Exploring the Untapped Potential of Text Fields in Creative Software

Maëva Calmettes, Jean-Baptiste Joatton, Nolwenn Maudet, Joëlle Thollot

Exploring the Untapped Potential of Text Fields in Creative Software

Explorer le potentiel des champs-texte dans les logiciels de création

MAËVA CALMETTES and JEAN-BAPTISTE JOATTON, École supérieure de design, France
NOLWENN MAUDET, Université de Strasbourg, France
JOËLLE THOLLOT, Laboratoire Jean Kuntzmann
UGA, CNRS, Inria, Grenoble INP, France

After the rise of direct manipulation, textual interactions have been progressively devalued in creative software. Text fields within creative software currently support limited use cases such as fine tuning of numerical values or layer naming. Following the increased popularity of programming in art and design, we believe that text field based interaction can be enhanced so as to combine the unique strengths of GUI with those of text. Based on the anatomy of the text field: both an interactive interface element and a writing space, we propose a design space that explores its interactive capabilities to facilitate both reading and comprehension as well as to support writing. To explore its potential, we apply this design space to VectorPattern, a pattern creation tool that focuses on complex pattern repetition based on explicit mathematical expressions written in text fields. With this work, we call for reevaluating the place of text fields within creative software.

CCS Concepts: • **Human-centered computing → Interaction design theory, concepts and paradigms**;

Additional Key Words and Phrases: Text Fields, GUI, Creative Software, Creativity Support Tools, Design Space

Après l'essor de la manipulation directe, les interactions textuelles ont été progressivement dévaluées dans les logiciels de création. Les champs-texte ne sont utilisés qu'à des fins limitées, comme le réglage fin des valeurs numériques ou la désignation des calques. Nous pensons que les interactions autour des champs-texte peuvent être améliorées afin de combiner les forces de l'interface graphique avec celles du texte. En nous basant sur l'anatomie du champ-texte : à la fois un élément d'interface interactif et un espace d'écriture, nous proposons un design space qui explore ses capacités pour faciliter la lecture et la compréhension de l'écrit, ainsi que pour soutenir l'écriture. Nous appliquons ce design space à VectorPattern, un outil de création de motifs qui se concentre sur les répétitions complexes à l'aide d'expressions mathématiques saisies dans des champs-texte. À travers cette recherche, nous appelons à réévaluer la place des champs de texte dans les logiciels de création.

Mots-clés additionnels : Champs-texte, GUI, Logiciels de création, Support à la création, Design Space

Reference:
This is the author's draft version of the work. It is posted here for your personal use. Not for distribution. The definitive version of record was published in IHM '23: Proceedings of the 34th Conference on l'Interaction Humain-Machine.
ISBN 978-1-4503-9825-1
https://doi.org/10.1145/3583961.3583971

1 INTRODUCTION

Textual interaction, through the use of a keyboard to input characters, have historically been the primary means for interacting with computers. In this paper, we define a textual interaction as an interaction that happens through the interpretation of text by the software. Whether the typed text is an instruction, a calculation, a numerical value or a keyword, all can be considered textual interactions if they are meant to be interpreted. Additionally, the language used for interacting with the software does not have to be a formal language, i.e. a programming, query or markup language. Interacting through text can effectively be achieved using natural

2023. Manuscript submitted to ACM

language, whose interpretation can be done via a semantic analysis of words or by using special symbols or keywords. As an example we consider a search engine input field to be a textual interaction, even though users generally use natural language. The entered text not only constitutes keywords for the algorithm, but can also contain additional symbols for an advanced query. Users can input brackets, a hash or arithmetic symbols to imply specific meanings or functions.

However, since the advent of direct manipulation interfaces, as described in Ben Shneiderman's seminal article [44], textual interactions have progressively been devalued in HCI. Indeed, textual interactions proved to be more complex to apprehend than interface controllers based on direct manipulation. They are prone to syntax errors as they require extensive knowledge, and slow due to the typing process and sometimes the required additional compiling time.

In recent years however, as stated by Norman[35] textual interaction, especially through programming, has seen a revival in arts and design because it provides advanced capabilities and an access to complexity which is difficult to emulate via direct manipulation tools [15]. Textual interaction supports a lot of creative practices and many of the new creativity support tools make use of its strengths [43] .

However, to harness its power as a creative tool, designers need to learn full programming languages and use complex IDE interfaces, both having a steep learning curve [27]. Moreover, they generally require that designers move away from traditional Graphical User Interfaces (GUI) and direct manipulation interfaces.

One solution to this set of issues has been node programming which provides means for designers to access most of the capabilities of programming without requiring them to interact via writing. Node-based programming is a widely spread visual programming method where the program is designed in the form of rectangular shapes connected by wires. Each shape usually corresponds to a simple function with exposed parameters. It allows the creation of complex algorithms without the need of learning a full programming language but it can become hard to read when the number of nodes increases. Still, writing provides certain specific benefits that are hard to emulate within the node paradigm, e.g., more modularity and malleability as well as the ability to copy and paste across software. Writing is also an entry door to more complex programming languages.

Whether it is to search for a term or command, to enter a numerical value or to program a script, textual interactions are discreetly omnipresent in creative software and beyond. In most mainstream software, textual interaction still exists, but in the form of text fields, that is standardized text-input widgets embedded within larger GUI. Command line prompts, code editors and text fields represent different modalities for users to interact using textual interactions. In this paper, we chose to focus on single-line GUI textual interaction, i.e. text fields, because they can be more easily combined with other GUI components. Textual interaction via text fields can be used, not only to input values or display text but also to input expressions that produce effects.

In creative software, text fields are already used for various but relatively limited use cases. The first one is to precisely manipulate numerical values. For example to edit the typeface size, or to change an element's width. The second widespread use is to name objects, files or layers in the software. Text fields also appear to be useful to search in a list, e.g. through a list of commands or to find a specific object in a whole 3D scene. A less obvious practice is to use text fields to transfer data across software via copy-pasting. The extensive use of color hexadecimal code is a great example of a text value that will be rarely directly edited character by character and often copy-pasted. Lastly text fields are also used to enter scripts or expressions or other kind of code for an advanced used of the software. A well-know example is After Effects's expressions that let users control animations programmatically. Based on these observations, a revival of programming in arts and design on the one side and the lack of interesting use cases for text fields in current creative software on the other side, we posit that, within GUI, textual interactions through text fields have an untapped potential and provide a promising avenue for research. We think that text fields are under-exploited and we propose to explore their power as an interaction tool for creative software. Text fields have the potential of combining the strengths of GUI, such as direct manipulation and ease of learning, with those of text, including expressive power and modularity.

We focused our exploration of text fields in creative software, i.e. software used within the creative process to create and manipulate digital content, such as images, vector shapes, 3D models, videos and audio. In this paper, our research and proposed concepts are primarily centered around graphic design tools, i.e. software used to generate and manipulate 2D or 3D shapes, still and animated images. But we posit that most of our findings would be applicable to a wide range of creative software outside graphic design tools. We believe that developing textual interactions is also a way to support creativity inside creative software, thus falling within the scope of Creativity Support Tools (CST) features. CST are software meant to assist users in the creative process (e.g. to generate new ideas or organize their thoughts), and include brainstorming tools, as well as support to non-digital activities like choreography. Therefore, while our research scope and references include CST and CST features, our proposed concepts and findings may apply to creative software specifically.

As a first step, we chose to concentrate on single line text fields, because of their prevalence upon multi-lines ones in creative software. As a standardised GUI element, text fields are available in a lot of graphical environments and GUI toolkits. Based on the anatomy of the text field, both an interactive interface element and a writing space, our first contribution is a design space that explores its interactive capabilities to facilitate both reading and comprehension as well as to support writing. Our design space is about the combination of direct manipulation and textual interaction

As a second contribution we demonstrate the use of the design space as a generative tool. We apply it to generate ideas for a graphics application, VectorPattern, a very simple prototype that we developed for this experiment. VectorPattern focuses on repetitive pattern creation using a unique text field to write mathematical expressions. We show that our text field design space supports the creation of original ideas to assist the user in math writing tasks.

2 RELATED WORK

Text-based interaction supports a lot of creative practices and researchers have long tried to develop and enhance features that support these practices. In this section we review the different strategies created to support writing practices.

We divided these initiatives in five categories that represent the five goals for supporting writing we identified while studying existing work: (1) understanding the returned value and its resulting effect, (2) understanding the basis of the language, (3) discovering and exploring the language, (4) helping with writing organization and (5) making the writing process more usable.

In this paper, we focus on text fields. However, compared with text fields as single line GUI components, IDE, understood as multiple line writing spaces, have implemented a lot of innovative features to support textual interaction, especially those related to programming. We thus decided to include IDE in our related work sampling but we still focus on GUI based strategies, especially those trying to be as close as possible to the writing space and the writing time.

2.1 Understanding the returned value and its resulting effect

Textual interaction via text fields can be used to input mathematical or keyword based expressions that produce effects and may return values. In creative software they can for example be used to manipulate the object properties such as its size. In this case, the returned value would be the computed size and the produced effect would be the object resizing.

In his 2012 article "Learnable Programming" [45], Victor proposes several innovative paths to support programming languages learning as close as possible from the writing space. His proposal to reveal all the states is a way to close the Gulf of Evaluation that was identified by Norman as: "the amount of effort that the person must make to interpret the physical state of the device and to determine how well the expectations and intentions have been met" [34]

Some of his ideas such as directly evaluating variables in the code and displaying its result on demand have been implemented, for example, in the Light Table [17] code editor.

