b5: A Web-Based Visual Programming Language

PEILING JIANG, New York University, USA

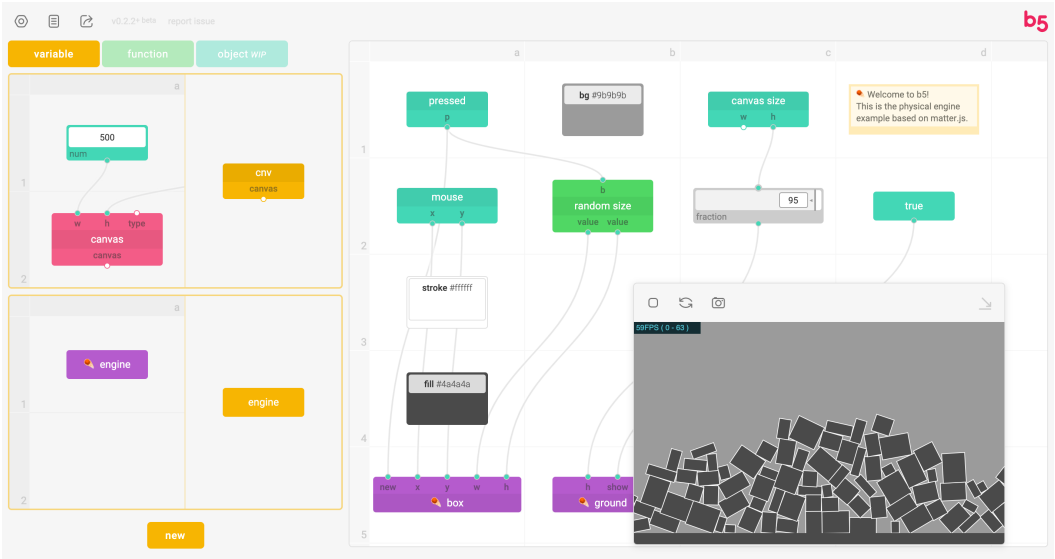

Fig. 1. The interface of b5 web editor, version 0.2.2.

How will the next generation learn creative coding? Do they have to look up hundreds of pages of documentation and take classes? Do they have to write redundant code for features already well accomplished due to lack of information? Do they have to learn English? b5 rethinks the future of creative coding in terms of clarity, accessibility, and customizability in a world that increasingly emphasizes remote collaboration, internationalization, and personalized learning. b5 is a web-based visual programming language for learning and fast prototyping for people with minimal to no programming experience, leveraging novel features compared to other visual programming languages, including (1) position-based sequential execution (blocks are executed in order based on their positions in the code canvas), (2) customized blocks (users can use the predefined blocks and the graphical interface to design and construct new blocks), and (3) effect blocks (block that affects other code blocks based on their contextual relationship instead of wire connection). b5 makes programming easy to learn and encourages programmers to explore and try, while more analysis is needed to assess its usability and long-term effects when used as the learning platform for beginners.

CCS Concepts: • **Human-centered computing** → *User interface toolkits.*

Additional Key Words and Phrases: visual programming language, creative computing

1 INTRODUCTION

Creative programming, using computational thinking to create expressive art and design projects instead of functional computer programs [6], is becoming a heated topic across the fields. Its history can be traced back to the 1960s [18], while the recent development of personal computers and web technology and the growing body of open-source software, learning resources, and community make it even more appealing, powerful, and accessible. Transforming from an avant-garde artistic tradition to a common practice in the era of "ubiquitous computing," its purpose, practitioners, and deliverables have changed - the goal is no longer only serving as a medium for artistic exploration and might be educational or auxiliary; the creators come from both fields of art and technology, who worked as designers, programmers, artists, etc., and may also be students in various levels of learning programs; the final display form can be graphics, installations (physical computing), films, projections, sound art, etc. Many programming languages, scripting tools, and visual programming interfaces emerge to meet the diverse needs of different groups of people.

Visual programming languages and interfaces gain their popularity among educators, designers, and beginners through this process. Benefited from the development of computer processing power and graphics capabilities, such tools become more accessible, powerful, and user-friendly than before [3, 4]. Using icons, images, pictures as a metaphor of functions and variables, which are common concepts in traditional text-based programming, people code by clicking, dragging, and dropping in a canvas of these graphical representations, instead of typing characters and words in the text editor.

Here, we propose b5, a web-based visual programming language that is designed to address several ignored problems and needs in the field, as well as proposing new features either as an extension or alternative solutions compared to the existing tools.

