

**Study on the Learning Effects Based on
Characteristics in Programming Learning
Environments for Novice Learners**

初学者向けプログラミング学習環境の特性に基づく学
習効果に関する研究

February 2018

Waseda University

Graduate School of Fundamental Science and Engineering

Department of Computer Science and Communications Engineering,
Research on Reliable Software Engineering

Daisuke SAITO

齋藤 大輔

ABSTRACT

Novice learners are often children aged 6 to 12 who are inexperienced with programming learning. They tend to use programming learning environments when learning to program. Programming learning includes computer science learning and mathematical learning. It is also used to develop problem-solving and abstraction abilities.

Each programming learning environment has unique characteristics. In this research, I identify the characteristics of the programming learning environments and investigate the learning effects based on these characteristics.

As novice learners utilize numerous programming learning environments, I initially investigated the kinds of programming learning environments. A Google Custom Search API with specific keywords yielded 800 search results. Then I extracted the programming learning environments by morphological analysis and visual observations, which resulted in over 70 environments for programming learning. Examples such as Scratch, Alice, and Greenfoot are used in a visual programming language, while CodeCombat and Minecraft Education Edition exist in game software.

Previously, Kelleher et al. classified multiple programming environments, demonstrating that these environments have unique characteristics. However, environments, including learning environments, continue to be developed. Several studies have demonstrated the learning effects in programming learning environments. Some have shown that the learning environment called Scratch is suitable to improve learners' interest and passion for programming. Others have revealed that using a game called Minecraft tends to improve programming skills. These studies suggest that the learning effects may depend on the programming learning method and the learning environment. However, these environments are used at the discretion of educators and learners. Moreover, it is unclear what kinds of learning

effects are derived from the characteristics of each learning environment. Investigating the characteristics of learning environments should reveal the learning effects. Considering the programming learning environment for novice learners, my research investigates the learning effects based on characteristics. This can be used to maximize the learning effects of novice learners.

The main research question is, “How can novice learners maximize learning effects in programming learning?” The goal of this research is to clarify the learning effects by grasping the characteristics of the programming learning environment because it should improve learning of novice learners.

Chapter 1 highlights that there are over 70 kinds of programming learning environments. This diversity leads to issues with programming learning environments. Additionally, I explain the research outline and research goals.

Chapter 2 describes the taxonomy to evaluate multiple programming environments and the classification results based on the taxonomy. The taxonomy is created by defining items to classify programming learning environments using Kelleher et al. as a reference. Specifically, I optimize Kelleher’s table for learning environments and add a new category. The taxonomy table divides the 56 items into 11 categories.

Then I apply the taxonomy to classify several programming learning environments. Based on the results, the characteristics of each environment, including the attributes of visual programming language environments and game software environments, are evaluated. I survey 43 kinds of environments with an emphasis on visual languages and software that works alone on PCs or similar devices to create a taxonomy table for programming learning environments. The proposed table can evaluate and compare such environments. An experiment confirms that the classification and evaluation results are independent of the evaluator.

Therefore, this classification table helps users (learners and educators) identify the characteristics of a programming learning environment.

Chapter 3 investigates learning effects as a function of characteristics in the same environment. Herein the differences between visual and text input methods (Representation of Code and Construction of Programs) are investigated in the same Lua programming environment to determine if the input method influences the learning effects. Although many visual and text comparative studies have been conducted, investigations including characteristics such as linguistic representation are scant. These differences in characteristics should impact the learning effects. Specifically, I compare a combination of text (Representation of Code) and typing code (Construction of Programs) with a combination of image (Representation of Code) and drag-and-drop (Construction of Programs). The results indicate that a visual input method is better suited for a novice programming learning. However, the comparison results suggest that actions change the learning effects. Hence, the text input method can be used for programming learning of novice learners from the viewpoints of the representation of code and construction of programs in a programming environment.

