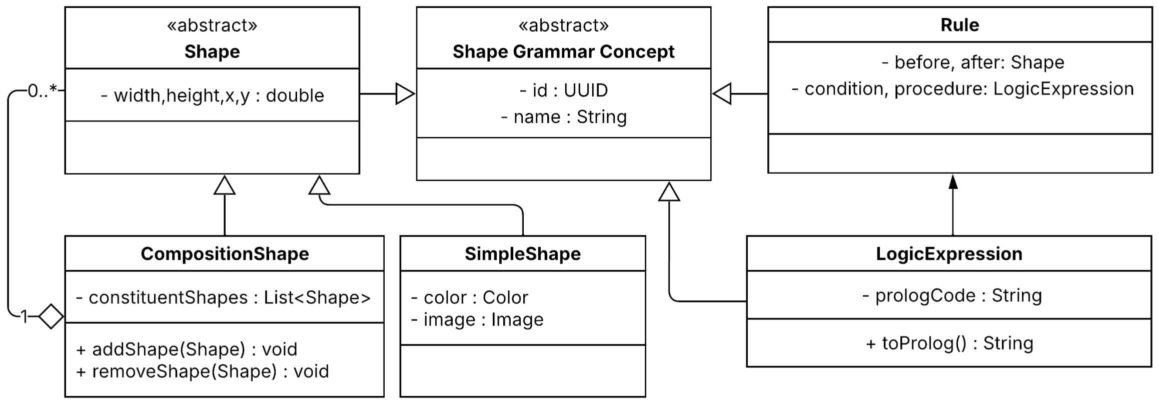


FIGURE 4.2. Shape Grammar core model hierarchy implemented in SAGE.

Rules are represented by the `Rule` class, which defines a transformation between two shape configurations: a "before" and an "after" state. Each rule can also include an optional condition or procedure, represented as a `LogicExpression`. In the current implementation, this class stores a String intended to hold the Prolog code generated from Blockly. Although the class structure is in place, it is important to remind that logic composed in Blockly does not currently propagate into SAGE. As such, the `LogicExpression` currently serves as a conceptual placeholder: it captures the intended Prolog output format, but no automatic transfer, storage, or execution of user-defined logic from the Blockly editor is performed.

Beyond defining the grammar structure, the Model layer also manages data persistence and logic translation. Two complementary mechanisms ensure the integrity and interoperability of SAGE projects:

- JSON serialization — for saving and reloading local projects, including shapes, rules, and procedures;
- Prolog export — for external interpretation and execution of grammars using the Alternative Shaper engine.

The PrologExporter component automatically generates Prolog predicates that describe the current grammar state, using the same lexical and structural conventions defined in the Alternative Shaper system. For instance, suppose the user creates three shapes: two Simple ones, named "one" and "two", each with dimensions of 1×1 and distinct colors; and a Composition shape "three" composed of the two previous shapes ("one" positioned at 0,0 and "two" at 0,1). Additionally, the user defines a rule named "Rule Name" that transforms shape "one" into shape "two". When the project is saved, SAGE automatically generates the following Prolog file:

```
scale_unit(40).

shape(one,'0x00ff00ff').
shape(two,'0x994d66ff').

basicShapeDimention(one,1,1).
basicShapeDimention(two,1,1).

shapeComposition(three, [
s(one, [1,0,0,0,1,0,0,0,1]),
s(two, [1,0,0,0,1,0,0,1,1])
]).

shapeRule('Rule Name', sr(one, two)).
```

This file defines all entities in the grammar using the Alternative Shaper-compatible syntax, ensuring that the shapes and rules created in SAGE can be directly interpreted by the same logical engine. In parallel, the system also produces a corresponding shapes.json and rules.json representation, which preserve the internal data structure of each entity, as shown below.

```
  [ {
  "id" : "aa56d5c2-e3f5-4287-a826-0724297013ce",
  "type" : "simple",
  "name" : "one",
  "width" : 1.0,
  "height" : 1.0,
  "x" : 0.0,
  "y" : 0.0,
  "color" : "#00FF00",
  "imageFilename" : null,
  "children" : null
}, {
  "id" : "d8dec135-1dd4-4bbc-959a-da665532fe0b",
  "type" : "simple",
  "name" : "two",
  "width" : 1.0,
  "height" : 1.0,
  "x" : 0.0,
  "y" : 0.0,
  "color" : "#994D66",
  "imageFilename" : null,
  "children" : null
}, {
  "id" : "6e70558a-756e-44a6-b77a-7fe94aa18a35",
  "type" : "composition",
  "name" : "three",
  "width" : 1.0,
  "height" : 1.0,
  "x" : 0.0,
  "y" : 0.0,
  "color" : null,
  "imageFilename" : null,
  "children" : [ {
    "id" : "e1768201-d379-42b8-9f84-ee2a1c668bcf",
    "type" : "simple",
    "name" : "one",
    "width" : 1.0,
    "height" : 1.0,
    "x" : 0.0,
    "y" : 0.0,
    "color" : "#00FF00",
    "imageFilename" : null,
    "children" : null
  }, {
    "id" : "722e8931-3a43-40e2-a966-1afbbbf1b0e1",
    "type" : "simple",
    "name" : "two",
    "width" : 1.0,
    "height" : 1.0,
    "x" : 1.0,
    "y" : 0.0,
    "color" : "#994D66",
    "imageFilename" : null,
    "children" : null
  } ]
} ]
}
```

```
   [ {
  "ruleId" : "59922afb-f96e-41e2-8cc4-21cad85bbb0c",
  "name" : "Rule Name",
  "shapeBeforeId" : "aa56d5c2-e3f5-4287-a826-0724297013ce",
  "shapeAfterId" : "d8dec135-1dd4-4bbc-959a-da665532fe0b"
} ]
```

However, the Prolog exporter operates independently from the Alternative Shaper engine: its role is limited to translating and saving the internal model into a compatible Prolog format, without initiating any inference or communication with the execution module. During the design generation phase, the controllers establish this connection instead, issuing Prolog queries through the Java–Prolog bridge (JPL). The Alternative Shaper engine processes these queries and returns the corresponding results as logical descriptions of generated designs.

For example, suppose the user selects the first procedure in the design to be "place-Basement", which is responsible for assembling the base layout of a design within the Alternative Shaper engine. In this case, the query invoked by SAGE to the engine would be:

```
shapeComposition(initialcell, IS), memory(global, M),
applyRuleProcedure(placeBasement, IS, FS, M, _).
```

This query instructs the engine to apply the placeBasement procedure to the default initial shape composition defined in the Alternative Shaper environment, producing a resulting set of shapes represented by the variable FS. The returned solution includes the following structure:

```
FS = [s(cell, [1.0, 0.0, 0, 0.0, 1.0, 0, -9.6, -4.5, 1]),
s(cell, [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, -9.6, -3.9, 1.0]),
s(cell, [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, -9.6, -3.3, 1.0]),
s(cell, [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, -9.6, -2.7, 1.0]),
s(cell, [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, -9.6, -2.1, 1.0]),
...]
```

In the user interface, this output is rendered as the generated design, as illustrated in Figure 4.3.

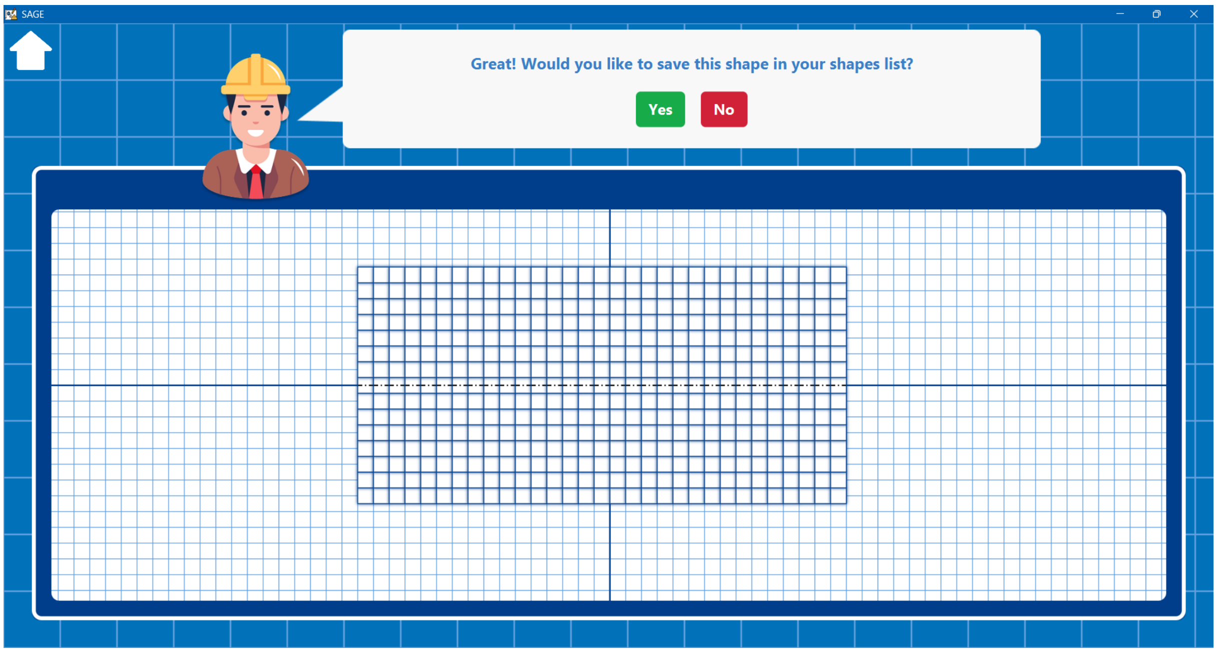

SAGE
Great! Would you like to save this shape in your shapes list?
Yes
No

FIGURE 4.3. Layout generated by the "placeBasement" procedure within SAGE.

If the user chooses to keep this generated layout, it can be saved as a new CompositionShape, following SAGE's internal representation for composite design structures.

4.5. System Interaction Flow

This section summarizes how the different architectural layers interact throughout the SAGE workflow, from user interaction to design generation. It describes the bidirectional data flow between the user interface, the grammar model, and the Prolog-based execution engine, illustrating how user actions are translated into formal logic operations and visual feedback (Figure 4.4).

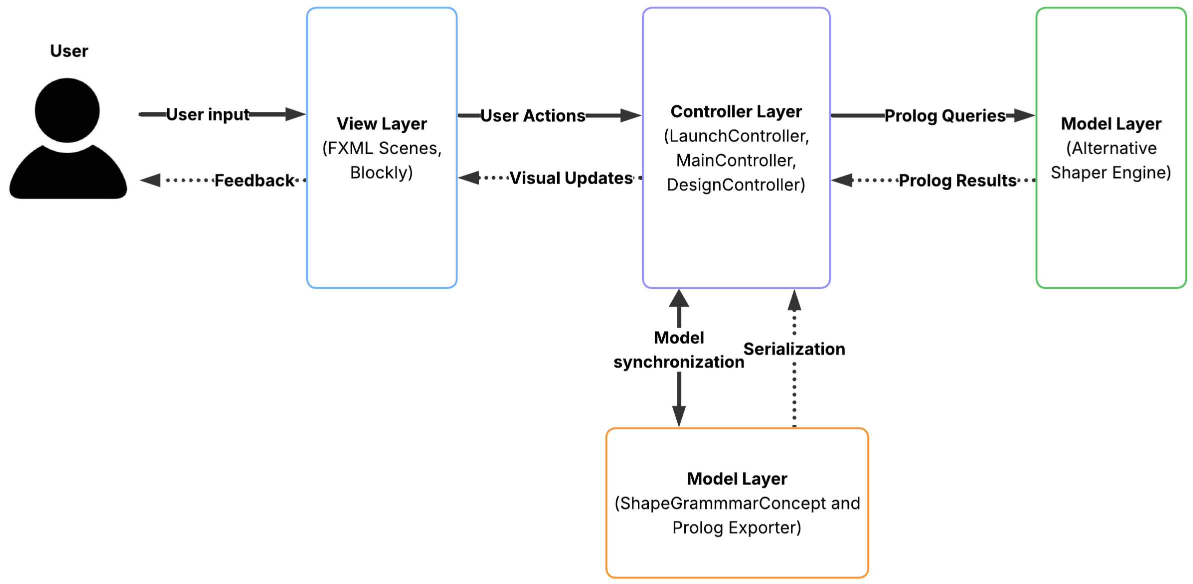


FIGURE 4.4. System interaction flow across SAGE's MVC layers and the Alternative Shaper engine.

When the user interacts with the interface, for example, by creating a new shape or defining a rule during the grammar composition phase, the Controller layer captures the event and updates the Model layer accordingly. The model then stores the corresponding grammar structures (e.g., Shape, Rule, or LogicExpression) and, when requested, serializes them into JSON and compiles them into Prolog predicates.

These JSON files preserve SAGE's internal object structure for local persistence and project reloading, while the Prolog file serves as the executable representation of the grammar, ready for interpretation by the Alternative Shaper engine.

During the Design Generation phase, the DesignController issues queries to the SWI-Prolog engine through the JPL bridge, invoking predicates defined in the Alternative Shaper logic (e.g., applyRuleProcedure(Procedure, InitialShapeComposition, FinalShapeComposition, InitialMemory, FinalMemory)). The resulting Prolog solutions are parsed and reinterpreted as composition layouts within the JavaFX environment, enabling the user to visualize and interact with generated designs in real time.

The process is cyclical and user-driven: feedback from the View layer (e.g., visual confirmation, shape placement, or rule animation) informs further actions, while updates propagate seamlessly through the MVC pipeline, maintaining system coherence.

CHAPTER 5

Evaluation

Following the completion of the application's functional development, a summative evaluation was conducted to assess its overall usability, clarity, and effectiveness. While earlier formative tests with medium-fidelity prototypes had already guided design improvements, this final evaluation aimed to measure the system's performance in its finished state and to gather evidence for discussion and future development.

The evaluation focused on understanding how users interacted with the final version of the application, particularly whether participants could intuitively perform the main design tasks, how they perceived the interface and interaction flow, and how effectively they understood and manipulated procedures using the Blockly integration.

5.1. Evaluation Goals and Methodology

5.1.1. Evaluation Setup

The evaluation combined quantitative metrics with qualitative observation, structured into two complementary parts. One part, conducted through the Maze usability testing platform, used an interactive Figma prototype [1] that replicated the final SAGE application, with the exception of an introductory tutorial. The other part took place directly within the SAGE desktop application, focusing on the procedure creation stage implemented with Blockly. This step was designed to assess how effectively users could understand, build, and apply procedural logic through visual programming. To ensure continuity, the application included the same shapes and rules created by each participant in the Maze test, allowing them to proceed seamlessly to the procedure definition and design exploration phases.

Maze is an online platform for remote usability testing that records behavioural metrics such as task completion rate, time per task, navigation paths, and misclick frequency. Its integration with Figma allows interactive prototypes to be tested as if they were real applications, while automatically collecting detailed analytics. Maze was chosen both for its analytic capabilities and for practical reasons: having only one evaluator, this setup enabled simultaneous observation and note-taking while participants completed tasks autonomously.

The short tutorial added to the Maze prototype provided essential theoretical context on Shape Grammars, an unfamiliar concept for most users, since a basic understanding of shapes, rules, and procedures was required to complete the tasks. Unlike conventional application tutorials, however, it did not include step-by-step instructions on how to

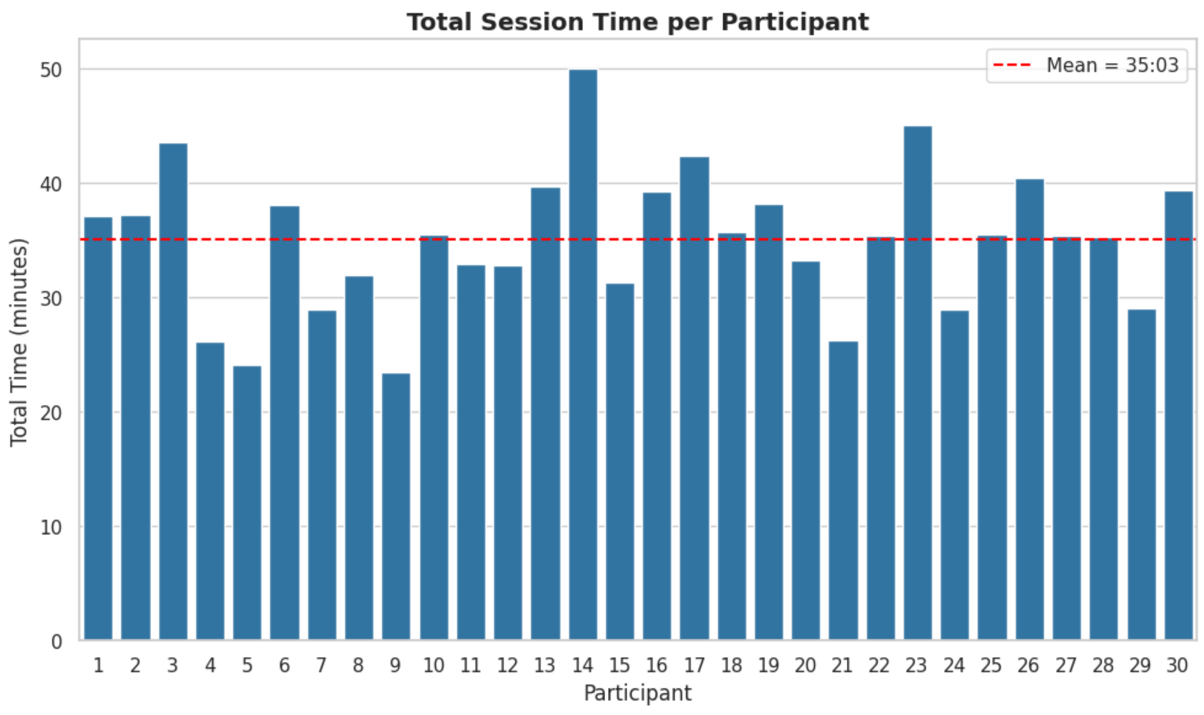
[1]Link to final Figma Prototype

operate the interface. This deliberate omission ensured that the evaluation would measure the system's intuitiveness and the participants' ability to learn through exploration, rather than their memory of procedural instructions.