Victor also proposed interaction principles to visualize differently the execution process of a program. Building on the idea, well-known in the context of animation software, of the timeline and its key frame, he lets users control the code execution, providing a better connection between each line of the program and its visual output. The idea of slowing down execution to better understand how the program works was also implemented in the pythonTutor [19] project. They introduced, inside the IDE, the idea of movement deconstruction as it is practiced in sports.

Adding some visual help is another way to facilitate understanding and writing. For example, Jacobs et al. [31] designed a visual code inspector that offers to artists the possibility to directly manipulate the execution process, while Hoffswell et al. [24] propose to integrate graphics within data files so as to circulate between different levels of languages.

Another option is to favor back and forth between the writing space and the results. For example, Sketch-n-Sketch [22] uses the notion of bi-directional and output-directed programming: the interface allows the user to manipulate the output (a vector drawing in svg format) as a way to visualize and modify the code. Glisp [21] also develops the idea of software offering hybrid modes of interaction, between code writing, direct manipulation and interface components. Glisp supports writing by simultaneously showing multiple views of the same object and allowing to seamlessly change between different modes of interactions. Selecting a line of code in the IDE section of the interface opens a control panel that lets users modify different values with their cursor, and the resulting graphical output can also be modified through direct manipulation.

2.2 Understanding the basis of the language

Understanding a language generally happens through reading a more or less accessible, complete and pedagogical documentation. A full documentation is often found on a dedicated web page, while some software include a limited embedded help in a separate editor's section. Various projects seek to better integrate documentation's access. Accessing documentation easily has been theorized in the context of Information foraging theory [38]: providing a better cost–benefit ratio by reducing the effort needed to find information (path) and improving the relevance of results (content). The toolkit Processing [41] offers the possibility to access offline documentation by right-clicking on a function name in the IDE. Some projects such as The Barista [29] emphasize access to language documentation by integrating it inside the text body. Codelets [37] goes even further and transforms the documentation into a graphical control interface allowing users to manipulate their code from the documentation.

Alternatively, Graphtoy [40] provides another type of hybrid documentation that relies directly on clickable terms. It lets users select functions, operators, numbers and variables in a button panel so as to write mathematical expressions. The permanent visibility of these functions encourages experimentation and serves both as a knowledge base and as a simplification of the writing process.

2.3 Discovering and Exploring the language

Learning a language can also be done by experimenting. In such a case, the user progressively understands the language inner logic through trial and error. This learning process is what Draper and Barton [12] called learning by exploration (LBE) in their 1993 study detailing this process in highly visual GUI. They introduced three kinds of affordance a GUI command must express to support LBE, which can be applied to text-based interactions alike. Several projects have been designed so as to encourage this exploratory learning approach. The (t,i,x,y) [28] project proposes a step by step onboarding into the logic of the program and lets users, through the use of examples, understand the basics. The minimalist interface made of a writing zone and a display zone uses comments as a way to explicit the program in an efficient way. Added comments invite users to modify the code through advises such as "multiply the time to change the speed". Sliderland [5] uses a similar principle and focuses on a progressive learning process that lets users discover all the possibilities. Such a strategy is typically found in video games, where the user progressively learns the interaction mechanisms during a first phase of the game. In a less guided approach, the live coding application Hydra [25] has two features that help users getting started: a *random sketch* button that loads a random example and a *make random change* button that randomly modifies variable or parameter values.

These recent projects use different strategies to overcome the apprehension of *white page* (or the *white text area*), and make the users enter in an undemanding way in the manipulation of the code

2.4 Making the writing process more usable

Research and commercial projects have proposed many different ways to support a more usable writing process. Generally, this usability is meant to accelerate writing.

A good example is Google Sheets that offers a large number of interactions to support the writing of formulae: functions suggestions via drop-down menus, help panel for each function, syntax coloring, dynamic result computation, but also automatic suggestions to apply the formula to a different row. To go further, the NLyze project even try to use automatic translation techniques applied to spreadsheets [18]. Still in Google Sheets, textual interactions are favored through subtle suggestions (the use of tabulation, arrows keys, etc.). Supporting keyboard shortcuts and textual interactions is also a way to avoid breaking the user's flow and switching between several interaction modes.

Since their widespread use, Word processing software have been offering automatic correction features such as orthographic verification. Commonly represented by a red underline under an uncertain word spelling, it lets the user choose to validate or not the suggestion. Messaging software, such as iMessage, even automatically replaces misspelled words (often typographical mistakes) without validation.

Queries writing happens most of the time in text fields, generally single line ones. The most common support for requests is probably automatic suggestion by drop-down list and auto-completion. Search engines also offer a lot of operators that lets users access specific and advanced functions instead of having to navigate through the advanced research menu.

Specialised search engines require users to know a query language such as SQL or SPARQL, or to be familiar with regular expressions (RegEX). Sparklis [13] offers a SPARQL query builder that lets user construct a request using natural language to specify its parameters. AutoRegex [1], automates translating back and forth between English and RegEx. Lately, many projects have emerged to try to use machine learning to automate corrections or even translate from a language to another.

Beside programming-oriented software, other types of software have developed interesting uses of text fields. Discord [36], Notion, Overleaf or Minecraft, all have integrated keyboard shortcuts or commands that give access to some functions through the use of specific characters ("/" ou "@" for example). In these software, commands act like shortcuts, quicker than their direct manipulation equivalent as they permits the user to stay in a textual interaction mode.

Existing assistance techniques range from text field enrichment to writing automation. These methods tend to keep the user exclusively in one mode of interaction: either they replace writing completely or they help to manipulate the software without leaving the keyboard.

2.5 Helping with writing organization

We also identified a last category, albeit a lesser developed one: features that support writing organization. Within note-taking and productivity tools, such as Notion, textual interaction is favored and provides ways to navigate within text and across documents by using: any words can be associated with another page and serve as a navigation shortcut. Many CLIs implement an history of previously used commands and offer a way to navigate through it with the help of up and down arrows keys. Some CLIs also allow users to copy text by selecting the targeted text with the mouse cursor and double-clicking on it. Once copied, a simple right or middle click in the terminal permits to paste it. The (t,i,x,y) project integrates a quick way to share and store projects by their unique URL. By a simple tap on the enter key, users can generate a URL that contains the mathematical expression used, which acts has a readable hyperlink.

All these examples show that textual interactions are generally supported through other interaction modalities (direct manipulation, etc.). In that sense, writing happens in a highly hybrid and interactive environment. In contrast with IDEs that provide many features that support textual interaction, relatively few examples exist within traditional creative software GUI even if a few do support scripting modes such as Adobe After Effects.

The discrepancy between the wealth of writing support strategies developed mainly within IDEs and the relatively few ones used in text fields supports the observation that text fields have an untapped potential that remains to be explored.

3 A TEXT FIELD DESIGN SPACE

To demonstrate and explore the potential of text fields as a GUI component in creative software, our first contribution is to build a text field design space.

3.1 Design Spaces in HCI

A design space is a conceptual space [4] that can be used to understand, analyze and explore the potential of technologies. They can be developed around devices e.g. smartphones [2], or interaction techniques e.g body-centric multi-surface interaction [46]. Design Spaces often take the form of a "multidimensional matrix that contains all possible combinations of parameters that are relevant to a specific problem"[4].

A design space is both a descriptive and a generative tool. First, Design Spaces can provide a richer and systematized understanding of an interaction technique, such as text fields, by mapping out its different properties. In that sense, design spaces help to "organize knowledge in a way that is meaningful to designers" [3]. Design spaces supports researchers and designers in their understanding of the state of the art of a specific interaction technique, by showing dimensions that are already well populated and explored. Design spaces can also, by contrast, help us uncover potential unexplored areas or properties. Furthermore, they provide inspiration for designers by showing them promising or underexplored possibilities for interaction design [23].

3.2 Text Field

The versatility of text fields makes them appropriate in many contexts. If we refer to Alan Cooper's controls classification [11], text fields can be considered as an intermediate between the 4 categories: imperative controls, selection controls, entry controls and display controls. Mainly described as an entry control, they can be bounded or not, depending on the level of textual input validation. Text fields can also be used to display content as a text container, in the way of an interactive dialog box. Some features such as auto-completion, or drop-down menu can turn it into a kind of selection control. It would also be the case if the text field only accepts a restricted list of keywords. And if the expected text is a command, it could also be considered as an adjustable imperative control.

The text field is therefore both an interactive interface element and a writing space that can be enhanced to facilitate both reading and comprehension as well as to support writing. By designing a text field design space our aim is to provide a way to systematize improvement opportunities of a given text field, from a blank text field starting point.

3.3 The Text Field Design Space

GOAL	CHARACTERISTIC(S)	STATE
	Visual characteristics	
Understanding the returned value and	*color, shape, visibility...*	Rested
its resulting effect	Layout characteristics	Highlighted
Understanding the basis of the language	*position, size, orientation, spacing...*	Focalised
Discovering and exploring the language	Additional assistive element	Edited
Helping with organisation	*icon, box, list, specific cursor shape...*	Selected
Making the writing process more usable	Text-manipulation & interpretation	Locked
	autocompletion, dorking, commands...	

Table 1. Text field design space dimensions.