2 RELATED WORK

[Ingalls et al.] proposed Fabrik, a visual programming language leveraging "wires" and "nodes" as the basic components to create interactive projects. However, limited by the human-computer interaction devices and graphics power back then, the interface looks primitive and barely usable. The Scratch language and its programming environment were proposed by [Maloney et al.] and soon become one of the most popular tools for teaching and practicing programming in schools of different levels ranging from kindergarten to pre-college. The use of it in teaching shows positive results - students who learned Scratch could then spend less time to learn new topics, have fewer difficulties, and obtain higher levels of understanding when learning more advanced text-based programming languages like Python and C [2], indicating positive cognitive-level assistance being provided by the interface so that students can better learn and comprehend concepts in programming in general. [Ruf et al.] also showed that learning Scratch helped students to perform better in later coding challenges with a higher intrinsic motivation to learn novel concepts. The visual programming languages not only outperform traditional programming languages in terms of educational purposes but also accessibility (learning curve and hardness of understanding) and can be specifically designed to better serve in particular domains and aid people with minimal or no coding experience to leverage for computational tasks. For example, Grasshopper [15], a plugin for Rhinoceros 3D, were designed to help complete computational industrial and architecture design. The target users of it are designers and architects who are supposed to have no programming experience at all. Another example is Max [1] designed for composer, performers, and artists to create music and multimedia projects. [McNerney], on the other hand, pushed the boundary of visual programming further into the physical world where people can use real LEGO bricks as building blocks to design their programs.

Because of different purposes, target users, and generalization capabilities compared to text-based programming languages, various visual programming languages and the platforms to create and edit them emerge in fields where there is a computation need, result in tons of different choices [7] and outnumbers the text-based ones.

3 DESIGN

Visual programming languages can be divided into two subcategories: node-based, e.g., Grasshopper [15], where nodes are separated and connected by wires in-between, and the execution order of the nodes is solely based on the connection between them, from root to edge; and block-based, e.g., Scratch [14], where blocks of lines of code are placed together sequentially, and the execution order of the blocks is thus determined by the contextual position of the block. The latter is more closely related to the traditional text-based languages with a strong emphasis on sequence.

b5, however, tries to take advantage of both designs - using a node-based layout, while still emphasizing the execution order by constraining the position of the "nodes" (or in the context of b5, blocks) into the code canvas with a grid layout. The blocks then run sequentially from left to right, in each line of the canvas, and line by line from top to bottom. In the context of b5, we call each individual code block a "block," and its ports that send outputs or receive inputs "nodes."

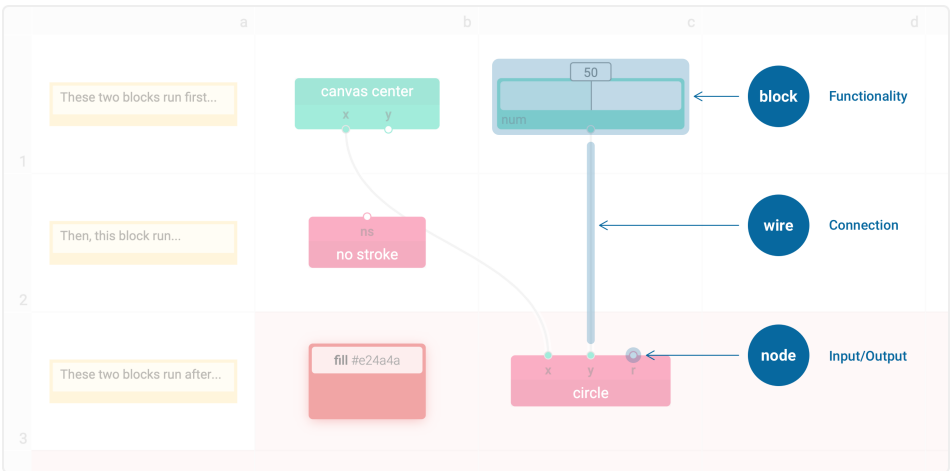

Fig. 2. The basic structure of a b5 program. b5 uses a node-based design where each code block is separate and connected by the wires in-between for the data to flow from the upper blocks to the lower ones. However, unlike other node-based interfaces, b5 uses a grid layout to constrain the position of the blocks in the code canvas (the canvas where all the code blocks are placed) for users to define the only possible execution order of all code blocks. In this figure, the *circle* block is taking outputs from the *x* node of *canvas center* block into its *x* node and from the output of *number slider* block into its *y* node. However, with the sequential execution design (from left to right, from top to bottom), the order that the blocks in this code canvas run is *canvas center*, *number slider*, *no stroke*, *fill*, and *circle*, even though the style blocks do not have direct wire connections with the *circle* block.