To keep sessions concise and accessible, the task set was simplified compared with the full workflow available in the Alternative Shaper, which was originally designed to generate complete single-family house layouts through an extensive set of shapes, rules, and procedures. While the Alternative Shaper's design process involves multiple functional areas (entrance, semi-public, and private) and numerous room types (e.g., vestibule, kitchen, dining room, bedrooms, and bathrooms), the evaluation proposed a simplified scenario focusing on the creation of an apartment floor plan. This adaptation preserved the conceptual structure of the original workflow, progressing from shape and rule creation to design generation,but reduced its scale to ensure that participants could complete all tasks within a single session.

Each session lasted on average 35:03 minutes (*Standard Deviation (STD)* = 6:13, MIN = 23:27, MAX = 50:01), as indicated by the recorded session durations (see Figure 5.1). Tests were conducted individually with 30 participants affiliated with ISCTE – Instituto Universitário de Lisboa, primarily undergraduate students from diverse academic backgrounds.

FIGURE 5.1. Time per session (in minutes)



5.1.2. Tasks

Participants were instructed to complete a predefined set of 16 tasks designed to simulate the complete workflow within SAGE. These tasks progressed from basic shape creation to rule definition, procedural programming, and design generation, replicating a realistic use of the system.

The full sequence was as follows:

(1) Complete the tutorial;
(2) Create the Simple Shape "bedroom";
(3) Create the Simple Shape "bathroom";
(4) Create the Composition Shape "suite" (composed of "bedroom" and "bathroom") and select it as a whole;
(5) Create the Simple Shape "corridor";
(6) Create the Simple Shape "living room";
(7) Create the Simple Shape "kitchen";
(8) Create the Simple Shape "pantry";
(9) Create the Composition Shape "kitchen + pantry" (composed of "kitchen" and "pantry", with "pantry" positioned at coordinates (1,1));
(10) Create the Simple Shape "apartment size";
(11) Create the Rule "Suite";
(12) Create the Rule "Kitchen with pantry" and delete it;
(13) Create two procedures (one applying a sequence of two rules: "Suite" and "Kitchen with pantry"; and other with logic: apply the Rule "Suite" only if the Shape "bedroom" exists);
(14) Enter the Design Generation Phase and choose the first procedure ("placeApartment");
(15) Apply the second procedure ("placeCorridor") and choose the first solution;
(16) Apply the remaining procedures ("placeLivingRoom", "placeKitchen", "placeSuite") to achieve a floor plan design for an apartment and save the final design.

In this simplified grammar, the procedure "placeApartment" positioned the shape "apartment size" at the origin of the grid, defining the spatial limits of the design. The procedure "placeCorridor" places the shape "corridor" along the "apartment size" baseline: its bottom edge shares the same y-coordinate as the bottom edge of "apartment size". The corridor must fit entirely within the apartment's width while leaving at least two grid units of free space on both the left and right sides. Given an apartment width of 7 units, this constraint yields exactly three valid horizontal placements. Subsequent procedures, "placeLivingRoom", "placeKitchen", and "placeSuite", each positioned their respective shapes in adjacency to the "corridor", guaranteeing continuous spatial contact and preventing overlap.

Tasks 1–12 (tutorial, Shape, and Rule creation) were conducted using the Maze prototype. Task 13 (procedure creation and logic definition) was performed directly within the

SAGE desktop application using Blockly. Finally, tasks 14–16 (Design exploration and saving) returned to the Maze prototype to evaluate interface clarity and comprehension of the generative process.

Upon completion of all tasks, participants obtained a final apartment layout. The specific outcome could vary depending on procedural choices, a total of 82 distinct valid designs were possible within the test scenario. Figure 5.2 illustrates one of these possible results, representing a complete apartment layout composed of the shapes defined earlier ("apartment size" for the grey outline and the shapes “suite,” “corridor,” “living room,” and “kitchen + pantry” representing the divisions).

Additional interface snapshots from the final SAGE prototype, including the tutorial sequence, the shapes and rules created, the intended procedures, and examples of design generation steps, are presented in Appendix E.

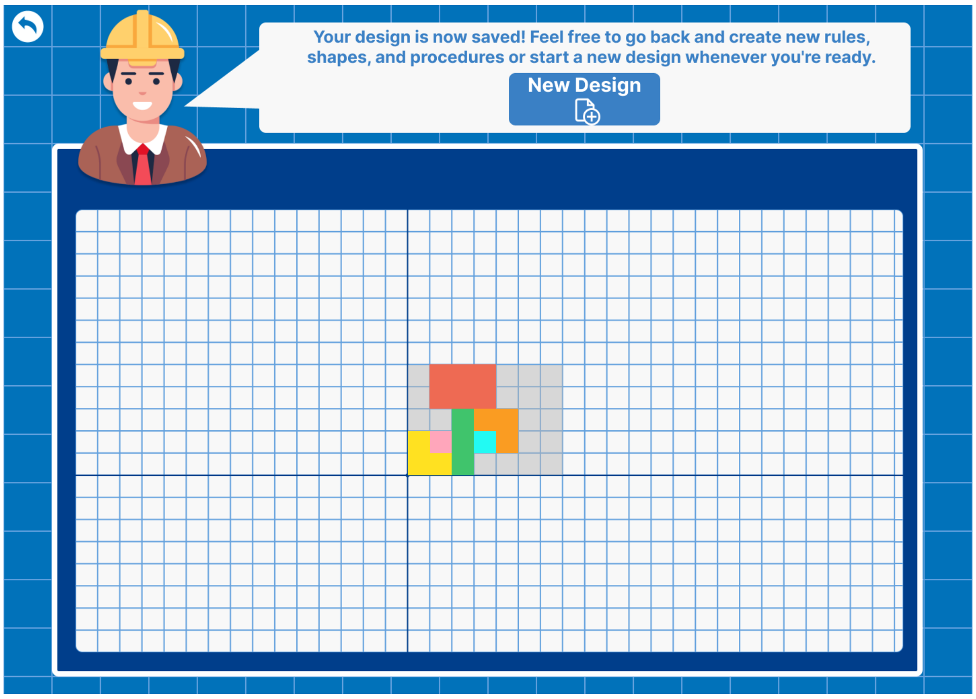

Your design is now saved! Feel free to go back and create new rules, shapes, and procedures or start a new design whenever you're ready.

FIGURE 5.2. Example of a final apartment layout generated during the evaluation test.

In addition to these main tasks, participants were asked to identify the function of several interface buttons not directly tested (e.g., saving and open buttons) and to express their preference between two icons for exiting the design phase: an arrow-shaped button (prototype version) or a home-shaped icon (final application version).

5.1.3. Data Collected

Three main data categories were collected:

- Behavioral metrics — automatically captured through Maze, including task completion rates, time spent per task, and misclick frequency;
- Perceived Usability metrics — gathered through questionnaires and rating scales assessing usability and satisfaction;
- Qualitative feedback — obtained from open-ended questions and direct observation, providing insight into user perceptions, challenges, and suggestions.

At the start of each session, participants also provided demographic information, including age, field of study, experience with digital design tools (None, Basic, Intermediate, Advanced), and familiarity with Shape Grammars (None, Heard of them, Some experience, Advanced experience). This information allowed for contextual interpretation of the results in relation to participants' prior knowledge and digital literacy.

5.1.4. Perceived Usability Metrics

Participants were asked to provide perceptual feedback on their experience with the system to complement the behavioural metrics. This assessment combined standardized questionnaires and open-ended questions to capture both quantitative and qualitative insights. Table 5.1 presents the complete questionnaire, comprising the ten System Usability Scale (SUS) items, three additional Likert-scale items on satisfaction and perceived learning, and three open-ended questions. All Likert items were rated on a five-point scale (1 = Strongly disagree, 5 = Strongly agree).

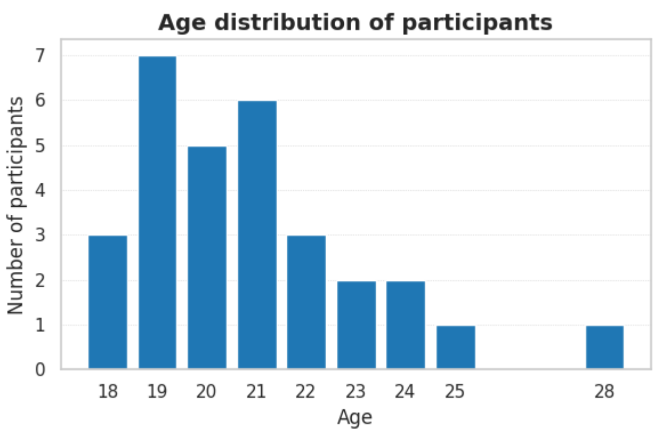
#	Type	Question
1	SUS	I think that I would like to use this system frequently.
2	SUS	I found the system unnecessarily complex.
3	SUS	I thought the system was easy to use.
4	SUS	I think that I would need the support of a technical person to be able to use this system.
5	SUS	I found the various functions in this system were well integrated.
6	SUS	I thought there was too much inconsistency in this system.
7	SUS	I would imagine that most people would learn to use this system very quickly.
8	SUS	I found the system very cumbersome to use.
9	SUS	I felt very confident using the system.
10	SUS	I needed to learn a lot of things before I could get going with this system.
11	Likert-scale	I enjoyed using the application.
12	Likert-scale	The interface was visually pleasant.
13	Likert-scale	The application helped me understand Shape Grammars.
14	Open-ended question	What did you like the most about the application?
15	Open-ended question	What did you find difficult to understand?
16	Open-ended question	What improvements would you suggest?

TABLE 5.1. Complete post-test questionnaire, including SUS, Likert-scale, and open-ended items.

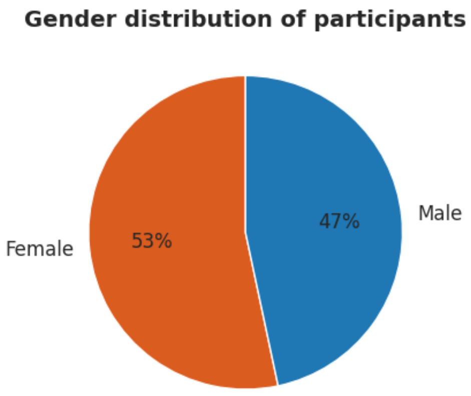
5.2. Results and Analysis

5.2.1. Participant Profile

Like previously mentioned a total of 30 participants took part in the summative evaluation. All were affiliated with ISCTE – Instituto Universitário de Lisboa, mostly undergraduate students from a range of study programmes. The participants' ages ranged from 18 to 28 years (*Mean (M)* = 20.67, *Median (MD)* = 20.5, *STD* = 2.48), indicating a young sample (see Figure 5.3a).The gender distribution was balanced, with 16 female and 14 male participants (Figure 5.3b).



(A) Age distribution of participants.



(B) Gender distribution of participants.

FIGURE 5.3. Demographic distribution of participants by age and gender.

In terms of academic background, the most represented programme was Computer Engineering (6 participants), followed by Economics, Management, and Psychology (3 participants each). Other areas included Architecture, Data Science, Political Science, Telecommunications and Computer Engineering, and Informatics and Business Management (2 participants each), with isolated cases from Finance and Accounting, Marketing Management,Industrial and Logistics Management, Social Work and Sociology (1 participant each). This variety ensured a balanced representation of both technical and non-technical profiles (see Figure 5.4).

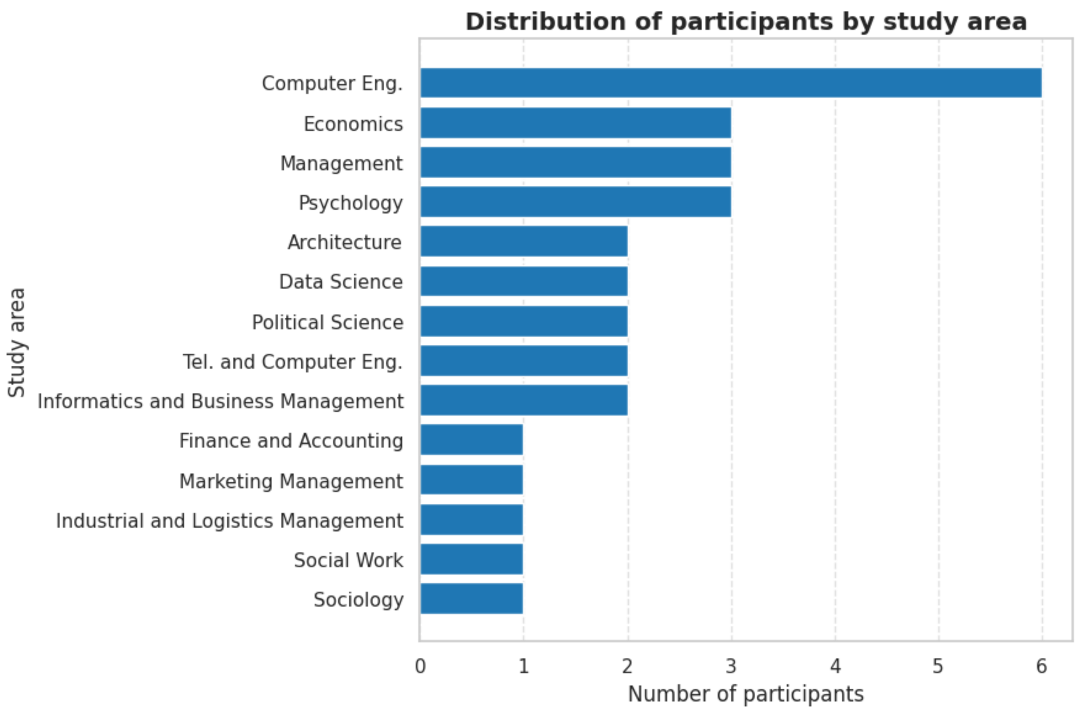
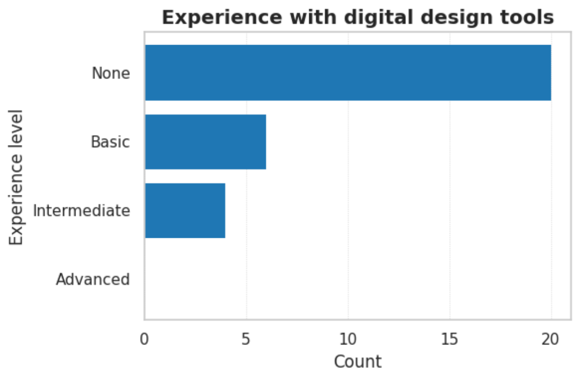


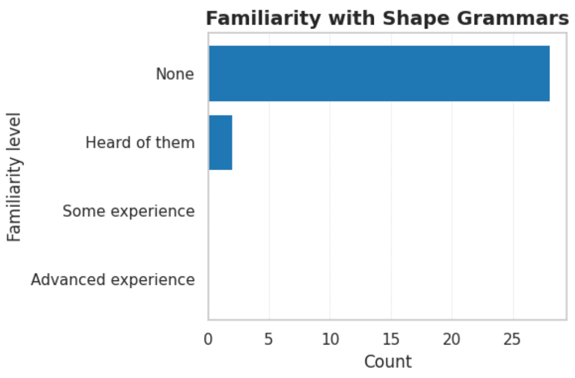
FIGURE 5.4. Distribution of participants by study area.

Regarding experience with digital design tools, most participants reported little or no prior experience: 20 had no experience, 6 had basic experience, and 4 had intermediate experience; none identified as advanced users (Figure 5.5a). Similarly, familiarity with Shape Grammars was very limited, with 28 participants indicating no prior knowledge and only 2 reporting that they had heard of the concept (Figure 5.5b).

This profile suggests that the participant group consisted primarily of novice users with minimal exposure to design or generative systems. Consequently, their performance and perceptions are particularly relevant for assessing the learnability, intuitiveness, and educational potential of the SAGE application when introduced to first-time users.



(A) Participants' experience with digital design tools.



(B) Participants' familiarity with Shape Grammars.

FIGURE 5.5. Participants' background regarding digital design tools and Shape Grammars.

5.2.2. Quantitative Results from Questionnaires

5.2.2.1. *System Usability Scale (SUS)*

The SUS was employed to obtain a standardized measure of perceived usability. The SUS provides a reliable benchmark for assessing the overall quality of user experience, with scores above 68 typically regarded as acceptable and those exceeding 80 considered excellent [72].

The individual SUS scores are summarized in Table 5.2 and are available in Appendix F. The overall mean score was 78.08 ($MD = 77.5$, $STD = 13.3$), which places the application in the *"Good to Excellent"* usability range.

Statistic	Value
Mean	78.08
Median	77.5
Standard Deviation	13.3
Minimum	45.0
Maximum	95.0
95% Confidence Interval	[73.1, 83.0]

TABLE 5.2. Descriptive statistics of SUS scores (N=30).

A 95% confidence interval was calculated to estimate the precision of the sample mean and the expected range of usability perceptions within the broader user population. The relatively narrow interval (73.1–83.0) indicates a stable and reliable mean estimate, suggesting that the observed usability level would likely generalize to similar user groups. Even at the lower bound, perceived usability remained above the standard threshold of 68, confirming a consistently positive user experience across participants.

The standard deviation (13.3) denotes moderate variability in responses, suggesting that while perceptions of usability were generally positive, a few participants experienced minor differences in ease of use or familiarity. Nonetheless, the relatively narrow confidence interval reinforces the consistency of positive evaluations. The distribution of scores (Figure 5.6) further supports this observation, showing that most participants rated usability highly, with only one outlier below 50.