Our design space is composed of three dimensions (see Table 1): the goal of the concept generated by the design space, the text field characteristics used to achieve the goal, the text field state during the interaction. We developed the design space goal dimension using our classification, presented in the related work section, of existing strategies used to support text-based interaction. The two other dimensions were developed through a careful analysis of existing examples of text fields in existing creativity software as well as in research literature. We also took into consideration their characterisation in popular UI Design systems such as Google Material UI [16] and GUI toolkits and libraries such as Tailwind CSS [48] or QT Designer [9]. This allowed us to assess a wide set of properties that are standard enough to be implemented in a production context.

3.3.1 Text Field Goals. We propose 5 main goals for the text field enhancement. They can be instantiated for each specific contexts.

Understanding the returned value and its resulting effect. The goal here is to help the user to understand the link between the typed text and the resulting behavior of the application. For example, in a spreadsheet, one may want to help the user to understand the result of a formula and its underlying operations, such as a sum or a mean exact computations. Another sub-goal could be to help the user to comprehend which property or action their input is changing and its units. For example understanding that the number next to the typeface name will indeed change the typeface size and that "pt" means that the value corresponds to a number measured in points.

Understanding the basis of the language. As soon as the text field accepts more complex inputs than a number, the user has to understand what is a valid input. Following the spreadsheet example, the user should understand that a formula starts with = symbol. The input text can be a formula, but also a list of keywords or a piece of code. Thus, this goal aims to help the user to discover the main syntax features of the textual input. It can be a support to learn the syntax (from a complete language or simply the correct format), or to understand the vocabulary (which keywords can be used, what a specific keyword means, what are the valid abbreviations).

Discovering and exploring the language. Once the user knows how to write valid inputs, he still needs to understand the possibilities offered by the language. This is especially useful as a creativity support feature as a mean to empower the user creativity. Sub-objectives for this goal include knowing the extent of the language (which functions and operators can be used), encouraging divergence (using different structures and clever expressions or algorithms), making expressions more complex (using nested functions and operators), or avoiding the "white field" effect (reduce the wandering time before writing anything in the field).

Helping with organisation. Here we propose to leverage the power of text as a way to organize ideas, store successive experiments, encompass copy-pasting and try and error approaches. This goal is about producing a situated documentation for a project (keeping important phases and insightful history), making it easier to share and reuse work (remixing a project, show-casting it), and generally keeping everything organised.

Making the writing process more usable. This last goal focuses on the user comfort. When inventing new text field based interactions, we must keep in mind their usability. It does not necessarily mean that the interaction should be faster, but more that it should be easier or more natural, what we call usable here. Thus commonly applicable sub-goals are speeding-up the writing process (helping to write faster, creating shortcuts or removing writing all-together), simplifying the writing (using an simpler language or an intermediate) or helping with typing mistakes (correcting mistakes, preserving a working state, suggesting).

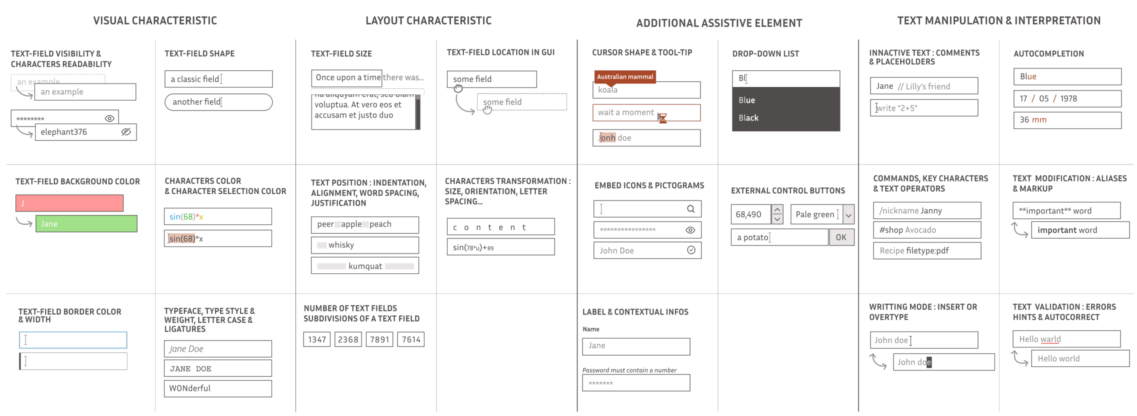


Fig. 1. Characteristics of the text field.

3.3.2 Text Field Graphical Characteristics. One of the text field main interests lies in its numerous graphical characteristics which makes it a rather complete interface element. As shown in Figure 1, we divide this dimension in 4 main characteristics groups: visual characteristics; layout characteristics; the use of an additional assistive element; and text manipulation and interpretation features.

Visual characteristics gather all graphical properties of the field and its text content. It includes characters and text field colors, text field shape and border style, text legibility... All these attributes are commonly found in graphical libraries.

In *layout characteristics* we consider all properties that relate to positioning and size. They differ from visual characteristics in that they impact the whole interface and may have to be designed at the same time as the rest of the GUI.

Additional assistive element groups all the text field external elements. We chose here to consider the most commonly used elements, such as icons, drop-down lists or cursor shape modification. This category can be easily extended as there is no strict limitation to genuine assistive elements, and it can also be a combination of multiple nested elements.

The last characteristics relate to *text manipulation and interpretation features*. Theses characteristics include all features that directly edit the text or make use of a specific semantics. We also considered features such as dorking, i.e. the capability to input advanced queries that goes beyond used language scope thanks to keywords or special characters. They are commonly embodied by conventional graphical forms or hints. For example an error hint resulting from a spelling verification is usually displayed by a red wavy underline under the misspelled word. As a system computation, they may impact any other characteristics group. For example, the text field border turns red when the content is invalid, or the text field itself is hidden and only shown when a keyboard shortcut is pressed to open a search bar or set a precise value.

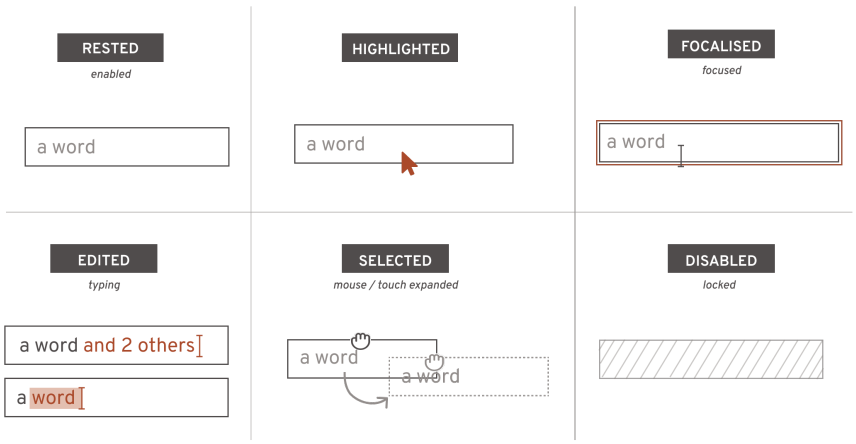


Fig. 2. States of the text field.

3.3.3 Text Field Interaction State. The third dimension of the design space describes the state of the text field during the interaction (see Figure 2). When we take a look at design systems and UI guidelines, like Google's Material, we often see that text fields graphical variations are described by their state. Even if they can be named slightly differently, text field state names are mainly standardized across design systems. This dimension takes into account active states of the field rather than the events that occur around or in it. It allows to diversify interactions with the text field, which can happen before or after the editing of the field content. The first state is the *rested* state, which is the initial state of the text field. When users do not interact with the text field in any way, the field is considered in a rested state. This state implies that the field is listening to events and its content can be edited, but does not necessarily mean that the field is visible or in its main graphical form.

The *highlighted state*, a commonly used state in direct manipulation GUI, is closely linked to the use of a pointing device. Often described as the hovered state, this appellation has the flaw of not applying to other means of interaction and navigation such as keyboard shortcuts or touch interactions. The highlighted appellation is intended to be more generic and depicts a highlight in the field, whatever its trigger.

The *focalised state* is triggered when the user effectively enters in the text field edition mode. We consider this state to last until the user starts typing or editing the text content. It can used as an in-between a highlight and an edition, when more information can be shown before the actual input.

The *edited state* groups everything that relate to text manipulation. It includes typing new values but also selecting, moving, coping, pasting or erasing text or characters from the field.

The *selected state* is a less common state, often named mouse or touch expanded. While it can be close from the highlighted state, this state asks for a deliberate or specific action from the user that is not typing, such as clicking on an option in a list. It is often triggered by a drag or a click on the field border, an icon or an arrow.