Chapter 4 investigates the characteristics and learning effects of multiple environments. This chapter considers the learning effects based on the characteristics of programming constructs and game elements as well as the characteristics discussed in Chapter 3. I conduct a quantitative evaluation by a workshop on six programming learning environments. The characteristics of the classification influence the learning effects. However, if the software involves "physical objects" and "assembling physical objects," the learner may become bored as the workload increases. The three groups (visual programming language, game software, and physical environment) show a difference in attitude toward programming. A visual programming language tends to reduce programming difficulty. Although environments with game elements tend

to increase fun, they also increase the perceived difficulty of programming.

Chapter 5 summarizes this thesis and explains future research. Future research will focus on three main areas: to propose and create a programming learning environment, to optimize the characteristics and functions of the taxonomy table, and to create guidelines to select the appropriate programming learning environment. One project that I am proposing is to develop an environment to predict the learning effects from characteristics. This environment would be an extension of the environment that I tried to develop to consider learning effects. In this work, only partial environments or prototypes are implemented. Currently, I am working on expanding the function of this environment.

ACKNOWLEDGMENTS

I would like to thank Prof. Hironori Washizaki for his considerable guidance in advancing this research and thesis. I would like to recognize Prof. Yoshiaki Fukazawa (Waseda University), Prof. Tatsuo Nakajima (Waseda University), and Dr. Tsuneo Yamaura (Tokai University, Former Associate Professor of Tokai University) for their cooperation in writing this thesis.

I would like to express my gratitude to Mr. Yusuke Muto (Fuji Television Kids Entertainment) and Mr. Akira Takebayashi (TENTO) for their great cooperation in this research. I would also like to acknowledge Mariko Tamura (D2C) and Mr. Toshihisa Nishizawa (Denno-Shokai) for their cooperation. In addition, I would like to thank all the companies and individuals who supported with this research.

I would like to thank Ms. Ayana Sasaki of the Washizaki Laboratory for preparing for the experiment and promoting this research as well as all members of the laboratory. Lastly, I would like to thank my family for supporting my research and thesis writing.

Table of Contents

ABSTARCT.....	II
ACKNOWLEDGMENTS	VI
Table of Contents	VII
List of Figures	X I
List of Tables.....	X III
1. INTRODUCTION	1
1.1. Programming Learning	4
1.2. Programming Learning Environments	4
1.2.1. Method to identify Programming Learning Environments	5
1.3. Contributions	7
1.4. Organization of This Thesis	7
2. CLASSIFICATION OF PROGRAMMING LEARNING ENVIRONMENTS	8
2.1. Background.....	8
2.2. Creation of Taxonomy	9
2.2.1. Taxonomy details.....	11
2.3. Selection of Environments to Classify.....	12
2.4. Results and Analysis of Classification by Taxonomy	14
2.4.1. Analytical method.....	14
2.4.2. Overall results	14
2.4.3. Results of each attribute.....	18
2.5. Discussion	20
2.6. Related Works.....	21
2.7. Limitations	21
2.8. Conclusion.....	22

3. COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS IN PROGRAMMING METHOD.....	23
3.1. Background.....	24
3.1.1. Programming learning for novice learners.....	24
3.1.2. Two input methods.....	24
3.1.3. Minecraft and ComputerCraftEdu with Programming Learning 26	26
3.2. Workshop Design.....	28
3.3. Experiments.....	29
3.3.1. Participants.....	30
3.3.2. Questionnaire.....	30
3.3.3. Analysis Method.....	31
3.4. Result.....	33
3.4.1. Attitude Toward Programming.....	33
3.4.2. Understanding Programming.....	39
3.5. Discussion.....	42
3.5.1. Result of RQ3-1.....	42
3.5.2. Result of RQ3-2.....	44
3.6. Limitations.....	45
3.7. Conclusion.....	46
4. QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF PROGRAMMING LEARNING ENVIRONMENTS ...	48
4.1. Background.....	49
4.2. Programming Learning Environments.....	50
4.2.1. Scratch.....	50
4.2.2. Viscuit.....	51
4.2.3. CodeMonkey.....	52