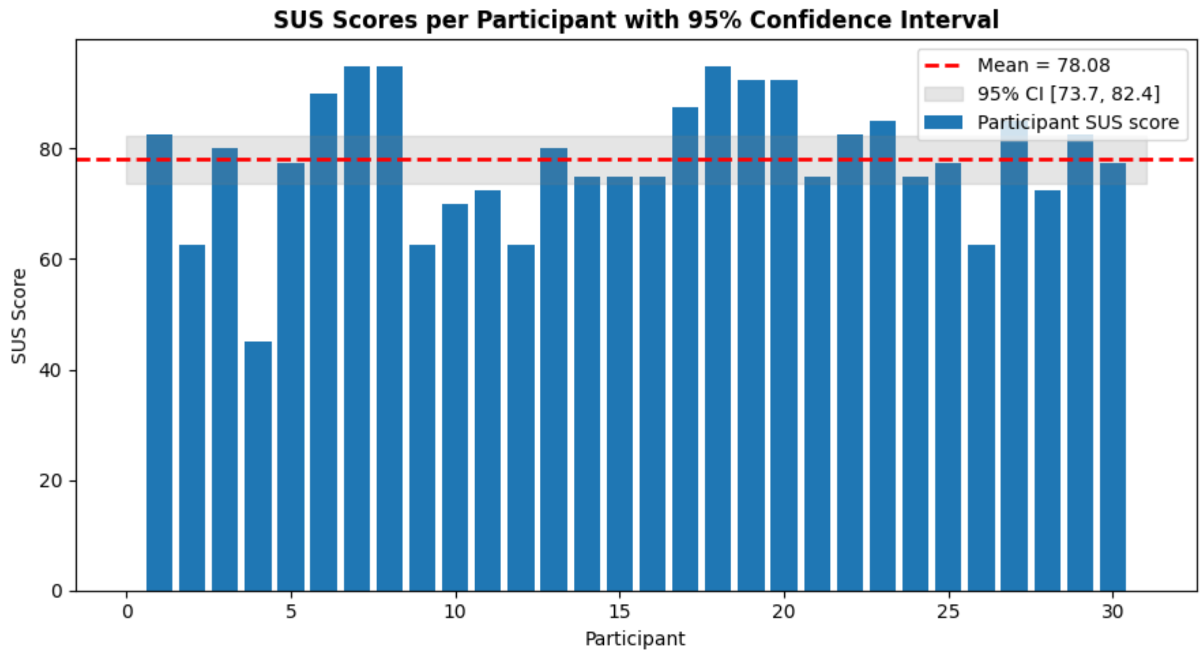


FIGURE 5.6. Distribution of individual SUS scores with overall mean line and 95% confidence interval.

To examine whether prior experience influenced perceived usability, a Pearson correlation was computed between participants' self-reported experience with digital design tools and their SUS scores. The Pearson correlation coefficient (r) measures the strength and direction of a linear relationship between two continuous variables, ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation).

The result indicated a weak and non-significant correlation ($r = 0.14$, $p = 0.462$), suggesting that usability perceptions were largely independent of prior technical experience. Both novice and more experienced users rated the system positively, suggesting that its intuitiveness and clarity mitigated any potential disadvantage associated with limited familiarity with design software.

5.2.2.2. *Additional Likert Questions*

Participants also answered three additional Likert-scale questions (1 = Strongly Disagree, 5 = Strongly Agree). Table 5.3 summarizes the responses.

Question	**Mean**	**STD**	**Response Distribution (1–5)**
I enjoyed using the application	4.30	0.70	(1:0, 2:0, 3:4, 4:13, 5:13)
The interface was visually pleasant	4.53	0.57	(1:0, 2:0, 3:1, 4:12, 5:17)
The application helped me understand Shape Grammars	4.20	0.85	(1:0, 2:1, 3:5, 4:11, 5:13)

Table 5.3. Summary of responses to additional Likert questions (N=30).

Across all three statements, mean ratings ranged between 4.2 and 4.5, indicating consistently positive perceptions across enjoyment, aesthetics, and learning outcomes. This convergence suggests that participants not only appreciated the interface but also perceived it as both engaging and educational. Participants demonstrated more agreement on the interface's visual appeal (STD = 0.57), while responses regarding the learning dimension ("The application helped me understand Shape Grammars") showed slightly greater variability (STD = 0.85), indicating differing levels of self-perceived understanding.

A Pearson correlation analysis between enjoyment ("I enjoyed using the application") and perceived understanding ("The application helped me understand Shape Grammars") revealed a strong positive relationship ($r = 0.65$, $p < 0.001$), suggesting that users who found SAGE more enjoyable also reported a clearer grasp of Shape Grammar concepts. This reinforces the link between positive affect and cognitive engagement in interactive learning systems.

To further explore these relationships, correlations were computed between the overall SUS scores and the three Likert-scale items. Results showed a strong positive association between usability (SUS) and enjoyment ($r = 0.60$, $p < 0.001$), and a moderate, significant correlation with perceived understanding of Shape Grammars ($r = 0.55$, $p = 0.0017$). No significant relationship was found with visual appeal ($r = 0.17$, $p = 0.3622$).

Overall, these findings indicate that higher usability perceptions were closely aligned with greater enjoyment and learning outcomes, underscoring the role of intuitive interaction design in promoting both engagement and conceptual understanding. The lack of association with visual appeal suggests that aesthetic quality, while appreciated, was not a primary determinant of perceived usability.

5.2.3. Behavioral metrics

The Maze platform provided detailed information on participants' temporal and behavioral performance across 15 of the 16 tasks (excluding the Blockly task, which was performed directly within the SAGE application). In addition to numerical metrics, Maze generated heatmaps displaying participants' click distributions for each screen, enabling a more detailed inspection of interaction behavior and error patterns.

From these tests, it was possible to collect three main quantitative metrics: time per task, success rate and misclick rate.

Since Maze imposes a limit on the number of tasks per questionnaire, the full workflow was divided into three sequential surveys. Consequently, tasks were grouped into four phases reflecting the logical progression of the workflow:

- Tasks 1–5, covering the tutorial and initial shape creation;
- Tasks 6–12, involving the creation of remaining shapes and rules;
- Task 13, performed directly in SAGE for procedural and logical creation;
- Tasks 14–16, focused on design generation and saving.

It is important to note that Maze automatically marks a task as "successful" once a participant reaches the intended final screen, without accounting for errors, retries, or evaluator assistance. To obtain a more accurate representation of actual performance, all heatmaps and observation notes from the sessions were manually reviewed and, based on this qualitative assessment, three levels of success were defined for each task:

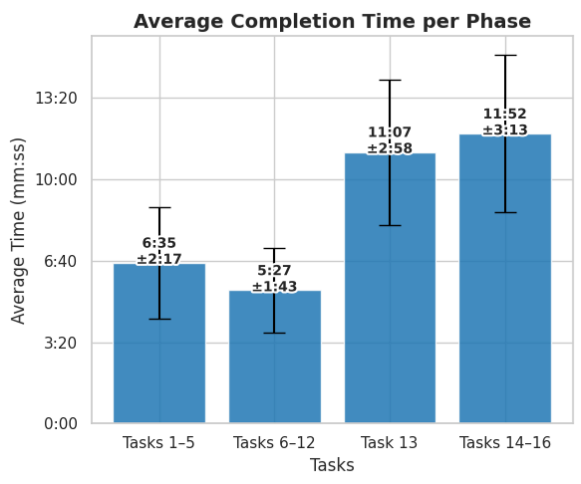
- Full success (1.0) — the participant completed the task correctly and independently;
- Partial success (0.5) — the participant initially tried incorrect actions or completed the task only partially, but reached the correct outcome;
- Failure (0.0) — the task required evaluator intervention, was abandoned, or involved repeated unsuccessful attempts.

This adjusted approach ensures a more faithful reflection of participants' actual performance, complementing the automatically collected data with a qualitative understanding of user behavior.

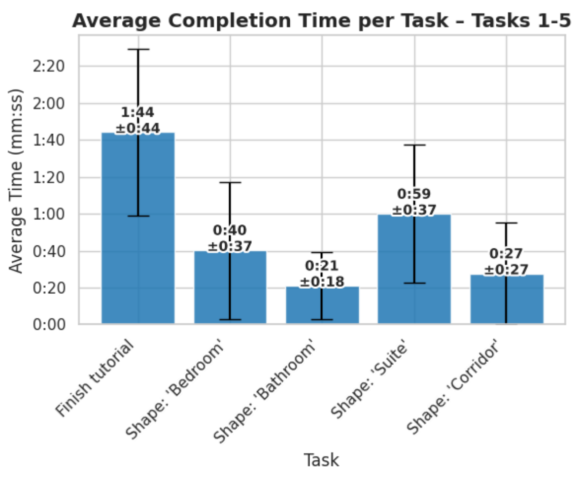
For the Blockly task, which was executed outside the Maze environment, the time spent on this component was estimated by subtracting the timestamp marking the start of the Design generation from the end of the Tasks 6–12 group. The same success evaluation strategy described above was applied to this task to ensure consistent assessment across all stages of the workflow.

5.2.3.1. *Time per Task*

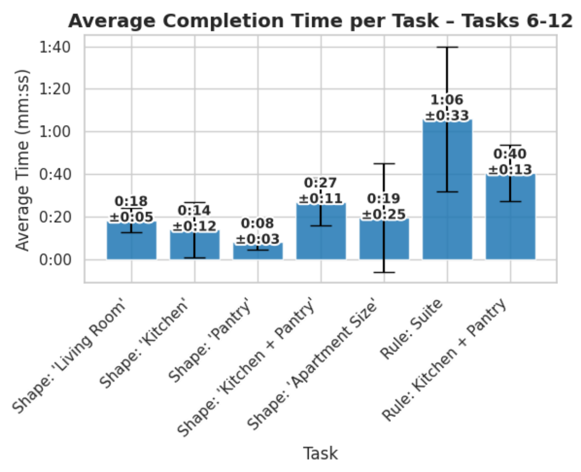
Regarding time spent completing each phase, Figure 5.7 presents the average duration (Mean) and standard deviation (STD) per phase, alongside the mean completion time for each individual task within the corresponding stages. Additionally, Figure 5.8 provides an overview of the overall temporal trend across all tasks, allowing a clearer observation of how completion times and variability evolved throughout the workflow.



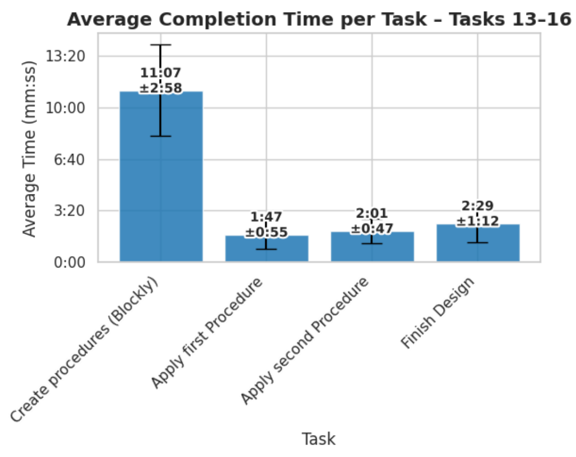
(A) Average completion time per phase.



(B) Average completion time per task – Tasks 1–5.



(C) Average completion time per task – Tasks 6–12.



(D) Average completion time per task – Tasks 13-16.

Figure 5.7. Average completion times per phase and per task.

It is important to note that the cumulative duration of all tasks within a phase does not necessarily match the total session time recorded for that same phase. This discrepancy occurs because Maze tracks both the overall session duration and the active time spent on each individual task separately. The difference between these two measures corresponds to unrecorded intervals between tasks, moments often used by participants to reflect on previous actions, plan their next step, or provide feedback to the evaluator. Although these pauses were not captured by Maze's automated task timing, they nonetheless contributed to the total session duration.

As shown in Figure 5.7a, the average completion time for the initial tasks (1–5) was approximately 6:35 minutes ($STD = 2{:}17$ minutes), reflecting participants' adaptation to the interface and the time required to complete the tutorial. The following phase (Tasks 6–12) presented a reduced mean of 5:27 minutes ($STD = 1{:}43$ minutes), indicating greater familiarity and efficiency once users had internalized the basic interaction model.

The most time-consuming activity occurred in Task 13 (M = 11:07 minutes, STD = 2:58 minutes), corresponding to the Blockly procedure creation step, which demanded higher cognitive effort and logical reasoning. The final phase (Tasks 14–16) maintained a similarly high duration (M = 11:52 minutes, STD = 3:13 minutes), reflecting the progressive complexity of the design generation process.

During the first five tasks, most of the time was spent completing the tutorial, which had an average duration of 1:44 minutes (STD = 44 seconds). This result was expected, as participants were not only learning how to navigate the interface but also absorbing the theoretical concepts introduced in the tutorial.

Regarding shape creation, it is notable that the average time required to create Simple shapes decreased over time, with tasks such as creating the Simple Shape Bathroom (21 s ± 18 s), Corridor (27 s ± 27 s), Living Room (18 s ± 5 s), Kitchen (14 s ± 12 s), Pantry (8 s ± 3 s), and Apartment Size (19 s ± 25 s) showing faster completion rates than the first one created, Bedroom (40 s ± 37 s). However, these tasks also presented high variability; in some cases, the standard deviation approached or even matched the mean, indicating that some participants completed them almost instantly, while others took considerably longer. This dispersion suggests heterogeneous exploration strategies, typical of early interaction phases when users are still familiarizing themselves with available controls. The higher variability observed in Bathroom, for instance, is explained by the fact that some participants attempted to modify the previously created Bedroom shape instead of generating a new one, resulting in longer completion times. The creation of the Composition Shapes Suite (59 s ± 37 s) and Kitchen + Pantry (27 s ± 11 s) followed a similar trend to that observed in Simple shape creation, with shorter times reflecting increased familiarity and confidence.

Rule creation tasks took longer on average, particularly the Rule Suite (1:06 minutes ± 33 s) compared to the following rule, Kitchen + Pantry (40 s ± 13 s). The peak observed during rule creation aligns with session notes, indicating that even though the requested rules were relatively simple substitution rules (placing one shape in the "before" and another in the "after" state), participants took time to understand the broader possibilities of rule definition, particularly due to the presence of conditions and procedures.

Finally, the design generation tasks (Tasks 14–16) showed a gradual increase in completion time, consistent with the growing complexity of each step: the first required applying a single procedure (1:47 minutes ± 55 s), the second selecting another with several possible outcomes (2:01 minutes ± 47 s), and the third combining multiple procedures to obtain the desired result (2:29 minutes ± 1:12).

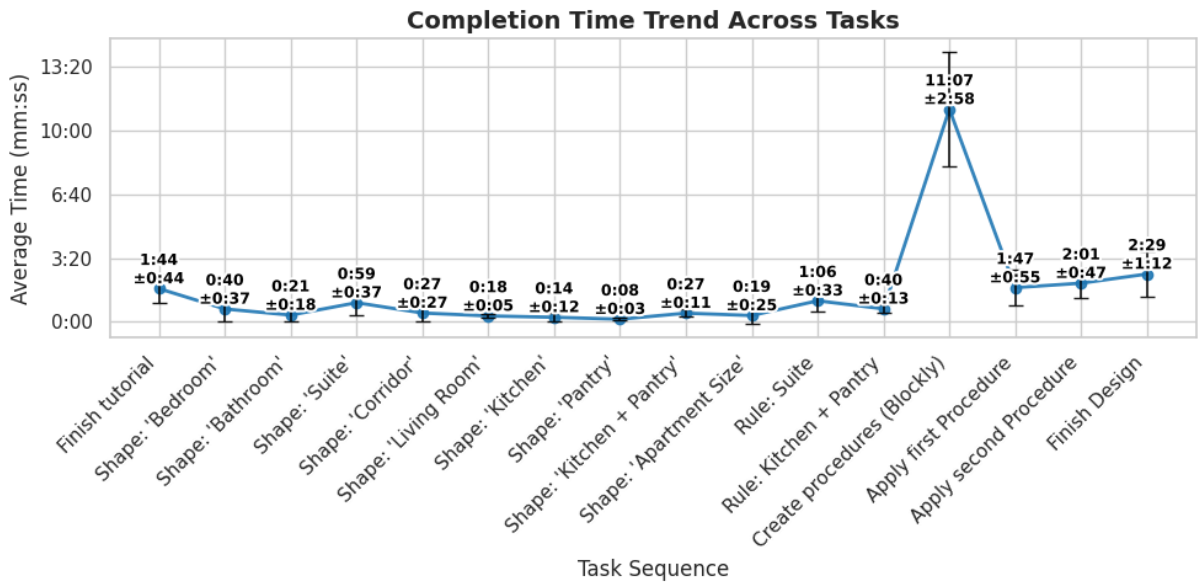


FIGURE 5.8. Overall trend of average completion time across the entire task sequence.

5.2.3.2. *Success Rate*

Figure 5.9 summarizes the average success rate across the four evaluation phases, revealing consistently high task completion levels, with values above 80% in all stages except for the Blockly phase.

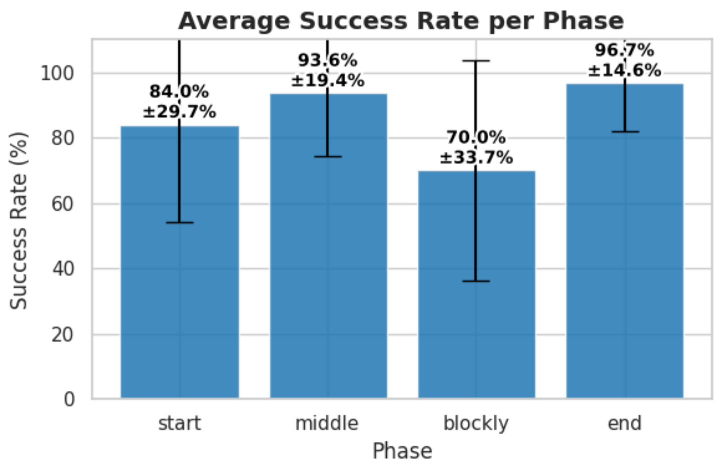


FIGURE 5.9. Average success rate across all four evaluation phases.

The Tasks 1–5 reached an average success rate of 84% ($STD = 29.7\%$), which aligns with the expected learning curve during the tutorial and initial shape creation. The relatively high variability indicates that while most participants completed the introductory steps successfully, some experienced initial hesitation or minor errors. This dispersion reflects different learning paces as users familiarized themselves with the interface and the underlying shape grammar concepts.

Performance improved significantly in the group of Tasks 6–12, achieving a mean success rate of 94% ($STD = 19.4\%$). The lower variability compared to the previous phase suggests that participants became more consistent and confident in their interactions. At this stage, they had already internalized the workflow for creating and managing shapes but were still consolidating their understanding of rule creation, introduced for the first time in this phase. The high accuracy nonetheless indicates that the interface effectively supported the acquisition of these more abstract concepts, even if the related tasks required slightly longer completion times.