Since all the blocks are sequence-sensitive and always aware of their positions in relates to the others, a novel characteristic is introduced for the *effect blocks*. Effect blocks can affect other blocks around based on their position relationships, e.g., all the following blocks in the code canvas or all the other blocks in the same line, instead of the wire connections. This feature is inspired by HTML canvas rendering context whose style settings can affect the following shapes drawn on

the canvas. In figure 2, the *fill* block is an effect block that sets the filling color for the following shapes cascadingly, till another *fill* block is encountered. When an effect block is selected and focused, the background grid cells will also change color to reflect its effective range, unlike when working with text-based canvas rendering context mentioned above, the underlying status of the drawing context always remains hidden to the programmer and needs to be inferred from the actual behavior of the program. This extra layer of information is supposed to help programmers gain a deeper understanding of their programs and debug or reconstruct them more easily.

We created a web-based programming interface to help users create, edit, and read b5 code files. As shown in figure 3, the whole system consists of two major components: a floating viewer window and an editor interface. The viewer window shows the live rendering of the program and can help snapshot or record the canvas for sharing. The editor, the programming interface, can be further divided into *Factory* and *Playground* sections. Users can construct their customized *variable*, *function*, and *object* blocks (See section 3.3.) in the panels of the Factory section, which can be later used in the code canvas in Playground. Playground has only one mega code canvas that behaves similarly to the *draw* loop of p5.js [8] and runs 60 times per second by default.

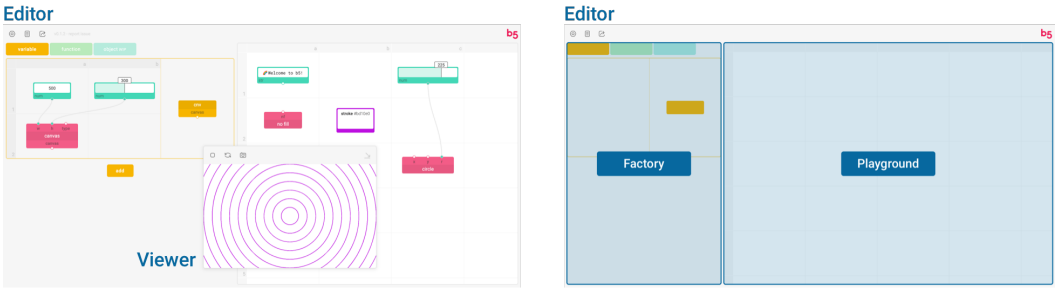

Fig. 3. The web-based programming interface of b5. The environment has a floating viewer window and an editor interface, which consists of two sections, *Factory* and *Playground*.

The following subsections will elaborate how other design choices were made to enhance b5's clarity, accessibility (e.g., understandability), and customizability.

3.1 Clarity

Besides the sequence-sensitive design in b5 that forces people to learn and think more about flow, which would benefit them when programming in more advanced languages, several other features make the interface of b5 clear and easily comprehensible.

Like many other visual programming languages, b5 is colorful, and each of the block types has its color. The color is consistent across the system to the connecting nodes and other buttons to further strengthen users' muscle memory and could help increase their ability to recognize the overall structure of the program. Just as shown in figure 4, *draw* blocks, e.g., and *canvas*, are pink, *default* function blocks, e.g., *constrain* and *map*, are gray, etc. Besides different block types, different blocks also have different designs, i.e. forms and structures, to best serve their particular functionalities - some may only take inputs and outputs, some have a slider or input box component to interact with, and some organize all the information into one line for simplicity.

Fig. 4. Different colors of different block types.

The blocks' wires and input/output nodes are also designed to precisely indicate their position, identity, and connecting status. The nodes are white by default and can switch to different colors (See figure 5.) when connecting to other blocks as the input or output hooks, indicating the data flowing through them. It may also turn alarming red, along with the connecting wire, when there is an error related to the current connection, e.g., the input node expects a number input while a shape is fed.

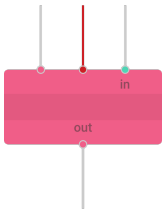

Fig. 5. Positions, identities, and connecting status of the nodes in a code block.

Instead of waiting for compilation and the code changes reflected on the canvas, b5 enables users to modify scripts on the fly and any changes made, additions, deletions, or parameter tweaks, will be instantly visible on the side of the code. Thanks to this live rendering feature, users are encouraged to make changes and explore different possibilities of the code and, as a result, gain a deeper understanding of the structure and functionalities of each code block through the effortless tries. Like some other visual programming languages that also features live rendering, b5 could encourage learning code by exploration and help form certain programming habits - that they are no longer afraid of trying and tweaking the code - which would benefit the learners later when they learn more advanced and harder topics [17].

To conclude, features including sequential specific, colorful blocks, status-sensitive nodes and wires, and live rendering help make b5 and its web-based programming environment clear and reduce ambiguity that might confuse new users when they first see and use the system with minimal guidance and teaching.

3.2 Accessibility

Another major design concern is accessibility in terms of the learning curve and the hardness of understanding, especially for people with limited educational resources.