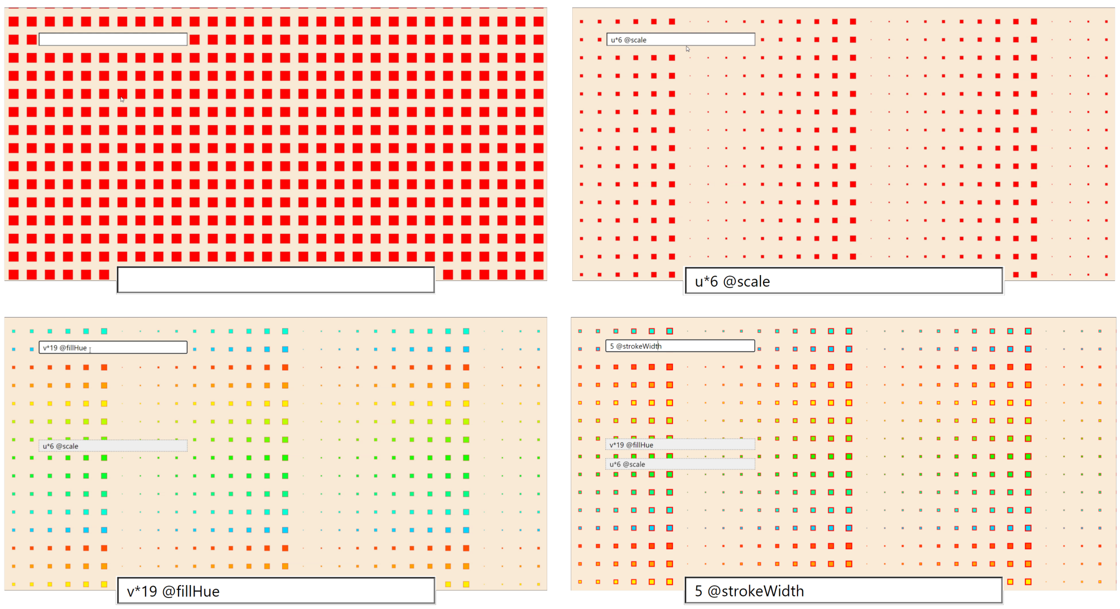


Fig. 3. VectorPattern interface. The drawn pattern is computed based on formulae written in a text field. The application starts with an empty field and a default pattern (red squares). We show here successive formulae written by the user. First a formula function of u (horizontal coordinate) is applied to the scale, using the @scale keyword. Then a formula function of v (vertical coordinate) is applied to the fillHue property. And finally a simple constant is applied to the strokeWidth.

Finally the *disabled state* characterized a lock in the text field edition feature. It can be used as a way to display information or to wait for a specific action from the user or computation from the program.

4 EXPLORING THE GENERATIVE POWER OF THE TEXT FIELD DESIGN SPACE

As the authors backgrounds are in graphics design and image synthesis, we have chosen to explore a simple graphics application in which text input would be the focus of the tool. Such an example could have been taken outside the graphics creation, like in music or movies support tools, as long as the text would be the main interaction. Following discussions with digital artists and textile designers who are curious about creative coding but not familiar with math concepts, we have chosen visual pattern creation through mathematical formula writing as our test application.

4.1 VectorPattern: our experimental setup

There is a long tradition of creating patterns in visual arts. They are based on a repetition principle that can go from regular to random, producing unity within the artwork. Artists and graphic designers currently produce most of their patterns using creative software, such as Illustrator, Inkscape or specialized applets, where they can find dedicated plugins offering standard repetitive procedures. However, if they want to depart from these predefined layouts, they have to manually arrange their motives, making the task laborious for complex repetitions such as using different repetition rules for each property of the motif. For that reason, the complexity of a pattern generally relies more on the complexity of the repeated pattern, than on the repetition itself.

As the repetitive nature of a pattern is intrinsically algorithmic, using a programming tool, like Processing, is a viable alternative to produce complex repetitions as can be seen in the creative coding community. Several attempts have been made to ease the pattern creation process either in the computer graphics research community via the definition of shape grammars [42], dedicated API [32]; or by commercial companies via node-based systems such as Patternnodes [20].

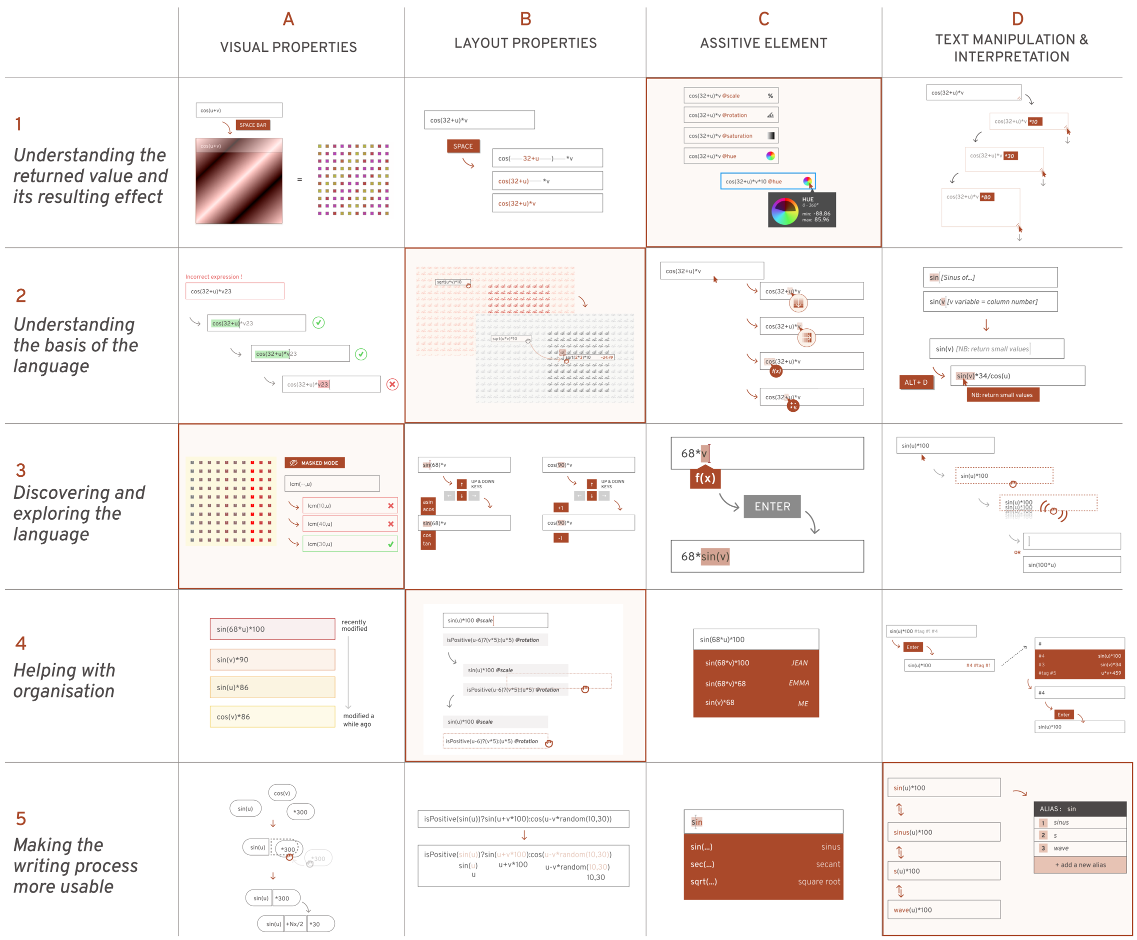


Fig. 4. Various ideas we obtained by applying the design space to our use case VectorPattern, trying to cover multiple combinations.Implemented ideas in the following section have a colored background. See Appendix A for a short description.

Following this line of work, we propose to apply our text field design space as a means to explore a new compromise between programming and GUI via the use of a unique text field. For that, we have designed a simple applet, called VectorPattern, in which a pattern is created by repetitively placing a basic shape on a simple regular grid parameterized by u, v coordinates. The user can write mathematical expressions, function of u and v, that will apply to a given property such as the motif transforms or style attributes, using the @ symbol. The expressions are written in a unique text field displayed on top of the resulting pattern as shown Figure 3.

Our goal here is not to evaluate Vectorpattern itself, but to show that our design space supports the creation of diverse original ideas so as to help a naive user to write meaningful formulae. This is why we have chosen a very strong constraint: in VectorPattern, the text field is the only tool available to generate and manipulate the pattern.

4.2 Applying the text field design space to VectorPattern

In order to use our design space to produce interactions that will help naive users understand VectorPattern concepts, the first step is to adapt the design space goals to our context:

(1) The first goal, *understanding the returned value and its resulting effect*, comes down to making the link between the written values of the properties and the pattern. First users have to understand the returned values of the math expression, and then comprehend that it will impact the graphical properties of each element of the pattern. For that the user has to discover the properties he can play with, identify the unit used and the range of values that are making sense, ...
(2) *Understanding the basis of the language* corresponds to learning the basis of writing mathematical expressions. It means knowing the basis syntax and functions of the language and respecting mathematical rules to write computable expressions. This goal also includes the understanding of the two variables u and v usage and that @ allows to choose the property on which the expression applies.
(3) *Discovering and exploring the language* will help the user play with the full range of available math functions and apply them to all the available graphical properties.
(4) *Helping with organisation* is similar to the definition of design space: help to organize the workspace, save expressions, have an effortless workflow in the software...
(5) *Making the writing process more usable* is also similar to its definition in the design space: simplify, speed up or personalize writing to each user need.

Based on the text field characteristics dimension, we organized three working sessions in order to generate ideas for each of these goals. The three sessions lasted one hour each and have been carried out by the authors (an interaction design graduate student with graphic design background, a computer science researcher, two interaction design researchers) using a constraint-guided brainstorming approach. We chose to combine specific dimensions of the design space that are currently under-explored in current software as we thought this would lead to original and non trivial ideas.

Ultimately, we obtained around 20 ideas (see Figure 4) among them we chose to select the most original ones, that is not commonly found in creative software in terms of type of help given in the text field; type of interaction; or the used text field characteristics.

In order to demonstrate these solutions, we have chosen to implement each concept separately. This allowed us to develop rich and novel interactions without being limited by the incompatibility of certain interactions with each other. We therefore do not target a complete version of VectorPattern but a set of independent interaction techniques that could each be applied to VectorPattern.