The Blockly phase (Task 13), with an average success rate of 70% ($STD = 34\%$), presented the most challenges and the highest dispersion of results. Observations revealed that some participants completed the task successfully on the first attempt, while others required multiple retries or evaluator guidance. This high standard deviation highlights the heterogeneity in participants' prior familiarity with logic-based problem solving. Tasks completed only after multiple attempts or contextual hints were classified as partial successes (0.5), whereas repeated failures or direct intervention by the evaluator were marked as unsuccessful. Given the conceptual shift toward visual programming and logical reasoning, this variability was anticipated.

Finally, performance peaked in Tasks 14–16 (Design generation), with an average success rate of 97% ($STD = 15\%$). Despite the small variability, the overall consistency across participants was remarkable. An almost perfect score was expected, as this stage was primarily exploratory and guided through the mascot's prompts during the design creation process. Because the outcomes depended on following predefined instructions, the likelihood of major errors was minimal. The few partial results occurred when participants did not immediately locate the "Build" icon, but once identified, the task was completed successfully, confirming both the intuitiveness of the generative component and the clarity of the final workflow.

To further illustrate these trends, Figure 5.10 details the average success rate for each individual task. The pattern mirrors the overall progression discussed above: high consistency across most tasks, with temporary declines in those introducing new or more complex interactions, namely the creation of the Suite (65%), Kitchen + Pantry (72%), and Blockly procedures (70%). These minor decreases align with the expected learning curve and do not necessarily indicate usability issues, as participants quickly adapted and recovered in the subsequent steps.

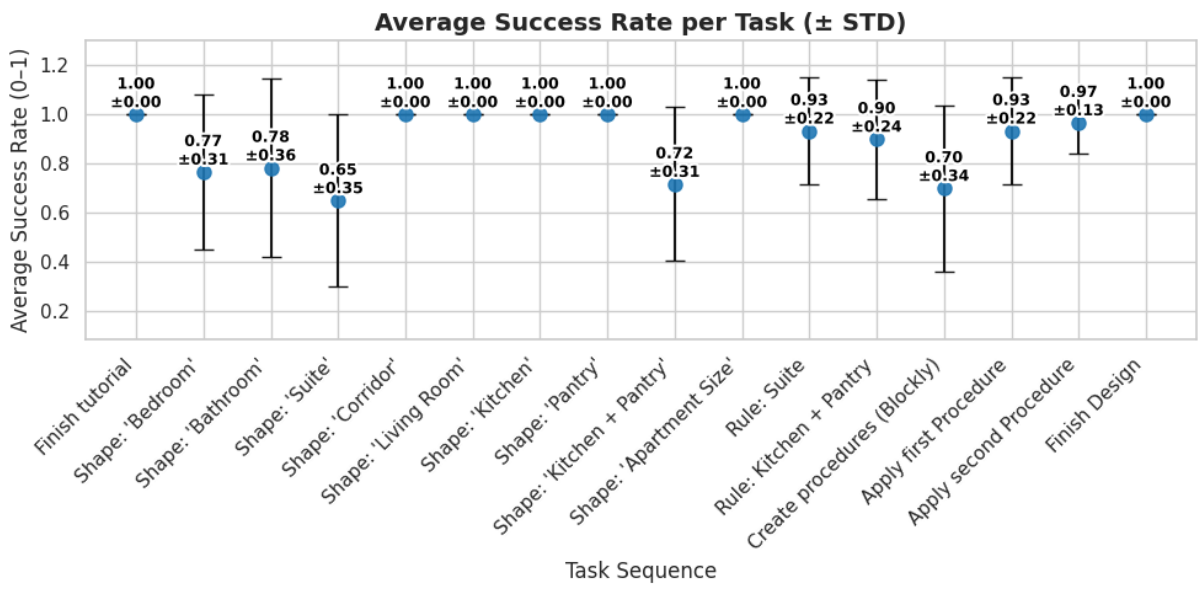


FIGURE 5.10. Average success rate per task across the full workflow sequence.

To examine whether longer task durations were associated with lower performance, a correlation analysis was conducted between completion time and success rate (Figure 5.11). Although a weak negative correlation ($r = -0.21$) was observed between completion time and success rate, this relationship is not strong enough to draw definitive conclusions. Longer task durations occasionally reflected moments of exploration or reflection rather than genuine difficulty, suggesting that time alone is not a reliable indicator of performance quality.

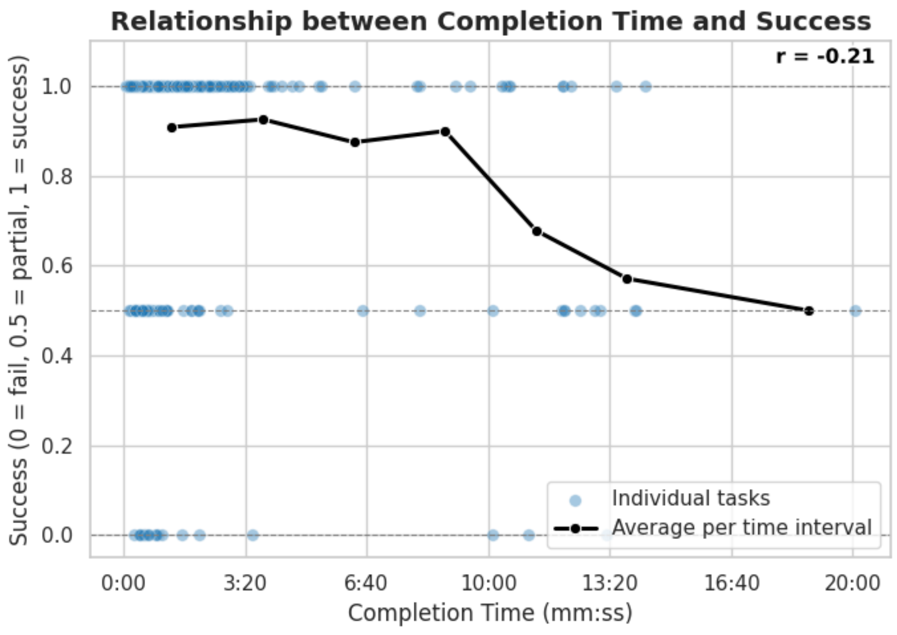


FIGURE 5.11. Relationship between completion time and task success across all participants

While overall task success was high, the analysis of misclicks provides further insight into how users interacted with the interface and where exploratory errors tended to occur.

5.2.3.3. *Misclick Rate*

Figure 5.12 illustrates the distribution of misclick rates across all tasks. The results show that errors were most frequent during the creation of composite shapes (Suite with 43.5% and Kitchen + Pantry with 39.8%) where users often attempted to manipulate elements directly on the canvas or misinterpreted the required sequence of actions. Similarly, early tasks such as the Bedroom and Bathroom shapes also registered moderate error rates (25–33%), consistent with the initial exploration period.

In contrast, once participants had internalized the workflow, misclick frequency decreased significantly, remaining below 10% during the rule creation and design generation stages. The Blockly task (Task 13) and the final Design task (Task 16) were excluded from this metric, as the former was executed outside the Maze environment and the latter was an exploratory task without a predefined target screen.

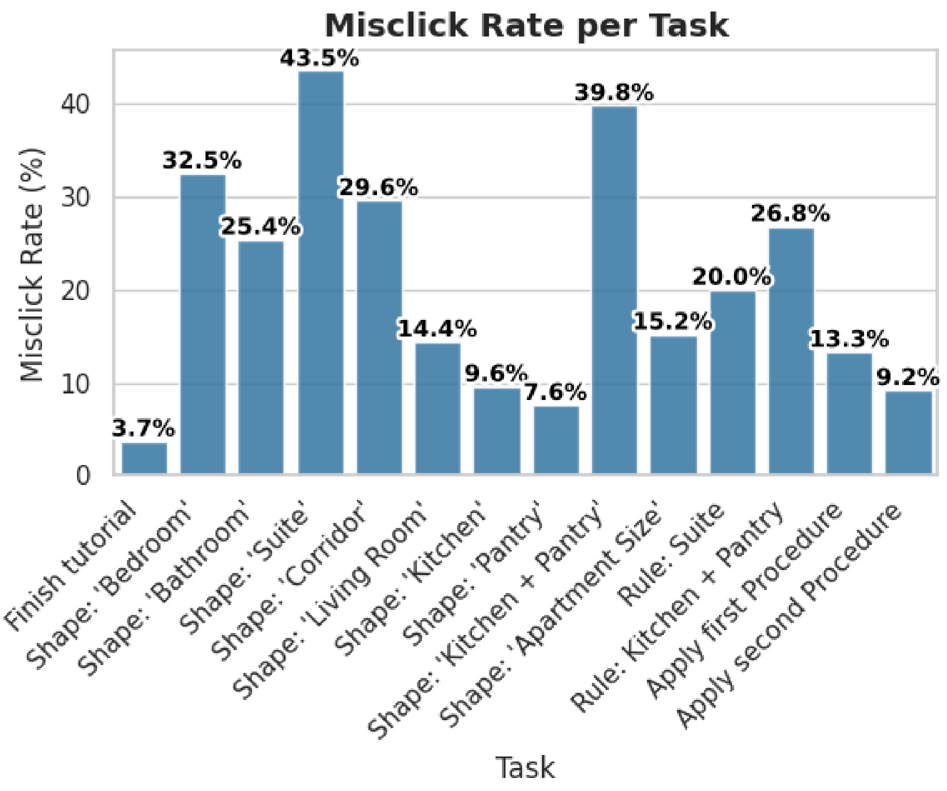


FIGURE 5.12. Misclick rate per task

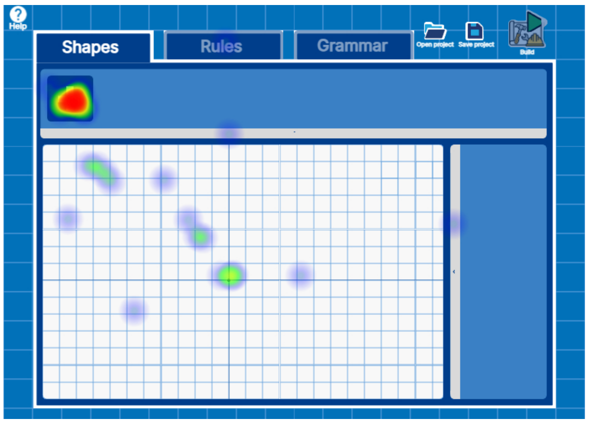
To better understand these quantitative results, all heatmaps generated by Maze were examined. Each task's heatmap was analyzed to identify potential causes for interaction errors or atypical click distributions.

It is important to clarify that, for the purpose of this evaluation, a specific interaction flow was defined to ensure consistent data collection across participants. For example, when creating a shape, the expected order of actions was: first assign a name, then adjust its size, and finally set its position. Consequently, any deviation from this predefined sequence, such as modifying a shape's position before naming it, was automatically logged by Maze as a misclick. However, these cases do not represent genuine usability issues, as the final version of SAGE will allow users to perform these actions in any order. Therefore, many of the recorded misclicks correspond to alternative, valid interaction strategies rather than actual user errors.

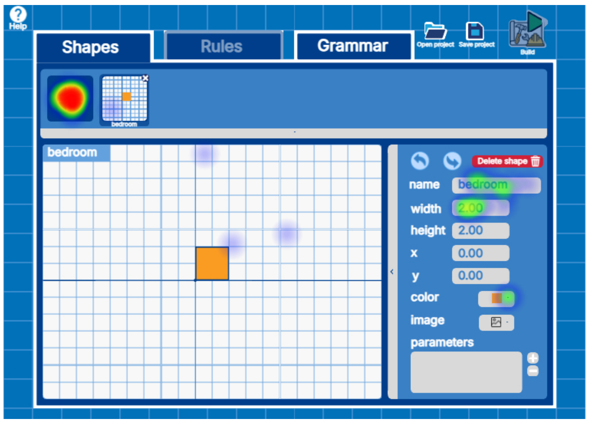
With that in mind, some of the observed misclick rates are attributable to such exploratory or non-sequential behavior. Still, a few screens revealed clearer patterns of interaction difficulty or misunderstanding, which are discussed below:

- Bedroom: Several participants initially attempted to click or drag directly on the grid to create a shape, instead of pressing the "Add Shape" button (Fig 5.13a).
- Bathroom: Some users misunderstood the task's intent and tried to edit the parameters of the previously created Bedroom shape instead of creating a new one, leading to repeated misclicks before identifying the correct control (Fig 5.13b).
- Suite: Apart from misclicks caused by drag attempts (which Maze occasionally registered as clicks), participants often tried to select the Suite as a whole by clicking its thumbnail, rather than by clicking outside the active shape as required (Fig 5.13c).
- Kitchen + Pantry: The most frequent misclicks were related to participants attempting to reposition the Pantry shape manually on the canvas instead of using the numeric X and Y parameters (Fig 5.13d).[2]
- Rule: Kitchen + Pantry: The misclick rate in this task was mainly due to scrolling issues within dropdown menus, which required highly specific click locations that Maze sometimes failed to register properly (Fig 5.13e).

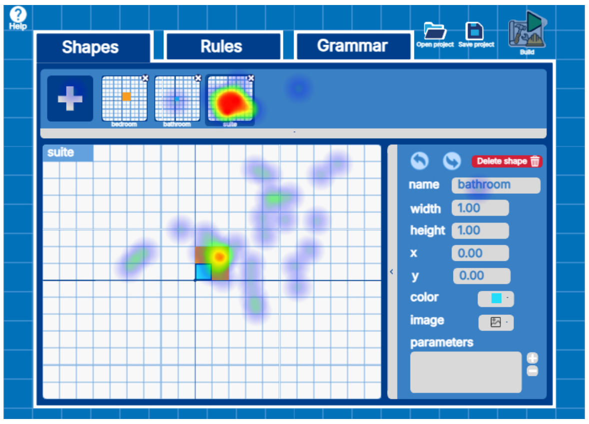
[2]The background screens visible in these heatmaps are low-resolution images automatically generated by Maze. As a result, certain interface elements may appear slightly distorted or misaligned. This is particularly noticeable in the Rule: Kitchen + Pantry heatmap, where text formatting artifacts cause overlapping buttons.These visual distortions do not affect the accuracy of the recorded click data.



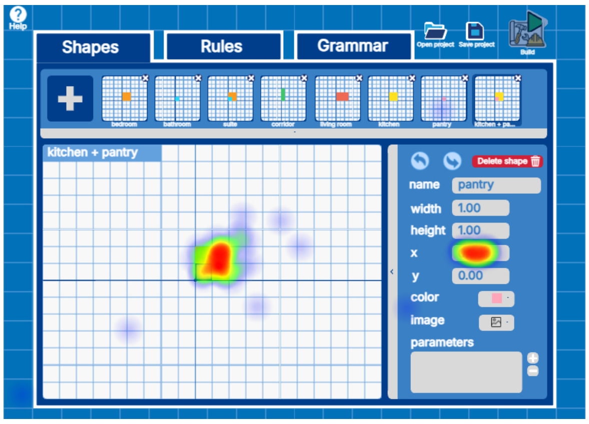
(A) Heatmap for the *Bedroom* task, showing concentrated clicks on the canvas area due to direct creation attempts.



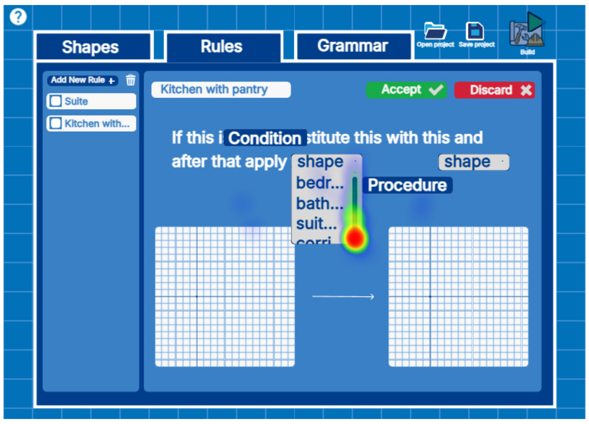
(B) Heatmap for the *Bathroom* task, highlighting misclicks from users editing the previous shape instead of creating a new one.



(C) Heatmap for the *Suite* task, with clustered clicks near thumbnails caused by incorrect selection attempts.



(D) Heatmap for the *Kitchen + Pantry* task, showing drag attempts on the canvas instead of parameter-based repositioning.



(E) Heatmap for the *Rule: Kitchen + Pantry* task, illustrating misclicks from dropdown scroll sensitivity.

FIGURE 5.13. Selected heatmaps highlighting interaction patterns associated with higher misclick rates. Red zones represent frequent click areas registered by Maze.

These observations highlight that most misclicks were not the result of confusion or poor usability, but of exploratory interaction, participants testing possible ways to achieve their goals or performing legitimate actions that happened outside Maze's strict success-tracking sequence. The heatmaps therefore provide valuable contextual evidence that complements the quantitative data, confirming that the high error rates observed in certain tasks stem from interface exploration and not from conceptual misunderstanding. This triangulation between quantitative rates, observational notes, and heatmap analysis strengthens the reliability of the evaluation, offering both numerical and behavioural validation of the results.

5.2.4. Qualitative Observations and Participant Feedback

In addition to quantitative metrics, qualitative insights were gathered through direct observation and notes taken throughout each evaluation session. These observations provided a deeper understanding of the participants' reasoning processes, interaction patterns, and expectations toward the interface and workflow. Overall, the feedback revealed a high level of engagement and curiosity, alongside several recurring suggestions for refinement across different stages of the workflow.

The tutorial stage was positively received but described as dense by some participants. Approximately 23% (7/30) found it contained too much information, while 30% (9/30) suggested that the tutorial could be made more interactive or include practical examples. A smaller group (4/30) reported difficulty understanding the concepts and difference between Rules and Procedures, indicating that these notions may benefit from progressive introduction or contextual reinforcement within later stages of the interface.