All the code documentation (what a particular predefined function does and how to use it) is directly embedded in its component, whether it's a block, a node, or some other interactive code components, e.g., an input box or a slider. One can intuitively look it up by hovering the mouse on the corresponding component, instead of opening another browser window and search for the component. The documentation popup includes all the information one would need to understand the functionality and use cases of a block, including the full name that better depicts the identity, a detailed description explaining its function and default values (for nodes), and the type (e.g., block or node) and category (e.g., *draw* or *object* block) of this component.

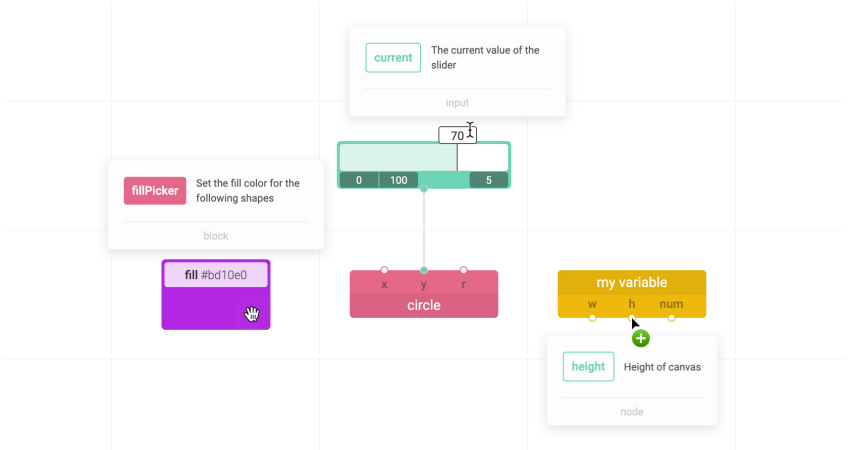

Fig. 6. Embedded documentation for the blocks, nodes, and other components. When hovering on a code block, the full name, description, and categorization will show. The feature also applies to other components in the code canvas, including the nodes, the input boxes, and more.

Since the text is just labels instead of the commands themselves, the blocks, node names, and even embedded documentation can be easily translated into another language to increase the accessibility of the system. Scratch has been translated into more than 70 different languages [5], and b5 is expected to have the same flexibility for better internationalization compared to text-based languages. People from over the world would then be able to learn computation without learning an additional language as a prerequisite.

Fig. 7. The same *camera* block translated into English, Chinese, French, and Japanese. Both the block name and node names are translated.

Finally, b5 editor, the whole programming environment, is web-based and can be loaded and ready to work swiftly, and the only requirement is a stable internet connection. The whole process of creating, editing, and reading b5 programs does not require any additional installation. There is also no hardware or software requirement - one can even easily use it on a tablet or mobile device. The b5 definition file used to store the type, location, and connection of the blocks is in JSON (JavaScript Object Notation) format and can be opened and read by almost all modern operating systems and web browsers. No private-domain extensions are associated with the files. Thanks to the nature of b5 and its coding platform, the system is highly accessible and straightforward to work with.

The easily accessible embedded code documentation, the ability to be multilingual, and the web-based nature all help make b5 and its programming interface highly accessible and easy to work with, even without a trained teacher for this novel system, without learning English, and without any software purchase or installation.

3.3 Customizability

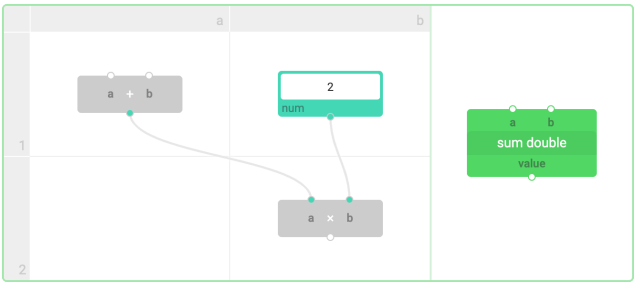

Fig. 8. The interface for designing and constructing customized blocks. A code canvas and the rendering of the customized block.

In b5, instead of only using the predefined blocks, programmers can easily build their own customized blocks, including variables, functions, and objects, within the Factory panel, just like using the default commands and functions to write a new *function* or define a new *object* in a text-based language. A live preview of the customized block is placed side by side with the mini code canvas where users can add code blocks as they did in the main code canvas in Playground.

The Factory panel has three tabs in which users can construct three types of customized blocks: *variable*, *function*, and *object*. The *object* type has not been implemented and the many details are yet to be finalized. The *variable* blocks do not take any inputs (all the input nodes from the blocks used for construction will be eliminated) and their output values are calculated once during their definition before the first call to the Playground canvas and are fixed even when the blocks are used for multiple times in the Playground canvas - just like a variable. The whole *variable* tab can be regarded as the *setup* function in p5.js [8], except that it also produces instances that can be directly referred to in the *draw* loop (the Playground code canvas).