4.3 Resulting ideas

To describe each generated idea and explain how it could be useful, we will follow Elsa, a fictional user who discovers VectorPattern for the first time. She is a graphic designer with no particular knowledge of programming and a vague memory of high school math. More used to mouse-based vector software such as Adobe illustrator than traditional IDEs, she is looking to create an interesting pattern that features complex repetitions for a project.

We start with the basic VectorPattern interface that does not provide any help and then for each design space goal, we present one concept that uses one or combines several characteristics from the text field design space and uses them to achieve the goal. Resulting ideas are showed in the figure 5, and described in the following paragraphs. The following examples are meant to demonstrate the potential breadth of novel interaction techniques involving text fields that can be generated.

Standard scenario with classical text field

Elsa meets the VectorPattern interface for the first time. On the screen there is only a pattern in full screen and a text field filled with a mathematical expression. Accustomed to direct manipulation software, Elsa's first instinct is to hover over the pattern and click

Fig. 5. Implemented ideas from the design space to our use case software VectorPattern.
Goal 1. Using icons and tooltip to explicit the selected property and its unit, range and default value.
Goal 2. A movable field which serves as an inspector tool to get more information about applied values.
Goal 3. A special "Masked Mode" providing incomplete expressions, a challenging way to learn the syntax.
Goal 4. Having multiple movable fields in the GUI for a better workflow.
Goal 5. Using custom or generic aliases to speed up typing or to help to remember key words.

on the elements to see if it has any effect. As nothing happens, she turns to the text field and its unknown expression. She changes a term in the expression, a number, and validates to see what happens. The pattern changes a little, but she does not really understand what's going on. She tries to erase all the content of the text field, and type things in. Nothing happens then, lost, she closes vector pattern and never opens it again.

Goal 1	*Understanding the returned value and the effects*
Charac.	*Embedded icon, Tooltip, Dropdown list*
State	*Highlighted*

Elsa meets the VectorPattern interface for the first time. After hovering the pattern to no effect, she notices that the only text field of the UI has a small icon next to its text content. Used to seeing information tooltips inside text fields in forms, Elsa hovers over the icon with her mouse cursor. A tooltip box opens right next to her cursor with some information about the field: its name, unit, range, default value and a short description. Elsa then understands the link between the "@scale" that's written in the field and the icon: the text content of the field has something to do with the scale of the pattern. She thinks that there must surely be other properties. To find out, she deletes the "@scale" from the expression. As soon as she has deleted a letter from the expression, a drop-down menu opens below the field, offering various values starting with "@". She scrolls through the list with the arrows on her keyboard, effectively changing the pattern in real time. In just a few taps on her keyboard, Elsa now has had a preview of all existing properties, with additional information if needed thanks to the property tooltip.

Goal 2	*Understanding the language and its syntax*
Charac.	*Text field position*
State	*Selected*

In this scenario, the text field has now a small handle indicating that the field can be moved. When Elsa grabs the handle, an outline appears over the pattern in the interface. She then moves the field above the pattern by dragging the text field by its handle. As she moves the field, the pattern elements are highlighted in green one by one, indicating that she can actually drop the field right over them. When she drops the field a copy of itself is snapped on top of the element she was overing. The content of the field has changed: instead of the variables U and V there is now a numerical value. In addition, a small box has appeared in the field indicating the result of the expression : "=14.1". Intrigued by this behavior she selects the handle again and moves the field to another element. The value of the field changes again, as well as the indicated result. She then understands that the result indicated in the box seems to indicate the value of each element, in her case the scale of the overed element. By trial and error she moves the field above the pattern elements multiple times until she understands that the values of U and V correspond to the coordinates of each element of the pattern.

Goal 3	*Understanding and exploring the language*
Charac.	*Character visibility, Text field color*
State	*Edited*

After trying a few patterns, Elsa would like to know more about the language used to create more diverse patterns. Not used to reading language documentations, she would rather learn on the fly by experimenting. She decides to activate the *"masked characters"* mode, a feature that proposes random patterns with pre-written expressions to be guessed by hiding one or more characters. As soon as the masked mode is activated, the content of the active field changes, showing an expression with a character grayed out, replaced by a box character. The text field is also highlighted in red, suggesting an error in the content. When reading the text field content, Elsa realizes that it is the "a" in "scale" that seems to have been replaced. She then decides to replace the box with the missing letter. As soon as she made the change, the text field turns green indicating that her change was correct. After a few seconds the content of the text field changes again automatically, proposing a new hidden expression. Elsa tries again to complete it, looking for the missing characters reading carefully the functions and getting progressively more used to the syntax of the language.

Goal 4	*Helping with organisation*
Charac.	*Text field position, Icons*
State	*Selected, disabled*

Now that she is more familiar with the mathematical language and the different properties of Vector Pattern, Elsa is starting to refine her creation workflow in the software. After trying out a few formulas, Elsa comes up with a rendering she likes. In order to put the expression aside, she duplicates her expression by grabbing the field by the handle and dropping it right next to her expression. As soon as she changes the field she just created, the other field is disabled and grayed out, because the same property is also on the new field. Elsa continues to modify her expression until she reaches another rendering that suits her. To compare the two patterns, she switches between the two fields by clicking on one after the other, thus activating the rendering of the focused field. She decides that she prefers the second expression, but she would like to keep the first one aside for later. So she removes the designator @ from the "@scale" property, which makes the field inactive. She then moves it to a corner of the screen to put it aside in case she would like to reuse it later without it getting in the way of the pattern.

Goal 5	*Making the writing process more usable*
Charac.	*Drop-down list, Text manipulation: aliases*
State	*Edited*

Elsa wants to write an expression but never remembers if it is "u" or "v" that designates the columns of the pattern. After checking, she decides to add a "horizontal" alias for the variable u, so as not to make any more mistakes. She focuses on the only text field of the interface, deletes all its content and activates the alias mode by entering "::" in the field. As soon as the alias mode is activated, a drop down list of created aliases opens. With the keyboard arrows Elsa scrolls through the list to see the pre-defined aliases, such as "sinus" to replace "sin", or the symbol "π", not supported by the language, which is replaced by the word "pi". She adds her new alias by typing "horizontal = u" and validates with enter. The property is now added to the list of aliases. She also adds "vertical = v" and exits the alias mode by deleting the "::" in the field. She then tests her new alias by entering "horizontal*10" in the text field. When the alias is detected, a list opens under the field, describing the corresponding alias: "horizontal = u". She presses "shift+enter" to apply it and sees that the alias has been correctly replaced.

5 DISCUSSION AND CONCLUSION

Our design space proved to be usable and useful to produce original ideas for our use case. A next immediate validation step will be to test our design space generative power with professional HCI designers. We also would like to apply it to other use cases, like procedural 3D scenes generation or music creation which are close to our pattern design test application. While we demonstrated that the design space supports the generation of novel ideas for enriching text fields, we did not evaluate the intrinsic quality of these ideas. Despite these current limitations, thinking, developing and testing the design space gave us the opportunity to reflect more broadly on text fields and the role they play today in our current interfaces.

5.1 Textual interactions and direct manipulation

In creative software, a large part of the interactions are done via direct manipulation with the mouse. Yet, textual interactions persist, in the form of shortcuts or enclosed in the small writing zones of the text fields. But whatever the size and the presence of the writing zone, the power of the text remains the same. In the restricted and discrete space of the text field, users could have access to a large number of useful functionalities in the creation process. A single text field can thus become a notepad, a message area to be shared, a command line or a configuration area. These functionalities, main or complementary to the creation workflow, are accessible to different users, whether they are neophytes or more experienced.

However, developing text-based interactions does not mean reverting to command-line interfaces, or putting aside the direct manipulation habits that many people have acquired. On the contrary, we believe that we can, and should, mix the two types of manipulation, direct and indirect, in order to get the best out of both worlds[33]. Leveraging direct manipulation habits can also be a way to benefit from the Assimilation bias [8], i.e. habits user have kept from other software, and only partially accommodate it. In this respect, the text field element represents a very good opportunity to develop this kind of practice, because of its standardization and its wide implementation across environments and languages. Text fields could be a significant element for an hybrid interface revival, which are GUI that mix textual coding and direct manipulation. Since the late 90's there has been works and software that implement this kind of interface, including the work of Jürg Lehni on Scriptographer [30], an Adobe Illustrator plug-in to manipulate graphical objects with JS scripts. The advent of Flash in the 2000's had demonstrated the enthusiasm from the creative community for theses interfaces. Despite its proven value as a middle ground between the power of programming and the accessibility of direct manipulation, most of these projects have been discontinued. Some hybrid interface's remains can be found in the scripting part of widely used creative software such as Adobe After Effects, but it's development and popularity has not reach what was common in the Flash interface. Yet in recent years, hybrid interface has seen a renewed interest as shown by Baku Hashimoto's work on Glisp [21]. Developing text field based interactions could be another way to renew this approach.

More familiar to people used to direct manipulation, text fields are present in many software and web sites. It shares many graphical and interactive conventions with other widgets such as its different states and their associated representations. It can easily be augmented with other elements such as handles, toolboxes or icons, without representing a significant change in its affordance. In that matter, text fields used for entering numerical values are already augmented with a pointer interaction to increase the speed of manipulation (dragging to increase or decrease the value). Its simplicity also makes it more accessible than IDEs, which can seems complex or intimidating to new users.