The Shapes environment generated the richest behavioral observations. A large majority of participants (23/30, 76.7%) expected to reposition shapes by directly dragging them across the grid, rather than relying solely on numerical parameter adjustments. Others suggested alternative interaction methods, such as using arrow-key controls (4/30, 13.3%) or clicking a target position on the canvas to place the shape (2/30, 6.7%). Although the prototype already supported vertex-based resizing, more than half of the participants (17/30, 56.7%) explicitly requested the ability to resize shapes by dragging edges as well, or by combining both approaches: vertices to maintain proportions and edges for free-form scaling. A smaller subset (2/30, 6.7%) also proposed the inclusion of a proportional locking shortcut, represented by a padlock icon, to help preserve aspect ratios during resizing.

When interacting with Composition shapes, participants consistently expressed a preference for selecting entire compositions through their thumbnails rather than by clicking outside the active shape, as the prototype required. A total of 20 out of 30 participants (66.7%) explicitly suggested adopting this method as the default interaction model, arguing that it felt more natural and aligned with standard practices in design software. This feedback, consistent with the quantitative findings discussed earlier, reinforces the intuitive expectation of direct manipulation typical of contemporary design tools and should be regarded as a necessary adjustment in future iterations of the system.

Only a few participants (2/30, 6.7%) failed to immediately navigate to the Rules Editor when prompted, suggesting that the tab-based navigation was generally well understood. Nonetheless, these same participants reiterated the need for more explicit guidance, both within the tutorial and inside the Rules Editor itself, to clarify the purpose and workflow of this stage.

The Blockly stage gathered substantial feedback, with 11 out of 30 participants (36.7%) requesting a dedicated tutorial for this section and 4 (13.3%) suggesting that such tutorial could be presented in video format. A total of 5 participants (16.7%) reported difficulty understanding the block logic, while 12 (40.0%) expressed uncertainty regarding where to access this feature, indicating that the Grammar tab label was not self-explanatory. Alternative names were proposed by 4 participants (13.3%), including "All", "Set up", "Index", "Workspace", and "Sandbox".

Participants also provided aesthetic and organizational feedback regarding the blocks themselves. Some noted that the colours of Rules and Procedures were too similar (3/30, 10.0%), while others suggested adopting a more consistent color-coding system across categories (5/30, 16.7%). The inclusion of titles or hierarchical categories within the Blockly menus was recommended by 7 participants (23.3%), and sorting blocks by complexity or usability was a common suggestion among 13 participants (43.3%). Additionally, 19 participants (63.3%) explicitly recommended placing the "Available Shapes/Rules" blocks at the top of the menu for easier access. Other minor suggestions included clearer syntax (e.g., adding colons or arrows for readability), simplified dropdown interactions, and enhanced visual consistency inspired by existing environments such as IntelliJ or Scratch.

Feedback in the Design Generation phase focused primarily on iconography and interaction flow. Two participants (6.7%) did not immediately click the "Build" button, while 4 (13.3%) considered its icon too similar to a settings symbol. Alternative suggestions included using a standalone "Play" icon (2/30, 6.7%) or changing its color to blue (3/30, 10.0%) to make it more prominent.

A total of 12 participants (40.0%) expressed confusion regarding the question about the Origin, suggesting that its wording should be simplified or that the color-coding of the options should be modified, since the red/green distinction implied a right–wrong dichotomy. A few users (2/30, 6.7%) proposed replacing the question altogether with a pop-up notification indicating that the origin was fixed at (0,0).

Several usability suggestions also emerged during this stage. Eight participants (26.7%) found the label "Shapes list" unclear and proposed alternatives such as "Shapes tab," "Previous list," or "Favourite shapes." Others (7/30, 23.3%) suggested adding clearer visual cues to emphasize that the grid in the design phase was interactive, allowing users to pan and zoom. Proposals included changing the cursor to a hand icon when hovering over the workspace and displaying a small zoom indicator (e.g., magnifier icon with percentage), similar to those used in web browsers.

Requests for more intuitive navigation were also noted, including arrow-key iteration between design solutions (2/30, 6.7%) and clearer export options, such as "Download" or "Save Design," mentioned by 13 participants (43.3%). Finally, a strong majority (25/30, 83.3%) preferred the "home" icon over the "arrow" one for exiting the design phase, confirming this as the definitive choice for the final implementation.

Following the prototype sessions, participants shared additional general suggestions aimed at improving accessibility and user experience. Multilingual support was requested by 4 participants (13.3%), while 2 (6.7%) recommended adding accessibility options for color-blind users. Five participants (16.7%) suggested incorporating more keyboard shortcuts (e.g., Ctrl+Z for undo or shortcuts for tab switching), and one (3.3%) proposed introducing greater color diversity across the interface to enhance visual appeal.

A small subset of participants (4/30, 13.3%) considered the overall visual design to appear somewhat childlike, whereas one participant (3.3%) suggested further developing the mascot's personality to strengthen emotional engagement with the user.

Overall, these qualitative observations reinforced the quantitative findings and validate the system's current design direction but also underscore specific opportunities for refinement for future work.

5.3. Summary of Results

The results of the summative evaluation demonstrate that SAGE successfully achieved its main objectives of usability, clarity, and accessibility, particularly among users with little or no prior experience in digital design or Shape Grammars. Through a combination of quantitative and qualitative data, the study confirmed that the system's interface and workflow effectively support both intuitive exploration and conceptual understanding of grammar-based design.

From a quantitative perspective, the overall SUS score of 78.08 places SAGE within the "Good to Excellent" usability range, surpassing the standard benchmark of 68. Complementary Likert-scale responses further reinforce this positive assessment, with high agreement on statements regarding enjoyment, visual appeal, and learning support. Correlational analysis revealed strong links between usability, enjoyment, and perceived understanding, suggesting that participants who found the interface more engaging also felt they learned Shape Grammar concepts more effectively.

Behavioural data collected through the Maze platform corroborated these perceptions. Task completion rates remained consistently high across all workflow phases, with an overall average success rate above 85%. Misclick analysis confirmed that most errors stemmed from exploratory actions rather than from confusion or design flaws.

Qualitative feedback further emphasized SAGE's intuitive character and educational potential. Participants quickly grasped the workflow of creating and combining shapes, defining rules, and applying procedures.

However, several patterns of improvement emerged:

- a strong expectation for direct manipulation through drag-and-drop and edge-based resizing in the Shapes tab;
- the need for more explicit differentiation between Rules and Procedures, ideally supported by progressive tutorials;
- clearer organization and color-coding of Blockly blocks, with frequently used items placed at the top;
- refined iconography and clearer labels (e.g., replacing “Build” and “Shapes list” with more intuitive alternatives).

Participants also suggested accessibility and usability enhancements such as keyboard shortcuts, multilingual support, and clearer zoom and navigation cues within the design environment. Despite these refinements, overall engagement and satisfaction remained high, with most users completing all tasks independently and expressing interest in using the tool again.

In summary, the evaluation confirmed that SAGE provides an effective and approachable environment for learning and applying Shape Grammar principles. Its visual, modular, and interactive design was perceived as both enjoyable and pedagogically valuable, bridging the gap between theoretical grammar formalism and practical design interaction. The identified improvement areas represent opportunities for future iterations rather than limitations of the current implementation.

CHAPTER 6

Conclusion

This chapter concludes the dissertation by revisiting its objectives, summarising the main findings, and reflecting on the implications of the developed system SAGE (ShApe Grammar assisted Environment). The work aimed to explore how Shape Grammars could be made more accessible through visual interaction and user-centred methodologies, bridging the gap between formal computational design theory and practical usability. The chapter also revisits the research questions, outlines the key contributions and limitations, and identifies directions for future work.

6.1. Overview

The motivation for this research stemmed from the recognised gap between the theoretical richness of Shape Grammars and their limited adoption in practice, largely due to the absence of intuitive and interactive tools. Addressing this gap, SAGE was conceived as a proof-of-concept application that integrates Shape Grammar logic, procedural reasoning, and User-Centred Design principles.

Following a DSR methodology, the project evolved through iterative design and evaluation cycles: from medium-fidelity prototypes in Figma, to a functional JavaFX desktop application that integrates Blockly and the Alternative Shaper engine. Each iteration was guided by feedback from users, ensuring that the final system aligned with both theoretical goals and practical usability needs.

6.2. Revisiting the Research Questions

The research was guided by three overarching questions, each addressing a specific dimension of the problem. Their answers are summarized below in light of the design, implementation, and evaluation outcomes.

RQ1. *Which graphical representations are most effective for supporting the visual design of Shape Grammars, particularly for novice users?*

SAGE demonstrated that separating the Shape Grammar workflow into three dedicated visual spaces (Shape Editor, Rule Editor, and Grammar Editor) is effective in supporting novice users. The combination of grid-based spatial composition, shape thumbnails, and explicit before/after rule previews helped participants reason about geometric transformations and grammar structure without requiring prior domain knowledge. The summative evaluation showed that users were able to identify shapes, create rule transformations, and understand grammar progression, despite most having only superficial awareness of Shape Grammars beforehand.

RQ2. *Which interaction metaphors best facilitate the intuitive creation, manipulation, and understanding of shapes, rules, and grammars within a visual environment?*

The evaluation showed that interaction metaphors drawn from familiar creative software were most effective. Tabbed navigation; grid-based canvases and parameter side-panels supported precise manipulation; and thumbnails enabled quick recognition and selection. Participants naturally attempted direct manipulation (drag-to-move, vertex-based resizing, canvas-first selection) before turning to form fields, reflecting expectations shaped by mainstream design tools. The block-based logic editor also helped translate procedural thinking into tangible, draggable units.

Crucially, the "experienced builder" character functioned as an on boarding and guidance metaphor: a domain-savvy companion that offers contextual prompts, step-wise instructions, and corrective hints. This narrative device operationalizes scaffolding[73] by chunking tasks, reducing uncertainty, and keeping attention on the design goal rather than on interface discovery.

Overall, the most effective metaphors were those aligning with users' prior interaction schemas (direct manipulation, staged workflows) and those that externalize guidance through a trusted, in-context "coach" (the builder), thereby lowering cognitive load and accelerating novice uptake.

RQ3. *How can iterative prototyping and user feedback inform and validate these representations and interaction models to enhance the accessibility, clarity, and usability of Shape Grammar-based design tools?*

The iterative development process guided by UCD and DSR proved essential to refining both the visual and interactive aspects of SAGE. Early Figma prototypes allowed the validation of interface layout and user flow, while later evaluations with the fully functional application provided concrete evidence of usability and learnability. Successive rounds of feedback directly informed design adjustments, such as the introduction of clearer iconography, improved thumbnails, and the reorganization of Blockly categories. The summative evaluation, with a mean SUS score of 78.08, validated that this iterative and feedback-driven process effectively enhanced clarity, usability, and overall user satisfaction.

6.3. Limitations

Although the results are promising, certain limitations constrain the current version of SAGE. First, the system depends on the Alternative Shaper Prolog engine, which restricts its independence and flexibility, as the grammar execution relies on an external inference module. Secondly, the Blockly integration currently functions only as a visual interface; it does not yet persist or execute user-defined procedures independently, limiting the extent to which users can create and test their own procedural logic within the environment. Moreover, the evaluation primarily focused on short-term interactions, providing valuable insights into initial usability but not addressing long-term learning effects or professional applicability. Finally, the participant sample, although diverse in disciplinary background, consisted solely of university students with no prior experience in Shape Grammars, which may affect the generalization of the findings.

Recognizing these limitations provides a solid foundation for targeted improvements in future iterations of SAGE and for subsequent research exploring its broader applicability and pedagogical impact.

6.4. Future Work

Building upon the limitations discussed in the previous section, several directions for future improvement and exploration have emerged from this research. A primary avenue for development concerns the technical independence of SAGE. Replacing the current dependency on the Alternative Shaper Prolog engine with an internally managed inference module would grant the system greater autonomy and flexibility, facilitating smoother integration between the interface and the underlying grammar execution.

Another key priority involves extending the Blockly integration to support persistent and executable logic, enabling users to define, save, and run custom procedures directly within the visual interface rather than relying solely on predefined logic. Further enhancements to direct manipulation, such as drag-and-drop positioning, edge-based resizing, and proportional scaling would also align the system with interaction patterns familiar from professional design tools. In parallel, refining the tutorial system to include contextual, example-based learning and step-by-step on boarding would make the tool more approachable to first-time users and strengthen its pedagogical value.

From a user experience and research perspective, future iterations of SAGE should also expand accessibility options, incorporating features such as keyboard shortcuts, color-blind modes, and multilingual support. Longitudinal studies could evaluate how sustained use of the tool affects users' conceptual understanding of Shape Grammars, while comparative experiments with traditional text-based grammar systems could help quantify learning gains and cognitive load differences more precisely. In addition, future evaluations should involve participants with prior experience in Shape Grammars, such as researchers, architects, or advanced design students to assess the tool's applicability and effectiveness in professional or expert-oriented contexts.

Collectively, these directions point toward a more mature and autonomous version of SAGE, capable not only of supporting interactive learning but also of serving as a research platform for studying the relationship between generative systems, pedagogy, and human–computer interaction.

6.5. Final Remarks

This dissertation demonstrated that the principles of Shape Grammars, often regarded as abstract and technically demanding, can be reinterpreted through accessible, user-centred design. The development of SAGE (ShApe Grammar assisted Environment) exemplifies how a rigorous computational framework can coexist with intuitive interaction design, bridging theoretical formalism and creative exploration.

The project's contributions extend beyond its technical implementation. By applying Design Science Research and Human–Computer Interaction principles, it established a replicable process for transforming formal generative systems into usable, pedagogical tools. The evaluation confirmed that even users without prior design or programming experience were able to understand and manipulate Shape Grammar concepts effectively, perceiving the tool as both enjoyable and educational.

In doing so, SAGE illustrates the broader potential of visual and interactive paradigms to democratize access to computational design. The findings suggest that when systems are designed with the user's cognitive processes in mind complex theories like Shape Grammars can evolve from academic constructs into practical design instruments.

Ultimately, SAGE stands as both a functional prototype and a methodological contribution: a demonstration that usability and formal expressiveness need not be opposites, but complementary forces in advancing generative design research.

References

[1] G. Stiny and J. Gips, "Shape grammars and the generative specification of painting and sculpture," in *Information Processing, Proceedings of IFIP Congress 1971, Volume 2 - Applications, Ljubljana, Yugoslavia, August 23-28, 1971*, C. V. Freiman, J. E. Griffith, and J. L. Rosenfeld, Eds., North-Holland, 1971, pp. 1460–1465.

[2] G. Stiny and W. J. Mitchell, *The Palladian Grammar*. Environment, Planning B: Planning, and Design, 1978, vol. 5, pp. 5–18. DOI: `10.1068/b050005`.

[3] U. Flemming, "More than the sum of parts: The grammar of queen anne houses," *Environment and Planning B: Planning and Design*, vol. 14, no. 3, pp. 323–350, 1987. DOI: `10.1068/b140323`.

[4] J. P. Duarte, "A discursive grammar for customizing mass housing: The case of siza's houses at malagueira," *Automation in Construction*, vol. 14, no. 2, pp. 265–275, 2005. DOI: `10.1016/j.autcon.2004.07.013`.

[5] T. W. Knight, "The generation of hepplewhite-style chair-back designs," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 1980.

[6] M. Agarwal and J. Cagan, "A blend of different brews: The application of shape grammars to industrial design," *Design Studies*, vol. 19, no. 1, pp. 91–111, 1998. DOI: `10.1016/S0142-694X(97)00002-8`.

[7] J. McCormack, M. Cagan, and K. Kotovsky, "Evaluating shape grammars and brand identity in automotive design," in *Design Computing and Cognition 2004*, J. S. Gero, Ed., Dordrecht: Kluwer Academic Publishers, 2004, pp. 583–600, ISBN: 978-1-4020-2408-2.

[8] J. Gips and G. Stiny, "Production systems and grammars: A uniform characterization," *Environment and Planning B: Planning and Design*, vol. 7, pp. 399–408, 1980. DOI: `10.1068/b070399`. [Online]. Available: `https://doi.org/10.1068/b070399`.

[9] H.-W. Chau, J. Cagan, and K. Kotovsky, "Designing consumer packaging using shape grammars: A case study of bottled water," in *Design Computing and Cognition 2004*, J. S. Gero, Ed., Dordrecht: Kluwer Academic Publishers, 2004, pp. 581–599, ISBN: 978-1-4020-2408-2.

[10] R. C. Correia and R. Coutinho, "Designa - a shape grammar interpreter," M.S. thesis, Tecnico Lisboa, 2013.

[11] J. Tching, J. Reis, and A. Paio, "IM-sgi: An Interface Model for Shape Grammar Implementations," M.S. thesis, 2019. [Online]. Available: `https://www.cambridge.org/core/product/identifier/S0890060417000695/type/journal_article`.

[12] T. Yazar and B. Çolakoğlu, "Qshaper: A cad utility for shape grammars," Sep. 2007.

[13] X. Shi, Y. Wang, R. Rossi, and J. Zhao, “Brickify: Enabling expressive design intent specification through direct manipulation on design tokens,” in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, ACM, Apr. 2025, pp. 1–20.

[14] F. Santos, K. Kwiecinski, A. de Almeida, S. Eloy, and B. Taborda, “Alternative shaper: A model for automatic design generation,” *Formal Aspects of Computing*, vol. 30, no. 3-4, pp. 333–349, 2018. DOI: `10.1007/s00165-018-0461-z`.

[15] Google, *Blockly: Visual programming library*, `https://developers.google.com/blockly`, "Accessed on June 20, 2025", 2025.

[16] J. vom Brocke, A. Hevner, and A. Maedche, “Introduction to design science research,” in Sep. 2020, pp. 1–13.

[17] B. Shneiderman and C. Plaisant, *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Pearson Addison Wesley, 2007.

[18] Y. Rogers, H. Sharp, and J. Preece, *Interaction Design: Beyond Human-Computer Interaction.* Wiley, 2011.

[19] L. K. Narayan and M. K. Rao, *Computer Aided Design and Manufacturing.* Delhi, India: PHI Learning, Dec. 2008.

[20] Autodesk, *AutoCAD*, `https://www.autodesk.com/products/autocad/`, Version 2024, Computer software, Accessed on June 20, 2025, 2024.