The *function* tab, however, works like defining functions. The customized blocks will not be called or calculated unless they are placed in the code canvas in Playground and are able to take inputs to calculate for different output values for each time different input values are fed in - just like a function.

To avoid circular definition (the definition of a customized block has itself in its defining canvas), the customized blocks can only be used in the mega code canvas in Playground.

4 WORKSHOP

To test the usability of the system, we set up a workshop with three participants (2 males, 1 female, all college students, 2 majoring in design, and 1 majoring in psychology, labeled as P1 to P3) who had no programming experience at all and did not know anything about b5 before. The workshop was conducted in Chinese and used the English version of the interface.

To get the participants to use the interface and program in b5, a follow-along tutorial to build a bouncing ball sketch (a circle bouncing around the canvas) was given. A few qualitative questions were asked and discussed after the tutorial, including:

(1) How do you like b5? Are you interested in continuing learning it to build more complex programs in the future?
(2) Do you find the interface confusing? Is so, which parts confused you?
(3) Are you confident that you can build the example again by yourself from scratch without any external help and hint?

(4) Do you understand the core concepts of this programming language and its interface, including the Factory and Playground structure, the sequence, different block types, the nodes and their connecting wires? How would you explain them to people with no experience in b5 and any other type of programming?
(5) Use one word to describe your thoughts on b5 and your experience.

We received a lot of positive feedback. Participants all agreed that b5 was easy to understand and use, with little difficulty comprehending the basic ideas, including the structure and the sequential design. P1 said that the interface made them believe that they can program, and coding was no longer scary. Design students compared the interface to Adobe Photoshop and Illustrator and agreed that the visual interface was more intuitive to learn and work with as all the options, features, and functionalities were visually available for reference and use. P3 said that the system was user-friendly and could be a good alternative for kids to learn programming or logical thinking in general. Other endorsements were given to features including the translation (verbally described to the participants) and embedded documentation. When asked to use one word to describe their thoughts on b5 or the overall experience, participants, from P1 to P3, used *fun*, *fast*, and *cute*.

Several concerns and suggestions were also brought up by the participants in the workshop.

(1) The block names may be hard to memorize. Currently, there is no browsing system to view all the blocks in a well organized and classified way, and the only way to find and add a new block is to use a search bar to search for the name, type, or description of the block, which does require a basic knowledge of the existing blocks.
(2) Factory is confusing. One participant was confused by the Factory and Playground structure and could not understand their relationship, executing order, and differences and use cases between the *variable* and *function* sections. Extra explanation was needed for them to fully understand this concept.
(3) Sequence-sensitivity is not indicated by any means from the interface. One of the major differences of b5 and other visual programming languages is the execution sequence being determined solely based on the position instead of the connection of the blocks. However, no visual cue is currently provided to inform the users of this design. Programmers need to either be told or try to re-position several blocks to find it out, and possible confusion may occur during this process.

All the feedbacks listed above are valid concerns regarding b5 and the current implementation of the programming environment and need to be addressed in future iterations to provide a better user experience. To better assess the acceptance rate, efficiency, learning curve, and long-term benefit of using b5 as the primary learning tool for beginners, more qualitative and quantitative analysis needs to be done.

5 LIMITATIONS AND FUTURE WORK

Several issues are to be addressed in the future. There is currently no block browsing system for users to see all the blocks, organized and classified, limiting the exploration for new users who have little knowledge of the available blocks. The novel Factory and Playground composition, as a graphical metaphor of the *definition - execution* structure in traditional text-based programming languages, however, needs a better design to inform the new users. The same as the importance of block position in b5 and its relationship to the execution order compared to other visual programming languages that more visual or audio cues need to be provided as a part of the interface. Is the current design and implementation optimized and understandable to programming beginners? To answer this question, we need more qualitative and quantitative analysis.

For the future works, two major pathways to extend the current work are:

(1) Sharing and collaboration. Sharing a well-wrapped block is much easier and visually more intuitive than sharing lines of code. A feature that enables users to share their customized blocks just like using Airdrop would greatly enhance current code sharing and collaboration experience. Users may also be able to browse the blocks designed and programmed by others in a marketplace and load then into their own projects easily.
(2) Modularization. b5 and its programming environment, the web editor, are initially designed for creative coding projects, while can also be modularized and extended to help non-programmers accomplish computational tasks in other domains, e.g., programming to customize and control smart home devices. As the blocks, in the backstage, have an extremely simple API for definition and execution, other developers and companies can use this system to create their own blocks to serve others needs in their domain of interest.

Community is also vital to the success, and many issues related to building a lively and supportive community are yet to be explored, e.g., on-boarding new developers and outlining contribution guidelines and code of conduct. These are not directly related to the design and development of b5 itself, but would define the future of this new programming platform.