5.2 Writing's friction and user experience

When discussing the use of text in a creative context, the first barrier to be overcome is the complexity of writing and the friction it represents. Writing is indeed more sensitive to errors and has a higher learning curve than other forms of interaction. Regarding "Principles for Successful Guessing", extracted from Polson and Lewis's CE+ model [39], software based on textual interactions can hardly apply 1 or 2 principles. The repertoire of available actions is obscure, feedback can be barely visible, multiple analogous alternatives are offered... Thus the development of interactions in CST has followed an ideal of user experience described as fluid, immediate and natural, in the replication of the movements of creation in the tangible world. Yet in a creative context friction can be interesting, even desirable. When creating patterns in VectorPattern, replicating a previously imagined or drawn pattern is very difficult. It would require a lot of projection skills, in addition to a good knowledge of math and software operations. In fact, creating patterns with VectorPattern is more a matter of trial and error, testing and then refining the mathematical expression. This way of creating, through attempts, trial and error, happens to works particularly well with the sensitivity of writing, which gives user the ability to propose significant changes quickly. This performativity of writing allows for quick and relatively simple changes of an expression, dramatically transforming the output and allowing for increased creative mobility. Moreover the use of mathematical expressions fits well with certain purposes such as describing a complex repetition. Even if it is indeed replaceable by other ways of generative creation like node-based systems, the efficiency and polymorphism (the ability to copy and paste text in different context and across software) of writing still is a great advantage.

Designing a GUI for direct manipulation requires abstraction of the software's operation so that it is understandable by users and limits errors. A slider, for example, has a minimum and a maximum, which prevents the user from going outside the range of expected values. However, the values or options provided do not necessarily coincide with all possible values, but rather with a logical choice from the programmers. Thus this graphical overlay tends to circumscribe the possibilities of control of the program on behalf of simplicity of use or learning. The textual form allows greater freedom in the user's choices, which are not reduced to a set of buttons, list items or nodes. The use of textual form permits a direct and free control with the operations of the program, in exchange for more frequent errors.

The error-proneness of writing can cause frustration, as it requires precise knowledge of the language and the program. It involves a certain amount of learning and mastery to get greater control over the software. At the same time, the rigour imposed by the textual approach helps to qualify users, by encouraging them to really understand and master what they are doing. Program operations become more explicit and intelligible to the user, who can then exploit it to its full potential. The use of text can be a means of regaining the control and experimentation power, both essential in the creative process. Despite this potential benefit, supporting creativity through text fields should involve mechanisms that alleviate or mitigate this error-proneness. We think that this can happen through several mechanisms: trying to show results as much as possible; reorienting users towards possible alternatives in case of error and guiding during the typing process to avoid errors altogether. In this last case, this can be done automatically, via auto-completion methods, but also in interactive ways through multiple options which would provide users with different choices.

5.3 Design space as a creative generation method

Text fields as design components have a certain number of characteristics that we identified and organized. However, the characteristics dimension may not be comprehensive and this is one of the possible shortcomings of our current design space. It has the positive effect of showing another avenue for exploring the potential of text fields: developing other text fields characteristics, beyond the ones we are used to see in current software. Through this lens, the text field design space can be generative in itself by welcoming the development of other characteristics. We also think that design spaces have a real potential as a method for UI or interaction design. This can help us examine and question the use of fixed and standardized components. Being able to dissect existing GUI component as a starting point to develop novel interaction could become an alternative to existing fixed component libraries that are being used in the industry.

5.4 Extending the design space - An interaction dimension

As we mentioned earlier, the design space can be extended to other dimensions, especially to a more interaction-centered dimension. We have classified 5 types of interactions allowing to transition between the states of the text field, deriving from Buxton's three-state model for graphical input [6]. These types of interactions are characterized according to their temporality (transient, durative and continuous) and the granularity of control they allow (predefined, rough or precise control). They can be associated with several types of controllers, including the most recent ones. Depending on the controller used and design choices some transitions may be nonexistent or one-way only. These basic interaction types can be combined together to form richer interactions abeit more complex (click and drag, or a keyboard modifier key and another keyboard key...).

(1) *Command-based interaction* : mouse clicking, tapping on a touchscreen, using keyboard shortcut, using voice command... Command-based interactions only allow a predefined action to happens. We consider them to be transient, i.e. to not last for a significant duration and to give immediate feedback. Thus we consider both "up" and "down" events rather than "pressed" events for a mouse click, a tap and a keystroke. If the action is hold and last in time we consider it whether as a repeated command-based interaction, as another delayed command-based interaction, or as the beginning of another type of interaction as described bellow.

(2) *Pointer-based interaction* : pointing at something with a cursor, dragging the pointer... Commonly used to select the text, to hover above the text field, or to move it around in the GUI. Pointer-based interactions are continuous and allow a rough but direct control over the field.

(3) *Text entry* : using a physical or digital keyboard, dictating text, writing using a tablet pen. Any form of text, traditionally a keyboard but can be extended to more original forms of text input such as hand writing recognition or a data-entry interface such as Dasher[47]. Text entry is dedicated to it's eponymous action and can be seen as a suite of transient actions of character typing.

(4) *Uni-dimensional interaction* (1D) : using 2 arrow keys, using the mouse wheel, using a gamepad triggers... A uni-dimensional interaction consist of any set of binary actions such as plus/minus or forward/backward attached to a specific set of controls. Controls can be common such as keyboard arrows but also more subjective or original like temporally bounding the 2 mouse triggers to a specific 1D action.

(5) *Multi-dimension interaction* (2D, 3D) : using 4 arrow keys, using a joystick, using a gyroscope sensor Following the uni-dimensional interaction, a multi-dimension interaction can allow a finer control by introducing more parameters. We include gesture-based interactions as a form of multi-dimension interaction.

We did not implemented this dimension in the proposed design space as we are unsure about its generative power. We found that these parameters are not mutually exclusive, i.e. they are often used together as alternative ways to interact with the text field. However, they seem to be useful for describing interactions in an analytical perspective, to highlight limited or redundant interactions possibilities.

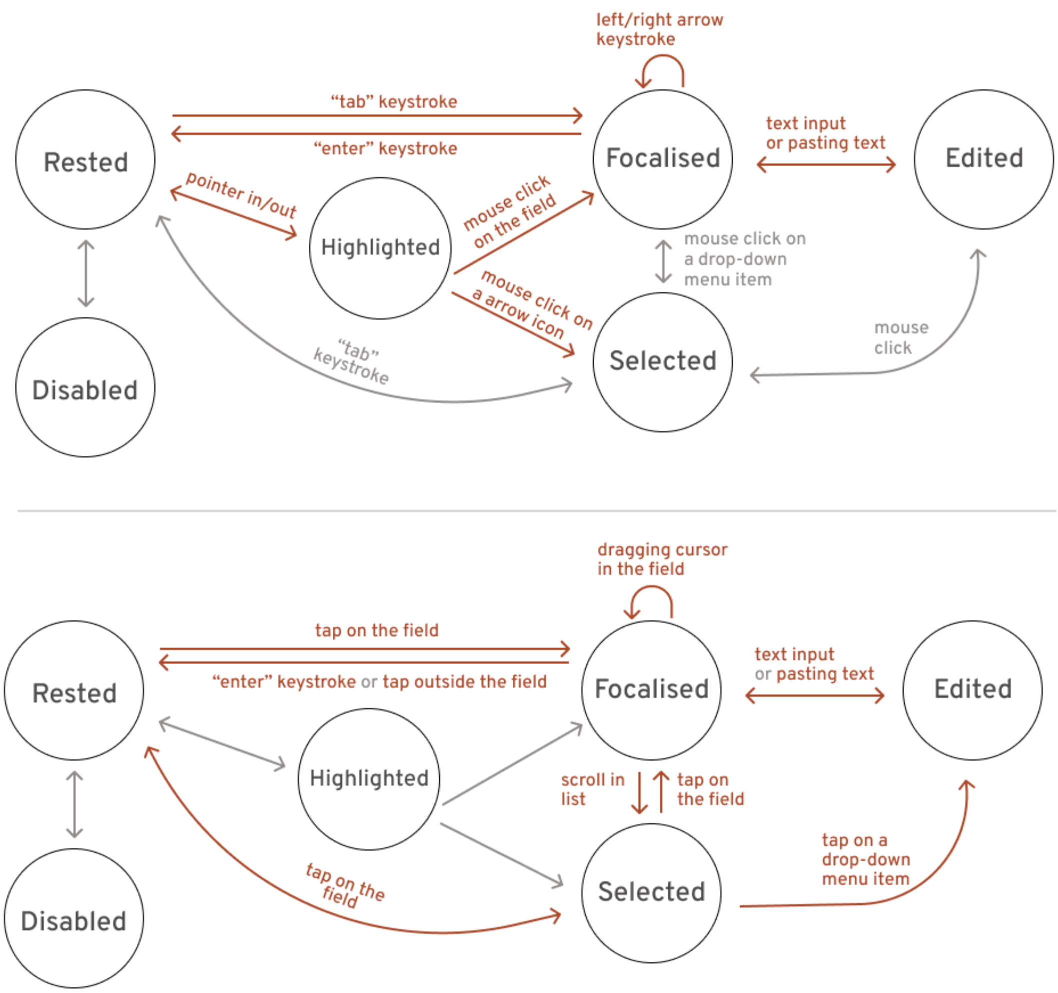


Fig. 6. Two synthetic examples of common text field interactions: with a mouse and keyboard setup, and with a touch-enabled smartphone.