[21] Y. E. Kalay, *Architecture's New Media: Principles, Theories, and Methods of Computer-Aided Design.* Cambridge, MA: MIT Press, 2004, ISBN: 978-0262112856.

[22] L. Sass and R. Oxman, “Materializing design: The implications of digital fabrication in architecture,” *Design Studies*, vol. 27, no. 3, pp. 324–355, 2006. DOI: `10.1016/j.destud.2005.11.002`.

[23] J. M. Carroll, *Human-Computer Interaction: Psychology as a Science of Design.* Annual Review of Psychology, 1997.

[24] Robert McNeel & Associates, *Grasshopper 3D – Algorithmic Modeling for Rhino*, `https://www.grasshopper3d.com/`, Computer software, Accessed on June 20, 2025, 2024.

[25] R. Woodbury, *Elements of Parametric Design.* New York: Routledge, 2010, ISBN: 978-0415779871.

[26] M. A. Boden, “Creativity and artificial intelligence,” *Artificial Intelligence*, vol. 103, no. 1–2, pp. 347–356, 1998. DOI: `10.1016/S0004-3702(98)00055-1`.

[27] M. Whitelaw, *Metacreation: Art and Artificial Life.* Cambridge, MA: MIT Press, 2004, ISBN: 978-0262232325.

[28] M. Gardner, “Mathematical games: The fantastic combination of john conway's new solitaire game “life”,” *Scientific American*, pp. 120–123, 1970.

[29] A. Lindenmayer, “Mathematical models for cellular interaction in development,” *Journal of Theoretical Biology*, vol. 18, pp. 280–315, 1968.

[30] C. W. Reynolds, “Flocks, herds, and schools: A distributed behavioral model,” *Computer Graphics (Proceedings of SIGGRAPH ’87)*, vol. 21, no. 4, pp. 25–34, 1987.

[31] Maxis, *Spore [computer game]*, `https://www.ea.com/games/spore`, Published by Electronic Arts, Platforms: Windows, macOS, Accessed on June 15, 2025, 2008.

[32] Y. I. H. Parish and P. Müller, “Procedural modeling of cities,” in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 2001)*, 2001, pp. 301–308. DOI: `10.1145/383259.383292`.

[33] J. McCormack, M. d’Inverno, O. Bown, A. Dorin, M. Boden, and M. A. Boden, “Computational creativity and cognition: The confluence of generative art, design, and science,” *Leonardo*, vol. 45, no. 3, pp. 285–289, 2012. DOI: `10.1162/LEON_a_00373`.

[34] J. McCormack and M. d’Inverno, *Computers and Creativity.* Berlin, Heidelberg: Springer, 2014. DOI: `10.1007/978-3-642-31727-9`.

[35] M. Hassenzahl, “The thing and i: Understanding the relationship between user and product,” in *Funology 2: From Usability to Enjoyment*, K. Hornbæk et al., Eds., Springer, 2018, pp. 301–313.

[36] P. Gabštur, M. Pollák, M. Kočiško, and V. Kotrady, “Implementation of generative design tools in the construction process.,” *TEM Journal*, vol. 14, no. 2, pp. 983–991, 2025, ISSN: 2217-8309. [Online]. Available: `https://research.ebsco.com/linkprocessor/plink?id=4d465fac-a526-3aa8-8ef5-3d2b1f032ddc`.

[37] R. Oxman, “Thinking difference: Theories and models of parametric design thinking,” *Design Studies*, vol. 52, pp. 4–39, 2017. DOI: `10.1016/j.destud.2017.06.001`.

[38] T. Dartnall, *Artificial Intelligence and Creativity: An Interdisciplinary Approach.* Dordrecht: Springer Netherlands, 1994. DOI: `10.1007/978-94-017-2245-7`.

[39] D. Davis, “Modelled on software engineering: Flexible parametric models in the practice of architecture,” in *Proceedings of the 31st International Conference on Education and Research in Computer Aided Architectural Design in Europe (eCAADe 2013)*, 2013, pp. 173–182.

[40] K. Terzidis, *Algorithmic Architecture.* Oxford: Routledge, 2006, ISBN: 978-0750667258.

[41] SideFX, *Houdini 20.0: 3d procedural software*, `https://www.sidefx.com`, 2023.

[42] Autodesk, Inc. “Generative design in fusion 360.” Accessed on June 20, 2025. [Online]. Available: `https://www.autodesk.com/solutions/generative-design`.

[43] G. Celani and L. Vaz, “Cad tools for creative exploration: Extending the possibilities of cad systems for conceptual design,” *International Journal of Architectural Computing*, vol. 10, no. 1, pp. 83–100, 2012. DOI: `10.1260/1478-0771.10.1.83`.

[44] N. Gu and P. Amini Behbahani, “Shape grammars: A key generative design algorithm,” in *Handbook of the Mathematics of the Arts and Sciences*, B. Sriraman, Ed. Cham: Springer International Publishing, 2021, pp. 1385–1405. DOI: `10.1007/978-3-319-57072-3_7`. [Online]. Available: `https://doi.org/10.1007/978-3-319-57072-3_7`.

[45] G. Stiny, "Introduction to shape and shape grammars," *Environment and Planning B*, vol. 7, no. 3, pp. 343–351, 1980.

[46] N. Chomsky, *Syntactic Structures.* The Hague: Mouton, 1957, ISBN: 978-9027933853.

[47] E. L. Post, "Formal reductions of the general combinatorial decision problem," *American Journal of Mathematics*, vol. 65, no. 2, pp. 197–215, 1943. DOI: 10.2307/2371809.

[48] R. Stouffs and N. Krishnamurti, "Computational design synthesis: An approach to shape grammars," *Design Studies*, vol. 15, no. 4, pp. 445–464, 1994. DOI: 10.1016/0142-694X(94)90018-3.

[49] T. Knight, "Shape grammars in education and practice: History and prospects," *International Journal of Design Computing*, vol. 2, 2000.

[50] M. Tapia, "A visual implementation of a shape grammar system," Ph.D. dissertation, Massachusetts Institute of Technology, 1996.

[51] J. Heisserman, "Genesis: A system for generative design exploration," in *Proceedings of the Association for Computer-Aided Design in Architecture (ACADIA 1992)*, 1992, pp. 143–152.

[52] S. Orsborn, J. Cagan, K. Kotovsky, and M. McCormack, "Creating cross-over vehicles: Defining and combining vehicle brand identity through shape grammars," in *Design Computing and Cognition 2006*, J. S. Gero, Ed., Dordrecht: Springer, 2006, pp. 55–74, ISBN: 978-1-4020-5130-9. DOI: 10.1007/978-1-4020-5131-6_4.

[53] J. P. Duarte, G. Oliveira, J. Morgado, and R. C. Correia, "A shape grammar for the urban morphology of the medina of marrakech," in *Proceedings of the 25th eCAADe Conference*, Frankfurt am Main: eCAADe, 2007, pp. 239–246.

[54] R. Krishnamurti, "The construction of shapes," *Environment and Planning B: Planning and Design*, vol. 9, no. 1, pp. 115–134, 1982. DOI: 10.1068/b090115.

[55] R. Krishnamurti, "Generic representations of shapes," *Environment and Planning B: Planning and Design*, vol. 19, no. 5, pp. 581–599, 1992. DOI: 10.1068/b190581.

[56] R. Stouffs and R. Krishnamurti, "Computational design synthesis: Reasoning about representations," in *Proceedings of the 3rd Design Computing Conference*, Pittsburgh, PA, 1994, pp. 107–123.

[57] J. P. Duarte and A. Simondetti, "Applying shape grammars to design problems," in *Proceedings of the 15th eCAADe Conference*, Vienna, Austria: eCAADe, 1997, pp. 611–616.

[58] M. Prats, N. Sass, E. Do, M. Gross, and T. Knight, "The role of shape rules in architectural design: A visual computing approach," *Automation in Construction*, vol. 15, no. 2, pp. 227–240, 2006. DOI: 10.1016/j.autcon.2005.06.007.

[59] H.-W. Chau, J. Cagan, and K. Kotovsky, "Implementing shape grammars in design: A rule-based shape recognition system," in *Proceedings of the 1999 ASME Design Engineering Technical Conferences*, ASME, Las Vegas, NV, 1999.

[60] T. Grasl and A. Economou, "From topologies to shapes: Parametric shape grammars implemented with graphs," in *Proceedings of the 31st eCAADe Conference*, Delft, The Netherlands: eCAADe, 2013, pp. 173–182.

[61] Esri, *Esri cityengine*, `https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview`, Version 2024. Rule-based 3D modeling software for urban environments, 2024.

[62] Unity Technologies, *Unity [game engine]*, `https://unity.com/`, Accessed on June 15, 2025, 2005.

[63] YoYo Games, *GameMaker [game engine]*, `https://gamemaker.io/`, Accessed on June 15, 2025, 1999.

[64] B. Foundation, *Blender 4.5 lts manual*, `https://docs.blender.org/manual/en/latest/`, Accessed on June 15, 2025, 2025.

[65] MIT Media Lab, *Scratch: Programming for everyone*, `https://scratch.mit.edu`, "Accessed on June 20, 2025", 2025.

[66] Figma, Inc., *Figma: Collaborative interface design tool*, `https://www.figma.com`, Accessed on March 30, 2025, 2025.

[67] J. Nielsen, "Why you only need to test with 5 users," *NNGroup*, 2000. [Online]. Available: `https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`.

[68] D. A. Norman, *The Design of Everyday Things*. Basic books, 2013.

[69] A. Dix, J. Finlay, G. D. Abowd, and R. Beale, *Human-Computer Interaction*, 3rd. Pearson Education, 2009.

[70] J. Nielsen, *Usability Engineering*. Morgan Kaufmann, 1993.

[71] M. Corporation, *Microsoft word*, Desktop version used to illustrate draggable vertices in shapes, 2021.

[72] J. Brooke, "Sus: A "quick and dirty" usability scale," in *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, B. Weerdmeester, and I. L. McClelland, Eds., Taylor & Francis, 1996, pp. 189–194.

[73] T. Catarci, L. D. Giovanni, S. Gabrielli, S. Kimani, and V. Mirabella, "Scaffolding the design of accessible elearning content: A user-centred approach and cognitive perspective," *Cognitive Processing*, vol. 9, no. 3, pp. 209–216, 2008. DOI: `10.1007/s10339-008-0213-3`.

APPENDIX A

Snapshots of Prototype 1

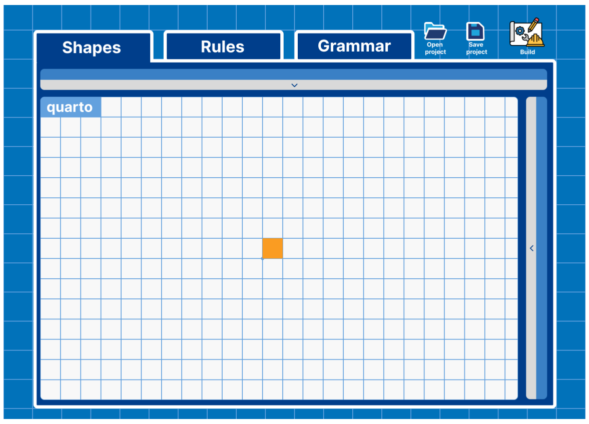


FIGURE A.1. Expanded workspace view in the first SAGE prototype, showing the shape "quarto" and illustrating the adjustable grid area relative to the thumbnail panel and shape properties.

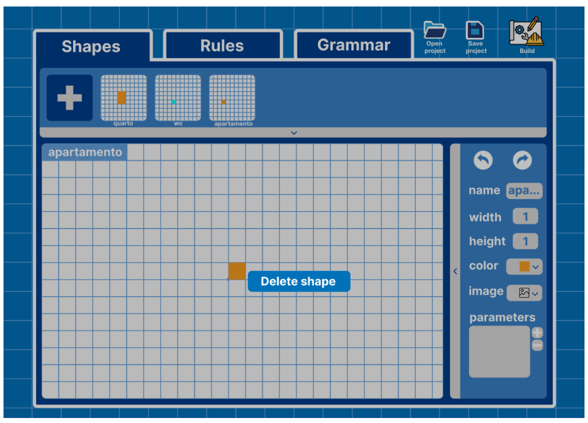


FIGURE A.2. View of the first SAGE prototype showing the default shape and the confirmation popup required to delete it before defining the new composition shape "apartamento".

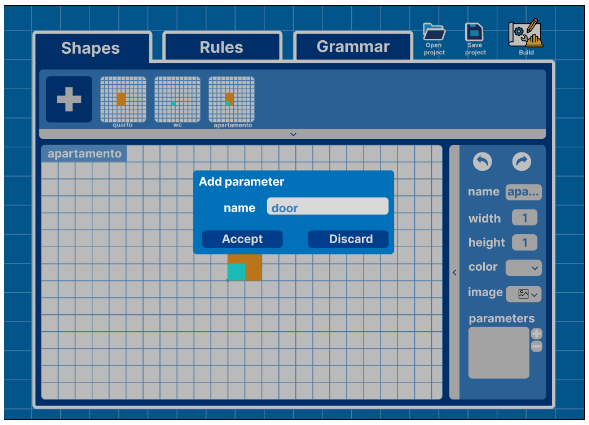


FIGURE A.3. Interface of the first SAGE prototype showing the parameter creation popup for the shape "apartamento".

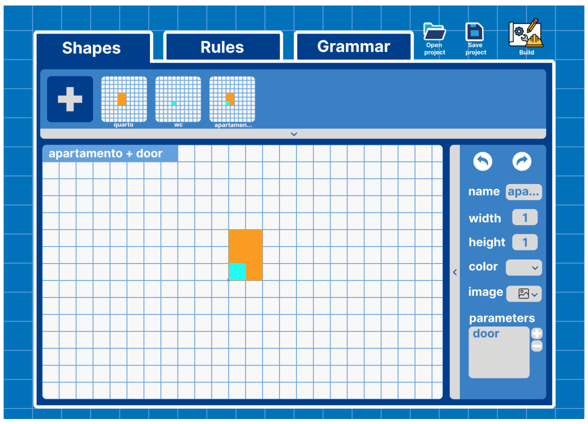


FIGURE A.4. Display of all shapes created by participants during the testing phase of the first SAGE prototype.

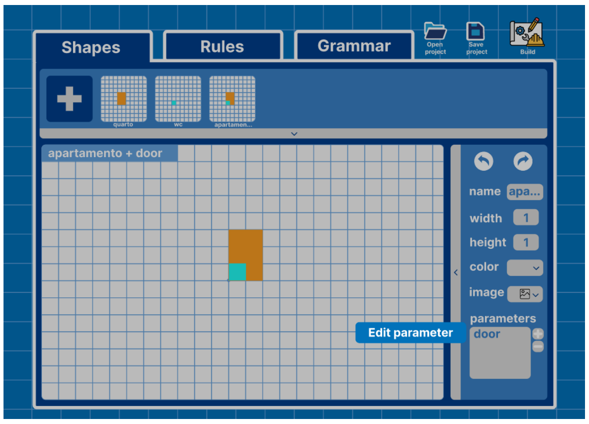


FIGURE A.5. Initial popup displayed before composing the parameter "door" within the first SAGE prototype.

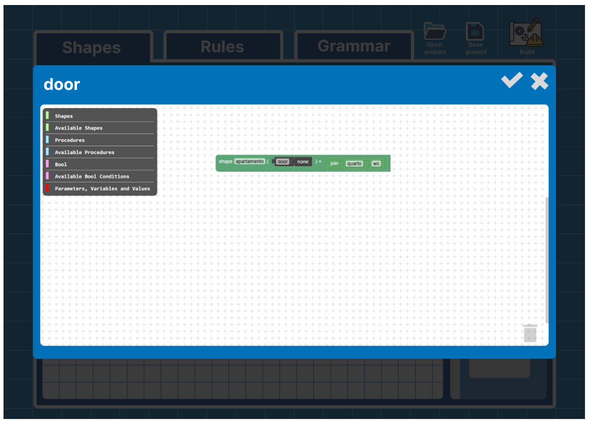


FIGURE A.6. Popup window displayed in the first SAGE prototype for defining the parameter "door" of the shape "apartamento".

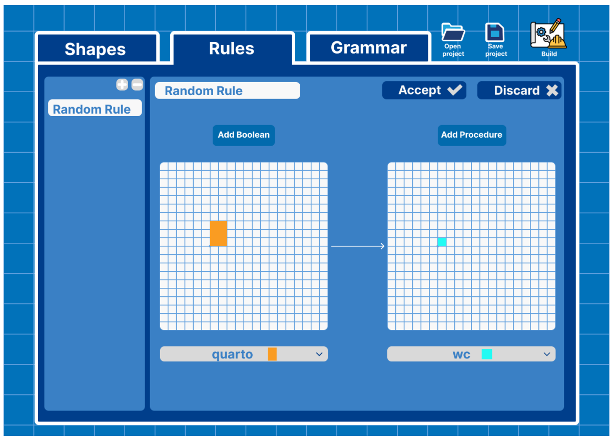
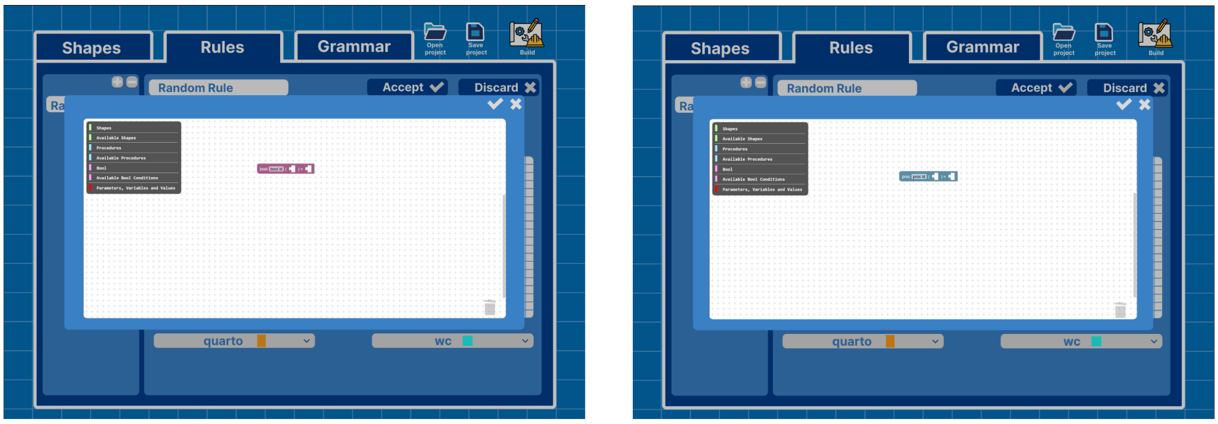


FIGURE A.7. Display of the rule defined by participants during the evaluation of the first SAGE prototype.