6 CONCLUSION

How will the next generation learn creative coding? Do they have to look up hundreds of pages of documentation and take classes? Do they have to write redundant code for features already well accomplished due to lack of information? Do they have to learn English? We designed and implemented a novel visual programming language, b5, and its web-based programming environment to address these questions. The name of b5 was inspired by p5.js [8] developed by Processing Foundation and developers from the open-source world, which also inspired many other concepts from creating a canvas to the Factory and Playground structure. b5 is a web-based visual programming language for learning and fast prototyping for people with minimal to no programming experience, leveraging many novel features compared to other visual programming languages and text-based languages like Python: (1) b5 is sequence-sensitive. There is only one possible global execution sequence for all the code blocks determined by their relative positions in the code canvas. The positions of the blocks play an essential role, just like the order of the lines when programming using text-based languages, and programmers are forced to think more about the sequence and flow. (2) Instead of only using pre-defined blocks, users can also use those blocks to design and construct their own customized blocks as new *variables*, *functions*, or *objects* and use them in the Playground code canvas to increase the flexibility and efficiency of their programs. The ability to define a function or wrap a group of commands into one that saves typing and computing is critical to more advanced programmers, and through this design and structure, a similar set of skills are supposed to get trained. (3) Due to the sequence-sensitive characteristic of b5, a new type of block, effect block, that affects other blocks by relative positioning instead of wire connection, is defined. The grid interface of the code canvas is also designed to help dynamically reflect the effective range of those blocks. Effect blocks reveal the underlying information, e.g., the drawing context settings, that usually remain hidden to the programmers, and help them understand the code better with this extra layer of information. The first two features were inspired by text-based languages, while the last one may help improve other visual and text-based languages to be more understandable and less ambiguous.

In the workshop, our current implementation received positive feedback from participants with no programming experience. All participants agreed that this novel programming language and its editing interface are easy to use and understand.

b5 rethinks the future of creative coding by emphasizing clarity, accessibility, and customizability in a programming system. With features including position-based sequential execution and customized blocks, b5 helps non-programmers learn computational thinking and fast prototype creative coding projects. While more analysis needs to be done to assess its usability and efficiency, b5 has received various positive feedback on its major innovation and novel features. The programming language and its web-based environment will keep evolving and iterating towards a functional tool that can be used for serious and critical production works.

ACKNOWLEDGMENTS

This is the capstone research project in support of candidature for my B.F.A. degree in Interactive Media Arts. This project is partially funded by NYU Tisch School of the Arts Undergraduate Creative Research Fund. After five years of undergraduate studies, this project well represents many of the fields that I have learned, practiced, and tried: industrial design, integrated innovation, computational design, interactive media, computer science, and psychology. Many thanks to everyone who has guided, helped, supported me, and shaped my academic path towards the one that enables the accomplishment of this project: my parents, professors, friends, and classmates. More specifically, I want to say thank you to Tao Jiang, Yongjun Zeng, Xingqiong Chen, Keru Wang, Daniel Shiffman, Daniel Rozin, Brandon Clifford, Xiangyang Xin, Denis Pelli, Weiji Ma, David Robert Wallace, Lin Zou, Enguo Cao, Eugenio Altieri, Yan Chen, Matt Romein, Katherine Dillon, Allison Parrish, Youxin Wu, Shawn Van Every, Linghao Zhang, Mimi Yin, Randi Williams, Arnav Kapur, Yiqing Liu, Wei Dai, Xintian Li, Zhongchao Tang, Xinyue Wang, Yuanda Xu, Yanan Wang, Jin Dou, Glenn Fernandes, Cezar Mocan, Lydia Jessup, Ruixuan Li, and Jinshuo Huang.

Thank you!

REFERENCES

[1] Cycling '74. 2020. Max. cycling74.com/products/max/

[2] Michal Armoni, Orni Meerbaum-Salant, and Mordechai Ben-Ari. 2015. From scratch to "real" programming. *ACM Transactions on Computing Education (TOCE)* 14, 4 (2015), 1–15.

[3] Marat Boshernitsan and Michael Sean Downes. 2004. *Visual programming languages: A survey.* Computer Science Division, University of California.

[4] Margaret M Burnett and David W McIntyre. 1995. Visual programming. *COMPUTER-LOS ALAMITOS-* 28 (1995), 14–14.

[5] Wikipedia contributors. 2007. Scratch (programming language). https://en.wikipedia.org/wiki/Scratch_(programming_language)

[6] Wikipedia contributors. 2021. Creative coding. https://en.wikipedia.org/wiki/Creative_coding

[7] Wikipedia contributors. 2021. Visual programming language. https://en.wikipedia.org/wiki/Visual_programming_language

[8] Processing Foundation. 2013. p5.js. https://p5js.org/

[9] Paul E Haeberli. 1988. ConMan: A visual programming language for interactive graphics. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. 103–111.