5.5 Conclusion

In this paper, we have shown that, following the increased popularity of programming within the arts and design fields, there is an untapped potential for the creation and development of richer text fields, especially in the context of creative software. However, the language is not always known from users and if we are to really unleash the potential of text fields, we need to rework text fields so as to support writing.

As a first step in this direction, we have developed a text field design space based on the anatomy of the text field as well as writing support oriented objectives. We also demonstrated the generative power of this design space through the generation of novel ideas for extending interactive power of text fields.

With the advance and development of broadly accessible *text-to-image* machine learning technique, text fields and more broadly text as an input technique is making a comeback. Different examples of *text-to-anything* (image, sound, 3D model, etc.) modules, being integrated into design and mainstream software such as Photoshop [7] or Blender [26], show that textual interaction will most probably be integrated with other interaction modes.

Writing plain language *prompts* may seem easier to learn at first sight, but appears to be a codified practice that requires users to follow rules to make it effective. These unstated guidelines are becoming apparent through the still limited recommendations of various platforms [10] and the emergence of prompt-helpers [14]. The prompt text field could clearly constitute an important HCI problem space to which our design space can be used to generate effective supporting principles for this new type of writing.

In the case of prompts, writing support tools could help to support the explicability of such systems, including questioning its use. In that case, we could for example add a new objective to the design space: how writing support tools within text fields can support reflexive practices, especially for tools such as machine learning based ones, that are known to be prone to specific issues such as datasets that reproduce existing bias for example.

ACKNOWLEDGMENTS

We want to thank Vincent Tavernier. He has been our git and JavaScript guru during VectorPattern development. We also thank the reviewers for their precious help in making this paper better.

REFERENCES

[1] Aarya. 2022. AutoRegex: Convert from English to RegEx with Natural Language Processing. https://www.autoregex.xyz. [Online; accessed 19-January-2023].

[2] R. Ballagas, J. Borchers, M. Rohs, and J.G. Sheridan. 2006. The smart phone: a ubiquitous input device. *IEEE Pervasive Computing* 5, 1 (Jan. 2006), 70–77. https://doi.org/10.1109/MPRV.2006.18 Conference Name: IEEE Pervasive Computing.

[3] Rafael Ballagas, Sarthak Ghosh, and James Landay. 2018. The Design Space of 3D Printable Interactivity. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 2 (July 2018), 1–21. https://doi.org/10.1145/3214264

[4] Michael Mose Biskjaer, Peter Dalsgaard, and Kim Halskov. 2014. A Constraint-Based Understanding of Design Spaces. (2014), 10.

[5] blinry. 2022. Sliderland. https://sliderland.blinry.org. [Online; accessed 19-January-2023].

[6] William Buxton. 1990. A Three-State Model of Graphical Input. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction (INTERACT '90)*. North-Holland Publishing Co., NLD, 449–456.

[7] Christian Cantrell. 2022. The Stability Photoshop plugin. https://christiancantrell.com/#ai-ml. [Online; accessed 21-January-2023].

[8] John Carroll and Mary Beth Rosson. 1987. Paradox of the active user. In *Interfacing Thought : Cognitive Aspects of Human-Computer Interaction*. The MIT press, 80–111.

[9] The Qt Company. 2022. Qt Documentation: TextField QML Type. https://doc.qt.io/qt-5/qml-qtquick-controls-textfield.html. [Online; accessed 21-January-2023].

[10] LMU Munich CompVis, Stability AI, and Runway ML. 2022. DreamStudio Prompt Guide. https://beta.dreamstudio.ai/prompt-guide. [Online; accessed 21-January-2023].

[11] Alan Cooper, Robert Reimann, David Cronin, and Christopher Noessel. 2014. *About face: the essentials of interaction design.* John Wiley & Sons.

[12] Stephen W. Draper and Stephen B. Barton. 1993. Learning by Exploration and Affordance Bugs. In *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems* (Amsterdam, The Netherlands) *(CHI '93)*. Association for Computing Machinery, New York, NY, USA, 75–76. https://doi.org/10.1145/259964.260084

[13] Sébastien Ferré. 2017. SPARKLIS: An Expressive Query Builder for SPARQL Endpoints with Guidance in Natural Language. *Open Journal Of Semantic Web* 0 (2017). https://hal.inria.fr/hal-01485093

[14] Claudio Fuentes. 2022. Prompt Tool for MidJourney. https://prompt.noonshot.com/midjourney. [Online; accessed 21-January-2023].

[15] Don Gentner and Jakob Nielsen. 1996. The Anti-Mac Interface. *Commun. ACM* 39, 8 (aug 1996), 70–82. https://doi.org/10.1145/232014.232032

[16] Google. 2021. Material Design. https://m3.material.io. [Online; accessed 21-January-2023].

[17] Chris Granger. 2014. Light Table. http://lighttable.com. [Online; accessed 19-January-2023].

[18] Sumit Gulwani and Mark Marron. 2014. NLyze: Interactive Programming by Natural Language for Spreadsheet Data Analysis and Manipulation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 803–814. https://doi.org/10.1145/2588555.2612177

[19] Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-Based Program Visualization for Cs Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 579–584. https://doi.org/10.1145/2445196.2445368

[20] Mars Gäfvert. 2022. Lost Minds Patternodes 3 - Parametric design and animation. https://lostminds.com/patternodes3/. [Online; accessed 21-January-2023].

[21] Baku Hashimoto. 2020. Glisp. A Lisp-based Design Tool Bridging Graphic Design and Computational Arts. https://github.com/baku89/glisp. [Online; accessed 19-January-2023].

[22] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-Directed Programming for SVG. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 281–292. https://doi.org/10.1145/3332165.3347925

[23] Teresa Hirzle, Jan Gugenheimer, Florian Geiselhart, Andreas Bulling, and Enrico Rukzio. 2019. A Design Space for Gaze Interaction on Head-mounted Displays. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–12. https://doi.org/10.1145/3290605.3300855

[24] Jane Hoffswell, Arvind Satyanarayan, and Jeffrey Heer. 2018. Augmenting Code with In Situ Visualizations to Aid Program Understanding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3174106

[25] Olivia Jack. 2018. Hydra. https://hydra.ojack.xyz. [Online; accessed 19-January-2023].

[26] Carson Katri. 2022. Dream Textures. Stable Diffusion built-in to Blender. https://github.com/carson-katri/dream-textures. [Online; accessed 21-January-2023].

[27] Caitlin Kelleher and Randy Pausch. 2005. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv.* 37, 2 (jun 2005), 83–137. https://doi.org/10.1145/1089733.1089734

[28] Martin Kleppe. 2020. (t,i,x,y) => "creative code golfing". https://tixy.land. [Online; accessed 19-January-2023].

[29] Amy J. Ko and Brad A. Myers. 2006. Barista: An Implementation Framework for Enabling New Tools, Interaction Techniques and Views in Code Editors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada) *(CHI '06)*. Association for Computing Machinery, New York, NY, USA, 387–396. https://doi.org/10.1145/1124772.1124831

[30] Jürg Lehni. 2006. Scriptographer. https://scriptographer.org/. [Online; accessed 23-January-2023].

[31] Jingyi Li, Joel Brandt, Radomír Mech, Maneesh Agrawala, and Jennifer Jacobs. 2020. Supporting Visual Artists in Programming through Direct Inspection and Control of Program Execution. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376765

[32] Hugo Loi, Thomas Hurtut, Romain Vergne, and Joelle Thollot. 2017. Programmable 2D Arrangements for Element Texture Design. *ACM Trans. Graph.* 36, 3, Article 27 (may 2017), 17 pages. https://doi.org/10.1145/2983617

[33] Nolwenn Maudet. 2018. Reinventing Graphic Design Software by Bridging the Gap Between Graphical User Interfaces and Programming. https://doi.org/10.21606/drs.2018.611

[34] Donald A Norman. 1988. *The psychology of everyday things.* Basic books.

[35] Donald A Norman. 2007. The next UI breakthrough: command lines. *Interactions* 14, 3 (2007), 44–45.

[36] Notion Labs Inc. 2016. Keyboard shortcuts - Notion help center. https://www.notion.so/help/keyboard-shortcuts. [Online; accessed 21-January-2023].

[37] Stephen Oney and Joel Brandt. 2012. Codelets: Linking Interactive Documentation and Example Code in the Editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Austin, Texas, USA) *(CHI '12)*. Association for Computing Machinery, New York, NY, USA, 2697–2706. https://doi.org/10.1145/2207676.2208664

[38] Peter Pirolli and Stuart Card. 1999. Information foraging. *Psychological review* 106, 4 (1999), 643.

[39] Peter G. Polson and Clayton H. Lewis. 1990. Theory-Based Design for Easily Learned Interfaces. *Hum. Comput. Interact.* 5 (1990), 191–220.

[40] Inigo Quilez. 2020. Graphtoy. https://graphtoy.com [Online; accessed 19-January-2023].

[41] Casey Reas and Ben Fry. 2006. Processing: programming for the media arts. *Ai & Society* 20, 4 (2006), 526–538.

[42] Christian Santoni and Fabio Pellacini. 2016. gTangle: A Grammar for the Procedural Generation of Tangle Patterns. *ACM Trans. Graph.* 35, 6 (Nov. 2016), 182:1–182:11. https://doi.org/10.1145/2980179.2982417

[43] Martin Schneider and Moritz Wallawitsch. 2020. RemNote. https://www.remnote.com. [Online; accessed 19-January-2023].