(A) Popup for editing the rule condition in the first SAGE prototype.

(B) Popup for defining the rule procedure in the first SAGE prototype.

FIGURE A.8. Rule Editor popups in the first SAGE prototype, showing the interfaces for defining conditional logic (a) and procedural behavior (b).

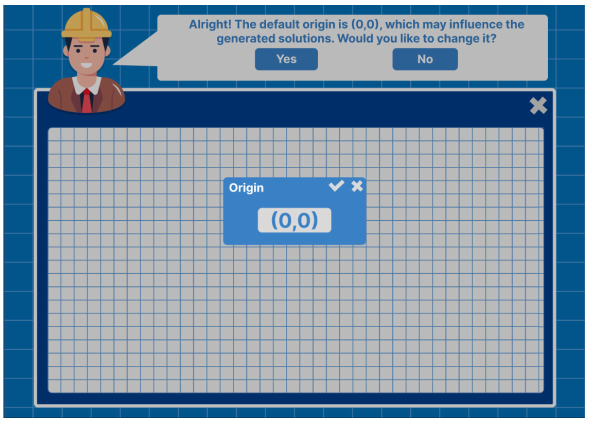

Alright! The default origin is (0,0), which may influence the generated solutions. Would you like to change it?

FIGURE A.9. Window displayed in the first SAGE prototype during the design generation phase, prompting the user to adjust the origin point of the generated layout before execution.

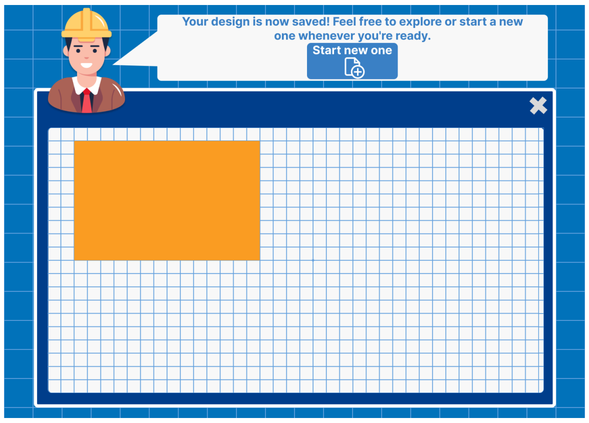

Your design is now saved! Feel free to explore or start a new one whenever you're ready.

FIGURE A.10. Final dialog in the design generation phase of the first SAGE prototype.

APPENDIX B

Design Options of Prototype 1

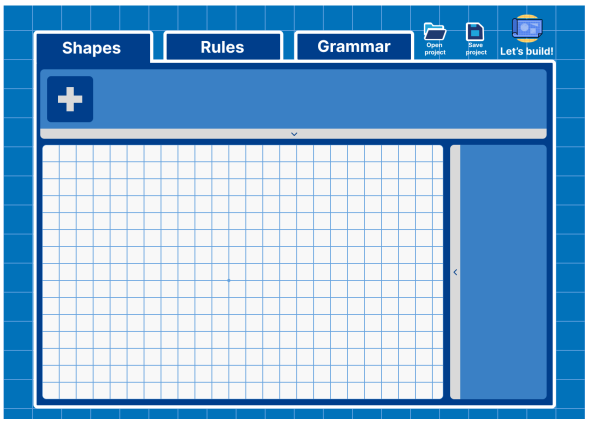


FIGURE B.1. Prototype 1 – Alternative "Build" icon (Option 1).

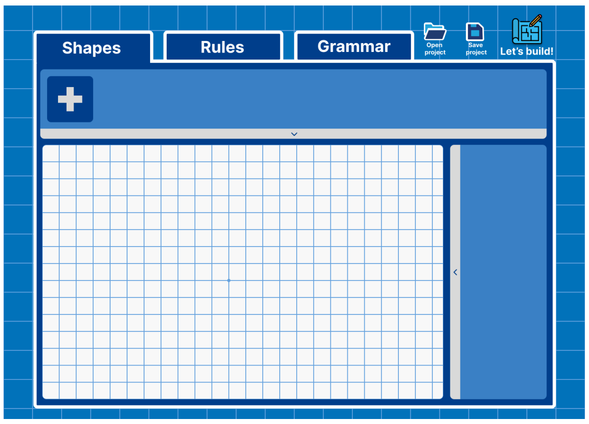


FIGURE B.2. Prototype 1 – Alternative "Build" icon (Option 2).

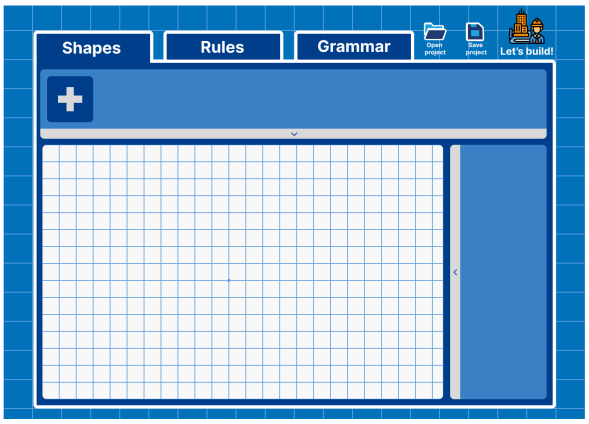


FIGURE B.3. Prototype 1 – Alternative "Build" icon (Option 3).

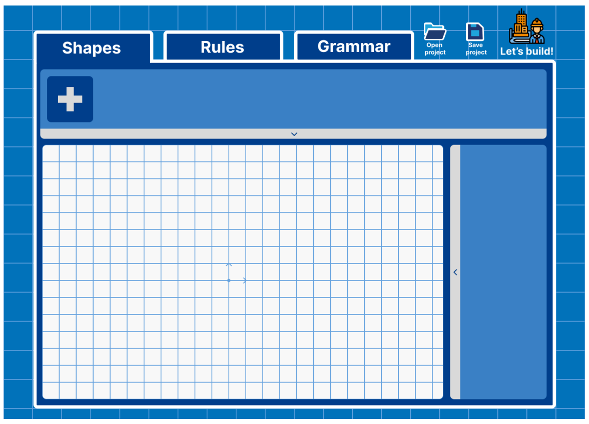


FIGURE B.4. Prototype 1 – Shape Editor showing a unitary origin indicator.

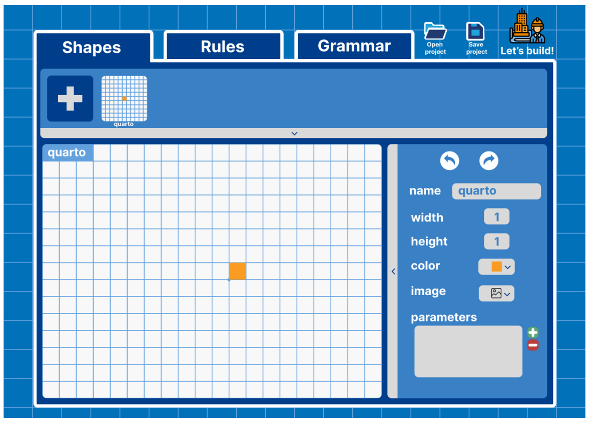


FIGURE B.5. Prototype 1 – Shape Editor showing the parameter buttons "+" and "–" with color coding.

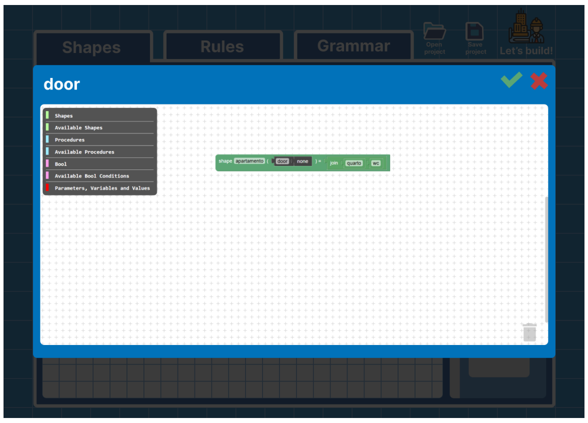


FIGURE B.6. Prototype 1 – Parameter Editor showing the "Accept" and "Discard" buttons with color coding.

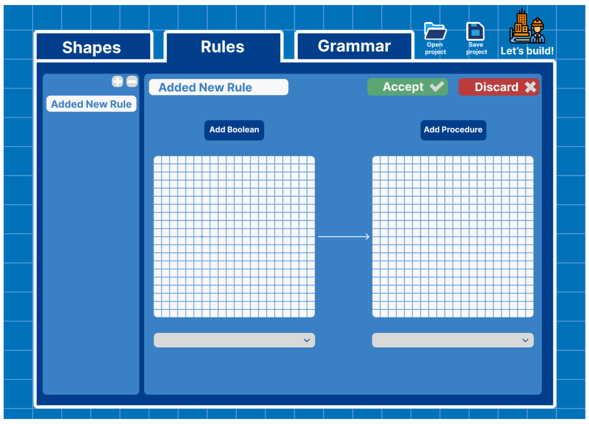


FIGURE B.7. Prototype 1 – Rule Editor showing the "Accept" and "Discard" buttons with color coding.

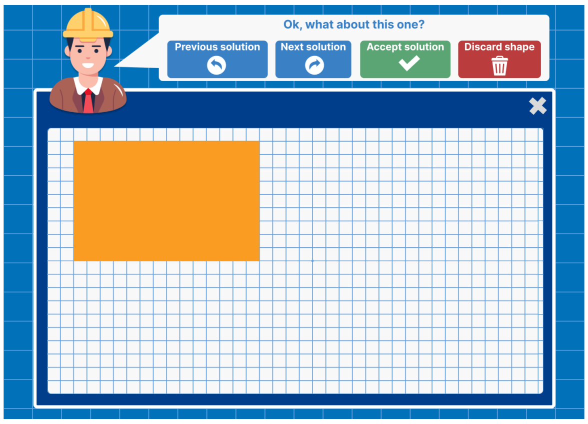


FIGURE B.8. Prototype 1 – Design Generation phase showing the "Accept" and "Discard" solution buttons with color coding.

APPENDIX C

Snapshots of Prototype 2

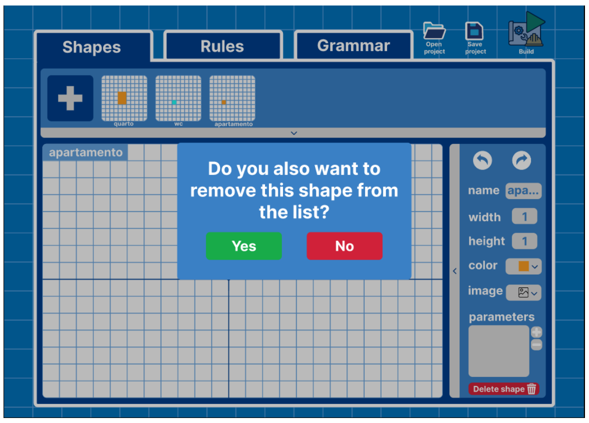


FIGURE C.1. Snapshot of the confirmation popup introduced in the second SAGE prototype, shown after clicking the "Delete Shape" button in the properties panel of a default shape.

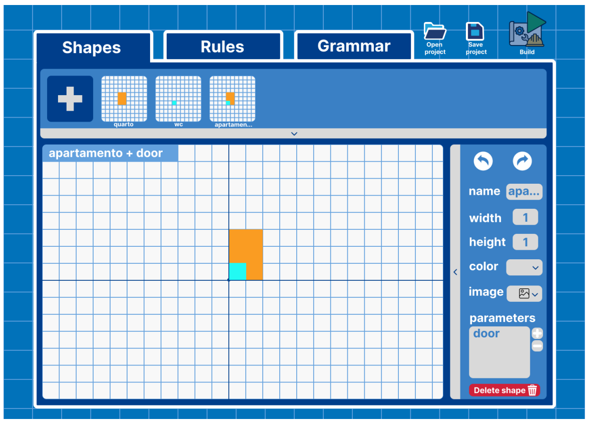


FIGURE C.2. Display of all shapes created by participants during the testing phase of the second SAGE prototype.

APPENDIX D

Snapshots of Prototype 3

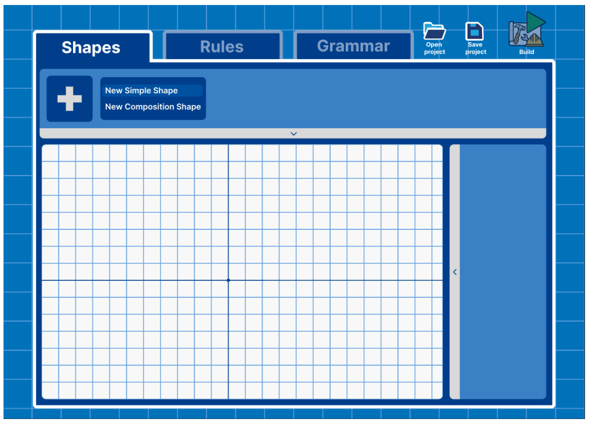


FIGURE D.1. Snapshot of the new Add Shape interaction introduced in the third SAGE prototype, showing the "Add Simple Shape" option highlighted as the user hovers over it.

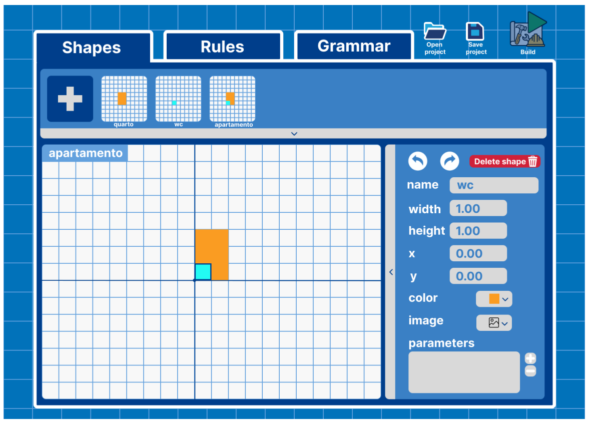


FIGURE D.2. Snapshot of the Shape Editor in the third SAGE prototype, showing the shape "wc" selected within the composition shape "apartamento".

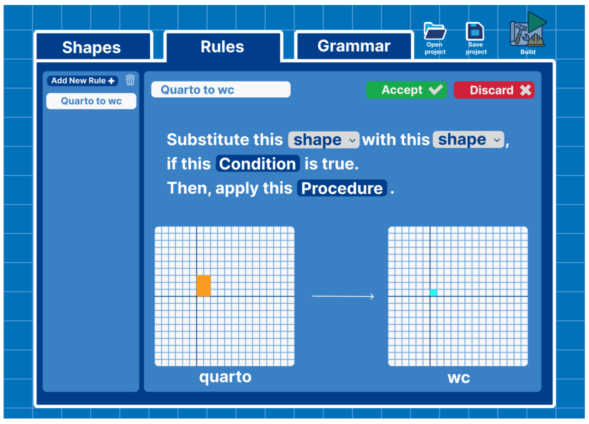


FIGURE D.3. Snapshot of a rule created by participants during the third SAGE prototype evaluation.

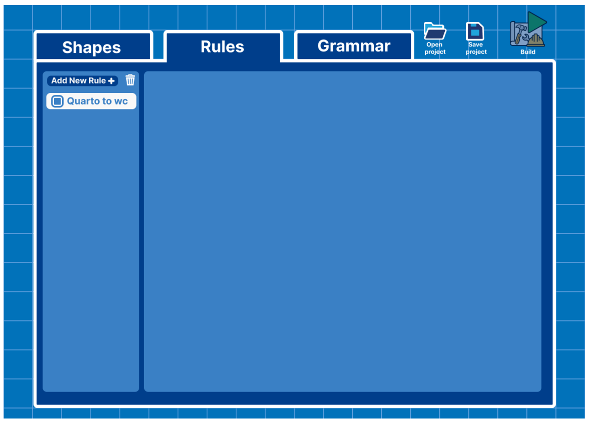
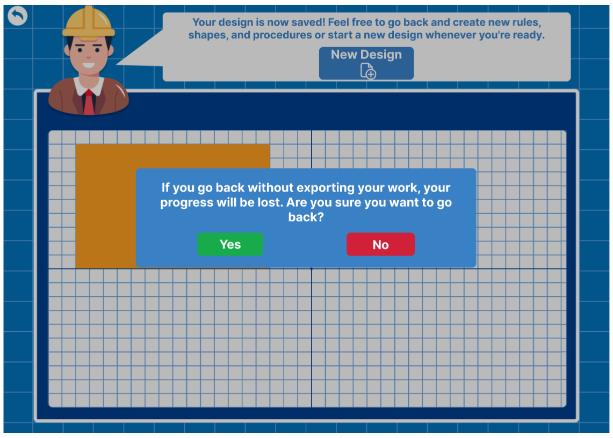


FIGURE D.4. Snapshot of the Rule Editor in the third SAGE prototype, showing the rule "quarto to wc" selected and the trash icon now available to allow deletion.

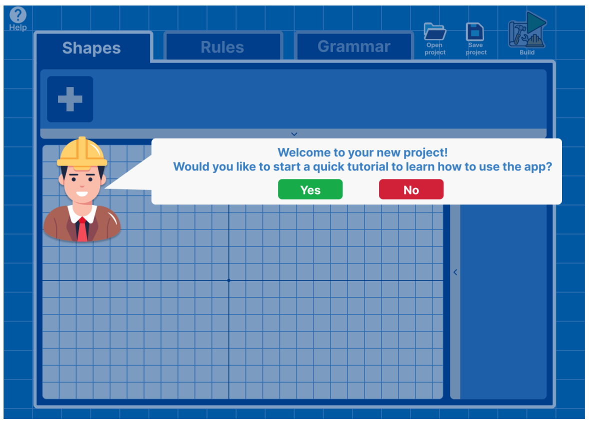

Your design is now saved! Feel free to go back and create new rules, shapes, and procedures or start a new design whenever you're ready.