[10] Dan Ingalls, Scott Wallace, Yu-Ying Chow, Frank Ludolph, and Ken Doyle. 1988. Fabrik: a visual programming environment. *ACM SIGPLAN Notices* 23, 11 (1988), 176–190.

[11] Anastasia Kovalkov, Avi Segal, and Kobi Gal. 2020. Inferring Creativity in Visual Programming Environments. In *Proceedings of the Seventh ACM Conference on Learning@ Scale*. 269–272.

[12] Paul Lyons, Giovanni Moretti, and Chrissy Reeves. 2001. Some possibilities of visual programming languages. In *Proceedings of the Symposium on Computer Human Interaction*. 43–47.

[13] Matthew B MacLaurin. 2011. The design of Kodu: A tiny visual programming language for children on the Xbox 360. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 241–246.

[14] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 1–15.

[15] Robert McNeel and associates. 2007. Grasshopper. grasshopper3d.com

[16] Timothy S McNerney. 1999. *Tangible programming bricks: An approach to making programming accessible to everyone.* Ph.D. Dissertation. Massachusetts Institute of Technology.
[17] Orni Meerbaum-Salant, Michal Armoni, and Mordechai Ben-Ari. 2011. Habits of programming in scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education.* 168–172.
[18] Monoskop. 2012. Compos 68. https://monoskop.org/Compos_68
[19] Alexander Ruf, Andreas Mühling, and Peter Hubwieser. 2014. Scratch vs. Karel: impact on learning outcomes and motivation. In *Proceedings of the 9th workshop in primary and secondary computing education.* 50–59.

A CODE AVAILABILITY

b5 is fully open-sourced.

The source code of the language and the web-based interface can be found at https://github.com/peilingjiang/b5. The web editor is hosted at https://b5editor.app/ and is automatically built based on the latest version of the code. Due to limited time, this version of the paper may contain typos and errors and may be modified and updated in the future. Future updates and more information about this project can be found at the links above and https://jpl.design/b5.

B EARLY SKETCHES

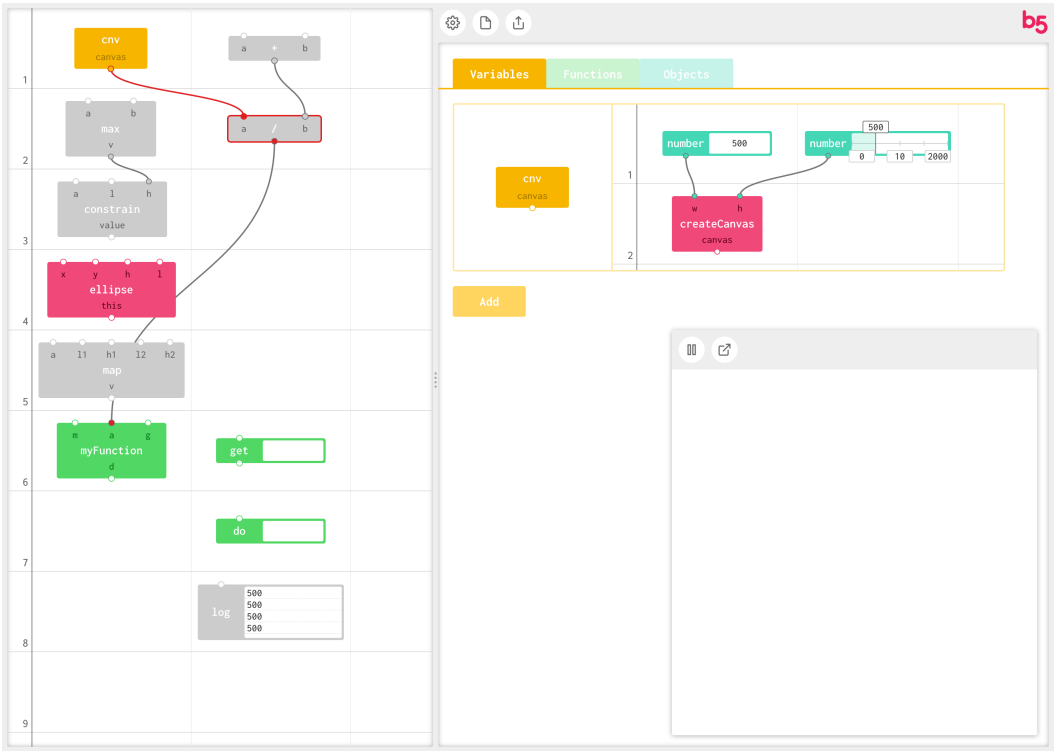

Fig. 9. The early sketch of the interface, including the structure, code canvas, blocks, and wire connections.