[44] Ben Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (Aug. 1983), 57–69. https://doi.org/10.1109/MC.1983.1654471

[45] Bret Victor. 2012. Learnable Programming. Designing a programming system for understanding programs. http://worrydream.com/#!/LearnableProgramming. [Online; accessed 19-January-2023].

[46] Julie Wagner, Mathieu Nancel, Sean G. Gustafson, Stephane Huot, and Wendy E. Mackay. 2013. Body-centric design space for multi-surface interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 1299–1308. https://doi.org/10.1145/2470654.2466170

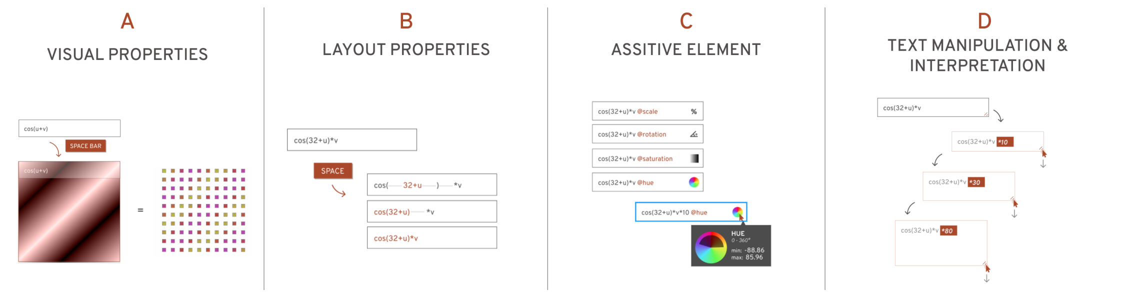
[47] David Ward, Alan Blackwell, and David Mackay. 2000. Dasher—a data entry interface using continuous gestures and language models. 129–137. https://doi.org/10.1145/354401.354427

[48] Adam Wathan, Jonathan Reinink, David Hemphill, and Steve Schoger. 2019. Tailwind CSS. https://tailwindcss.com. [Online; accessed 21-January-2023].

A APPENDIX: IDEAS GENERATED USING THE DESIGN SPACE

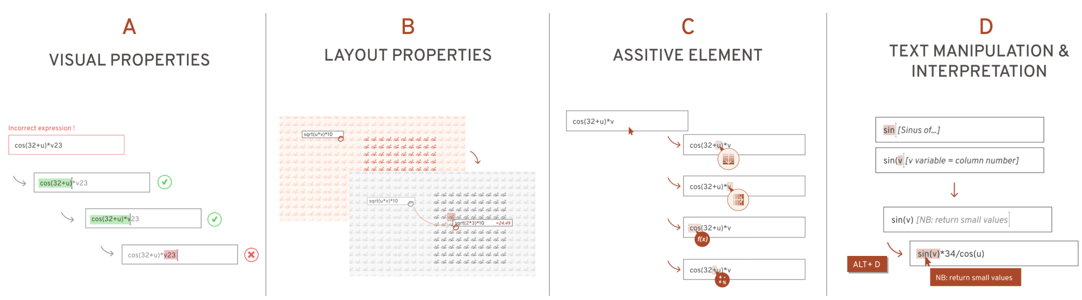
We give here a short description of each idea presented Figure 4.

A.1 Understanding the returned value and its resulting effect



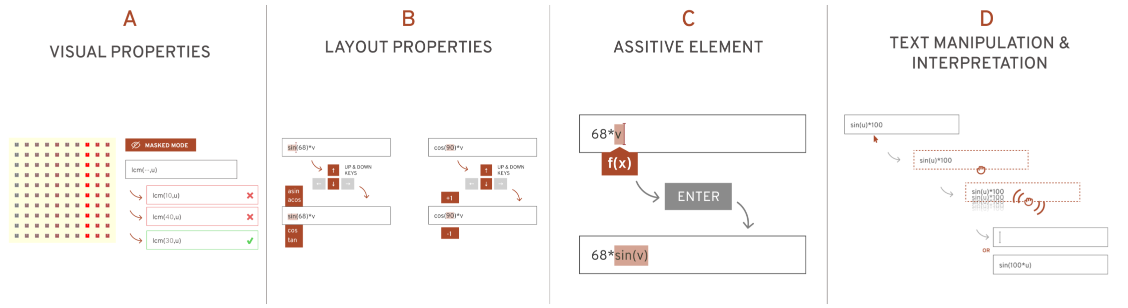
(1) 1.A - When the user presses the space bar inside the text field, the field is transformed to the same proportions as the pattern. Its background color is set to a color gradient that represent the normalized variation of the pattern.
(2) 1.B - When the user presses the space bar inside the text field, expression's blocks are grouped according to their evaluation priority. One by one following the evaluation order, the blocks are animated back to their original spacing.
(3) 1.C - The text field is accompanied by an icon that visually indicate the selected pattern property. When the user hovers over the icon, a tooltip opens with additional information such as the range of possible values and the unit used.
(4) 1.D - The text field is resizable and its height serves as a multiplier for the expression. When the user grabs the handle in the bottom-right corner, a multiplier value is added to the expression and the pattern changes dynamically.

A.2 Understanding the basis of the language



(1) 2.A - When the expression is not correctly written, users can debug quickly by isolating a part of the expression to evaluate. When selecting text, if the selected part has no syntax error the selection color will turn to green otherwise it will turn to red.
(2) 2.B - The text field is draggable over the pattern as a way to inspect any pattern element individually. When the user drags the field and drops it over a pattern element, a duplicate field is created, changing pattern variables names to their real values and showing the computed value.
(3) 2.C - When hovering over an expression term, the cursor icon changes to indicate the type of the hovered element (a variable, a function, an operator...).
(4) 2.D - Inline documentation is inserted inside the text field and appears while typing. This minimal documentation serves as a quick translation between English and mathematical symbols, as well as a reminder for variables. Users can edit the documentation by adding their own comments, that will also appear when hovering a block of expression.

A.3 Discovering and Exploring the language

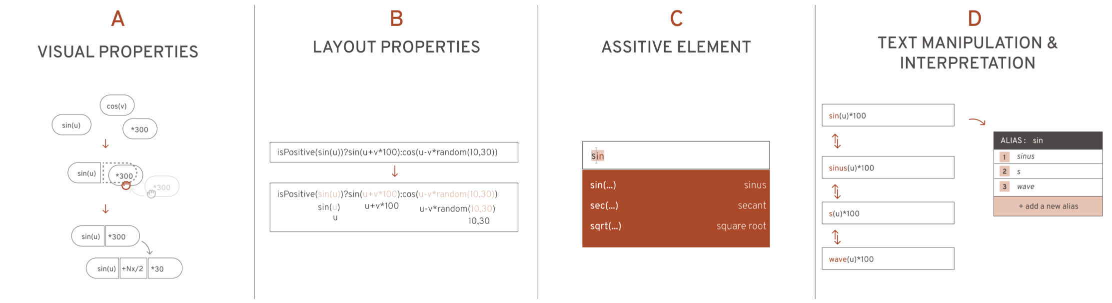


(1) 3.A - A special "masked mode" is available for users to try to find the missing character in a given expression. This mode serves as a way to discover new expressions and to train to remember the syntax.
(2) 3.B - Changing quickly between functions or increasing numbers can be done by pressing up and down arrows on the keyboard. The selected word is set depending on the cursor position in the expression. When pressing an arrow, a similar terms list will appear, allowing users to test quickly.
(3) 3.C - Whenever a variable is selected in the expression, a "random function" tooltip will appear. When pressing enter, a random function will be inserted around the selected variable.
(4) 3.D - The text field is "shakeable" to permute all the expression blocks between each others. When grabbing the field with the pointer and wobbling it around, the desired action is triggered. This interaction could also be used to erase text field content.

A.4 Helping with writing organization

(1) 4.A - Whenever the user has multiple fields in the page, the background color of the field is used to indicate the last modified field.
(2) 4.B - Text fields can be set as inactive in the page, used as a note taking zone rather than an expression zone. The activation is set by focusing the field, permitting users to quickly test multiples expressions by focusing different fields.
(3) 4.C - Text fields are accompanied by a drop-down menu that serves as a shared history panel. Whenever a pattern is shared, the new user has a trace of saved expressions from the original user, allowing a detailed re-appropriation of the pattern.
(4) 4.D - Expressions can be tagged using a hash symbol followed to any other characters. Once tagged, expressions can be searched using the hash as a trigger for the search list opening. Users can now organise their expressions to quickly use their favorites.

A.5 Making the writing process more usable



(1) 5.A - Bits of expressions can be moved around freely in the page, resulting in a kind of text field puzzle. Users can then drag fields together to create a complex expression from simpler parts.

(2) 5.B - When faced with a long and complex expression, users can activate a multi-line decomposition of it. The expression is then vertically indented using parenthesis as a delimiter.

(3) 5.C - The text field is accompanied with a drop-down list with an auto-completion function. The list also includes an English term for functions.

(4) 5.D - Users can edit aliases for any expression terms or blocks. When activating the alias mode, users can set an English word such as "wave" to a "sin" function for example. They can also link a short term to a full expression, e.g. using "f1" as an alias for "sin(v)*15". When writing in the field, when the correct alias is detected, a prompt appears asking for a potential replacement.