If you go back without exporting your work, your progress will be lost. Are you sure you want to go back?

FIGURE D.5. Popup displayed in the third SAGE prototype asking the user to confirm whether they wish to exit the design generation phase.

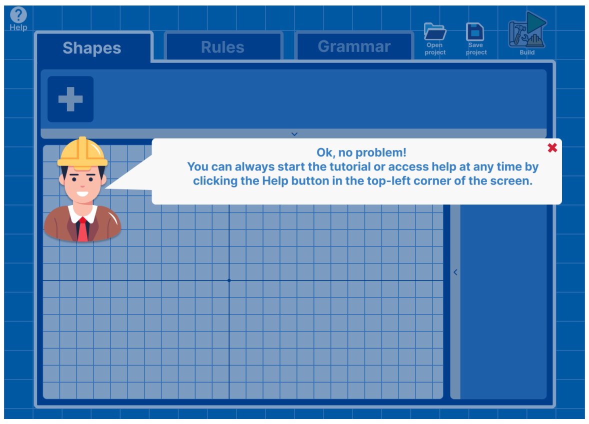
APPENDIX E

Snapshots of Final Prototype simulating SAGE



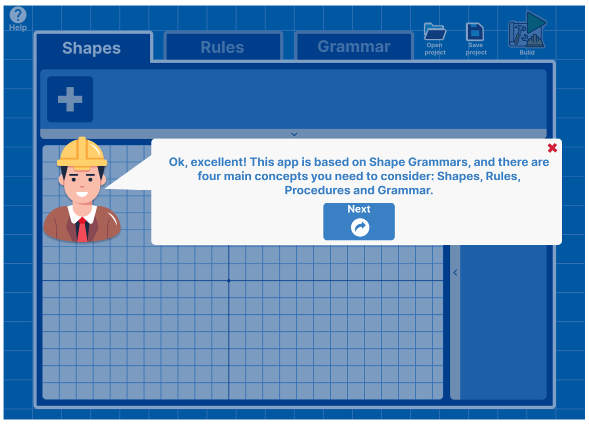
Welcome to your new project!
Would you like to start a quick tutorial to learn how to use the app?

FIGURE E.1. Snapshot of the first tutorial dialog introduced in the final SAGE prototype.



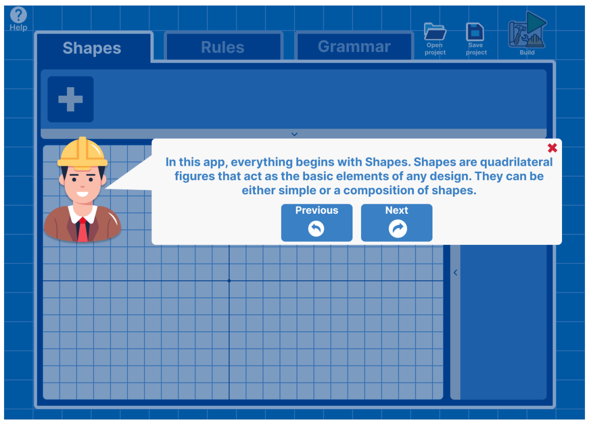
Ok, no problem!
You can always start the tutorial or access help at any time by clicking the Help button in the top-left corner of the screen.

FIGURE E.2. Snapshot of the dialog displayed when the user chooses not to follow the tutorial in the final SAGE prototype.



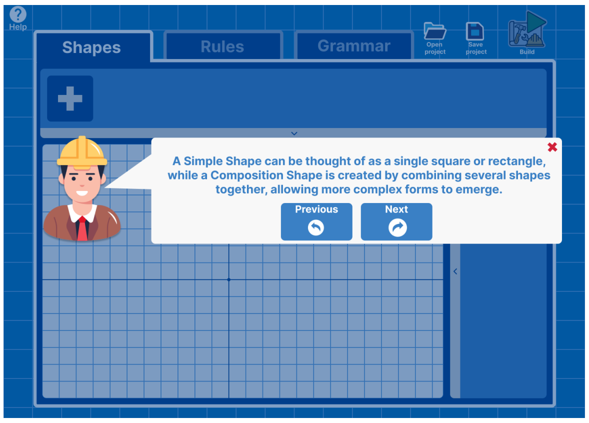
Ok, excellent! This app is based on Shape Grammars, and there are four main concepts you need to consider: Shapes, Rules, Procedures and Grammar.

FIGURE E.3. Snapshot of the dialog displayed when the user chooses to start the tutorial in the final SAGE prototype.



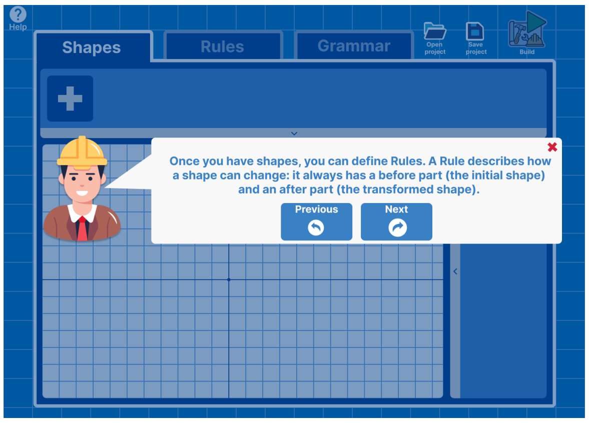
In this app, everything begins with Shapes. Shapes are quadrilateral figures that act as the basic elements of any design. They can be either simple or a composition of shapes.

FIGURE E.4. Snapshot of the tutorial dialog in the final SAGE prototype introducing the concept of shapes and their function in the system.



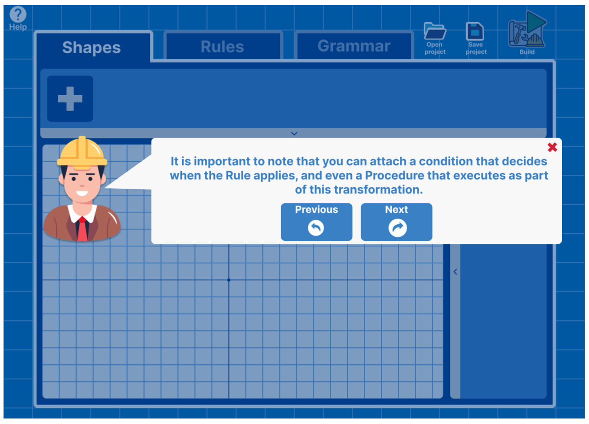
A Simple Shape can be thought of as a single square or rectangle, while a Composition Shape is created by combining several shapes together, allowing more complex forms to emerge.

FIGURE E.5. Snapshot of the final SAGE prototype tutorial, detailing the differences between simple and composition shapes.



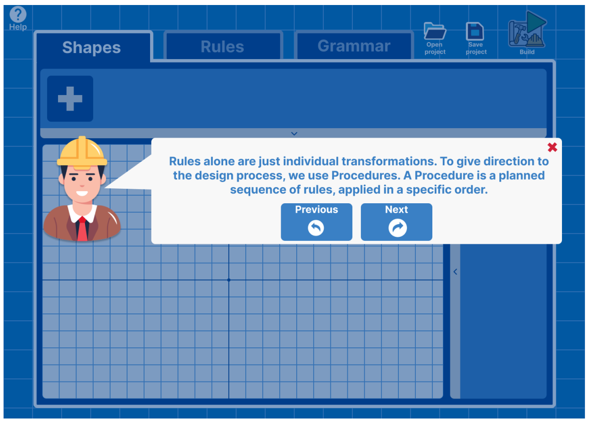
Once you have shapes, you can define Rules. A Rule describes how a shape can change: it always has a before part (the initial shape) and an after part (the transformed shape).

FIGURE E.6. Snapshot of the tutorial in the final SAGE prototype introducing the concept of Rules.



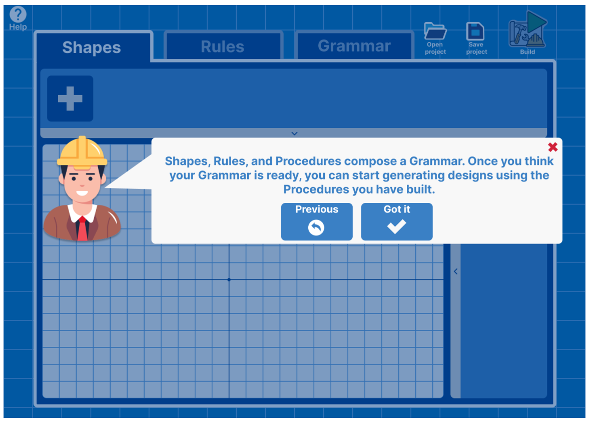
It is important to note that you can attach a condition that decides when the Rule applies, and even a Procedure that executes as part of this transformation.

FIGURE E.7. Snapshot of the tutorial in the final SAGE prototype, detailing how Rules may include Conditions and Procedures that extend their behavior.



Rules alone are just individual transformations. To give direction to the design process, we use Procedures. A Procedure is a planned sequence of rules, applied in a specific order.

FIGURE E.8. Snapshot of the tutorial in the final SAGE prototype, introducing the concept of Procedures.



Shapes, Rules, and Procedures compose a Grammar. Once you think your Grammar is ready, you can start generating designs using the Procedures you have built.

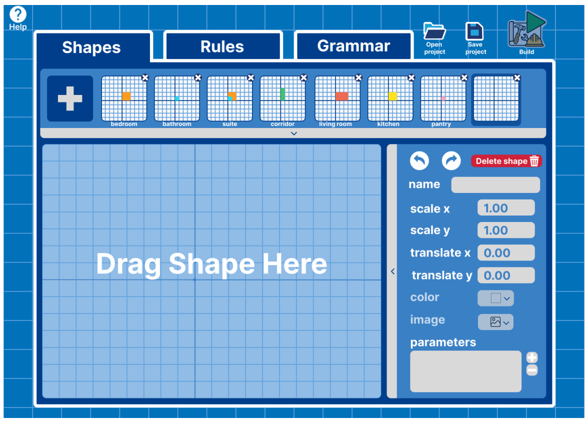
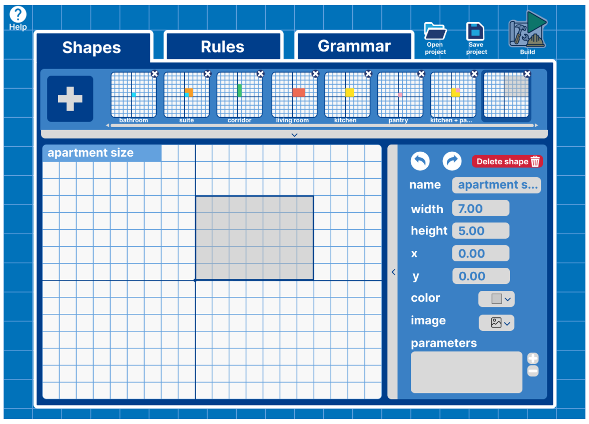
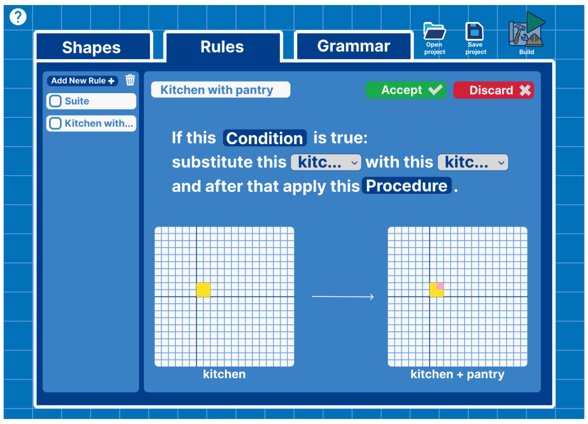


FIGURE E.9. Snapshot of the concluding tutorial dialog in the final SAGE prototype, summarizing how shapes, rules, and procedures together compose a grammar.

FIGURE E.10. Snapshot of the Shape Editor in the final SAGE prototype, illustrating the process of creating a new composition shape with previously defined Shapes.

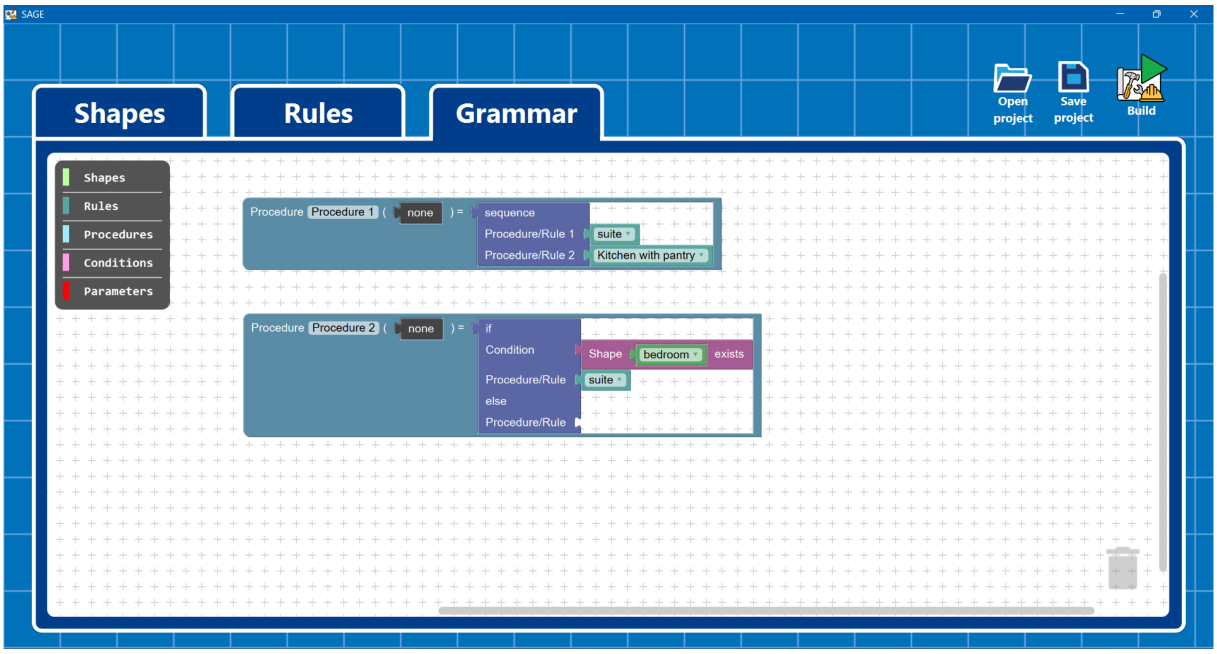




If this Condition is true:
substitute this kitc... with this kitc...
and after that apply this Procedure .

FIGURE E.11. Overview of all shapes generated during the final SAGE prototype evaluation.

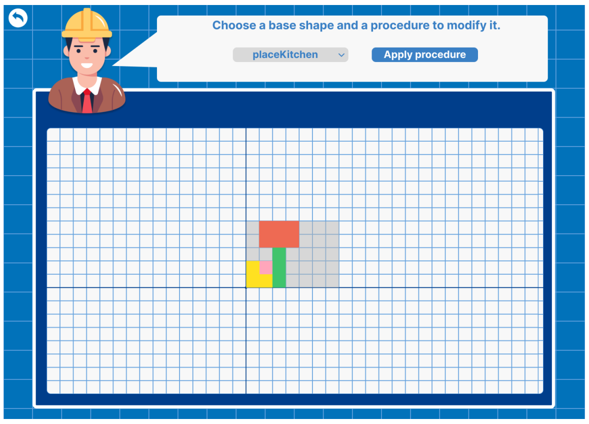
FIGURE E.12. Overview of the Rules produced by participants during the final SAGE prototype evaluation.



Procedure Procedure 1 (none) = sequence
Procedure/Rule 1 suite
Procedure/Rule 2 Kitchen with pantry

Procedure Procedure 2 (none) = if
Condition Shape bedroom exists
Procedure/Rule suite
else
Procedure/Rule

FIGURE E.13. Snapshot of the intended procedures created by participants during the final SAGE prototype evaluation.



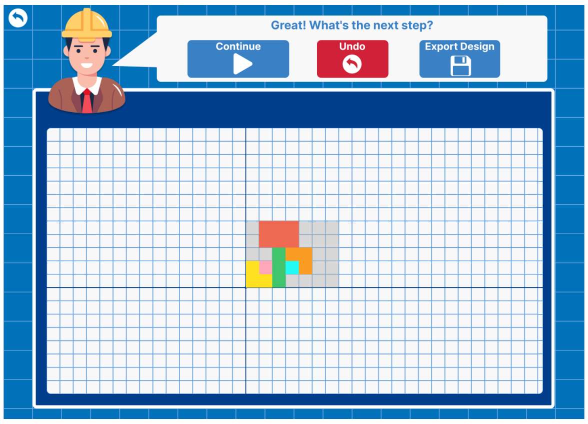


FIGURE E.14. Snapshot of the design generation phase in the final SAGE prototype, illustrating the intermediate step where the user chooses the procedure to continue the generative sequence.

FIGURE E.15. Snapshot of the design generation process in the final SAGE prototype, showing the "Next Step" stage that precedes the completion of the design.

APPENDIX F

SUS Results

Participant	SUS Score
1	82.5
2	62.5
3	80.0
4	45.0
5	77.5
6	90.0
7	95.0
8	95.0
9	62.5
10	70.0
11	72.5
12	62.5
13	80.0
14	75.0
15	75.0
16	75.0
17	87.5
18	95.0
19	92.5
20	92.5
21	75.0
22	82.5
23	85.0
24	75.0
25	77.5
26	62.5
27	85.0
28	72.5
29	82.5
30	77.5

TABLE F.1. Individual SUS scores for all participants (N=30).