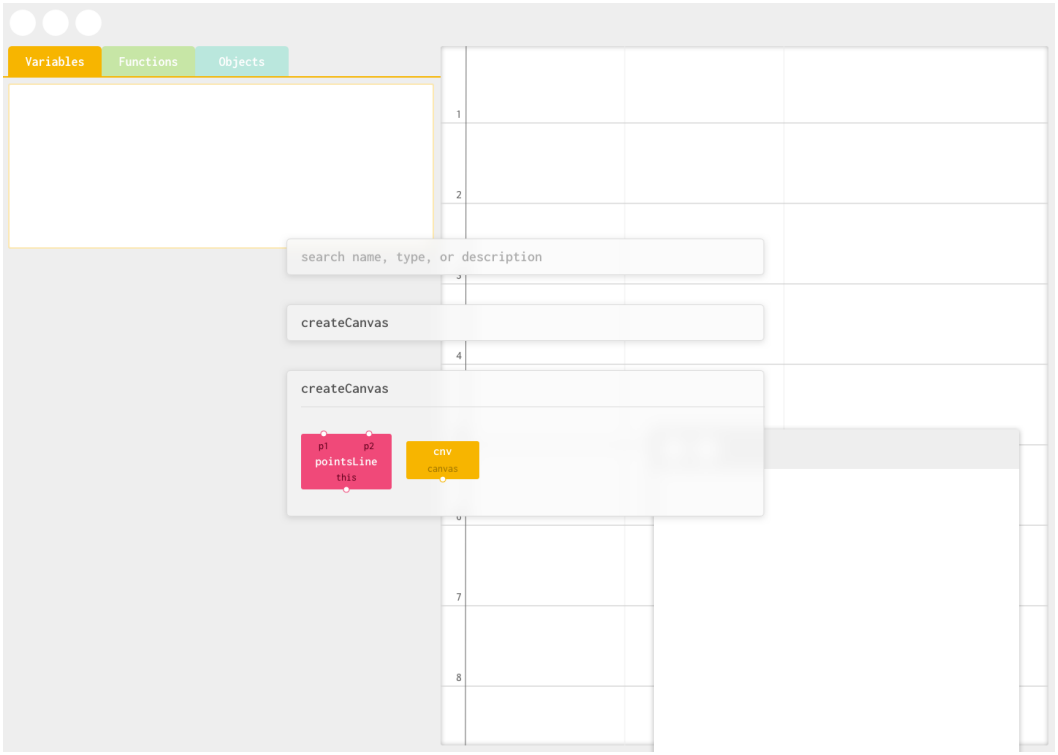

Fig. 10. The early sketch of the block search interface.

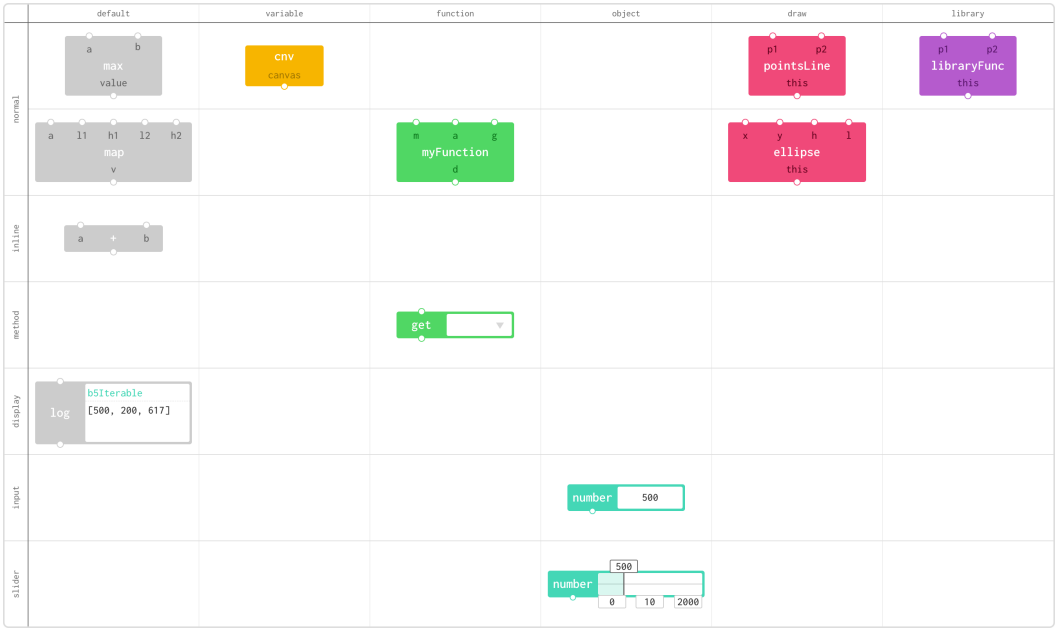

Fig. 11. The early block designs classified by the type and form.