4.2.4. Lightbot	53
4.2.5. OSMO Coding	54
4.2.6. Robot Turtles	55
4.3. Classification.....	56
4.4. Experiments	58
4.4.1. About Experiments.....	58
4.4.2. Questionnaire and test.....	58
4.4.3. Learning comprehension test	58
4.4.4. Questionnaire about the attitude toward programming	60
4.5. Workshop	61
4.5.1. Schedule of the workshop	61
4.5.2. Number of students and effective questionnaire responses	61
4.6. Results and Analysis	62
4.6.1. Learning comprehension test	62
4.6.2. Attitude toward programming	65
4.6.3. Comparison of the characteristics in individual Environments 68	
4.7. Discussion	78
4.7.1. Answer of RQ4-1	78
4.7.2. Answer of RQ4-2	78
4.7.3. Answer of RQ4-3	79
4.8. Related Works	80
4.9. Limitations	81
4.10. Conclusion	81
5. CONCLUSION	83
5.1. Summary.....	83
5.2. Future research.....	84

5.2.1. Propose and Create a Programming Learning Environment	.86
5.2.2. Other future research	89
REFERENCES	92
RESEARCH ACHIEVEMENT	101
Journals	101
International Conferences	101
Domestic Conferences	103
Lectures	103
Books	104

List of Figures

Figure 1-1. Outline of this research	3
Figure 2-1. Result of Classification	18
Figure 3-1. Two input method (Visual).....	27
Figure 3-2. Two input method (Text).....	27
Figure 3-3. Comparison of input methods.....	28
Figure 3-4. Analysis Method	32
Figure 3-5. Result of VG	34
Figure 3-6. Result of TG.....	34
Figure 3-7. Six problems Response Rate	41
Figure 3-8. Result of Free Problem	42
Figure 4-1. Scratch [71]	50
Figure 4-2. Viscuit [41].....	51
Figure 4-3. CodeMonkey [42].....	52
Figure 4-4. Lightbot [14].....	53
Figure 4-5. OSMO Coding [43].....	54
Figure 4-6. Robot Turtles [44].....	55
Figure 4-7. Question Example	59
Figure 4-8. Free description problem.....	60
Figure 4-9. Results of Visual Programming	63
Figure 4-10. Results of Game Software.....	63
Figure 4-11. Results of Physical Environment	64
Figure 4-12. Results of the Free Description Problem.....	65

Figure 4-13. Results of visual programming language.....	67
Figure 4-14. Results of game software.....	67
Figure 4-15. Results of the physical environment	68
Figure 5-1. Future Research	85
Figure 5-2. MakeCode [58] for MEE.....	87
Figure 5-3. Python Environments for MEE	88
Figure 5-4. Basic specifications	88
Figure 5-5. Comparison of source code.....	89
Figure 5-6. About Guideline.....	90

List of Tables

Table 1-1. Google Keyword Search	5
Table 1-2. Programming Learning Environments List	6
Table 2-1. Taxonomy Table (Optimized Kelleher's [11])	10
Table 2-2. Environments survey list	13
Table 2-3. Classification Result	17
Table 2-4. Crosstabulation with game elements	19
Table 2-5. Attribute Table	20
Table 3-1. Problem Contents	29
Table 3-2. Questionnaire	31
Table 3-3. Result of the Shapiro-Wilk test	32
Table 3-4. Result of Arithmetic average (A1)	38
Table 3-5. Result of A2 - A5	39
Table 3-6. Description formula questionnaire result	42
Table 4-1. Classification Result	57
Table 4-2. Results of The Significant Difference Test (Learning Comprehension Test)	64
Table 4-3. Results of The Significant Difference Test (Attitude Toward Programming)	68
Table 4-4. Feature Table of the Environments	75
Table 4-5. Analysis of test results	76
Table 4-6. Analysis of attitude questionnaire	76
Table 4-7. Analysis of the learning effects	77

CHAPTER 1

INTRODUCTION

This chapter discusses issues in programming learning and programming learning environments. Novice learners often use programming learning environments when learning to write code. In this thesis, novice learners are defined as children aged 6 to 12 who are inexperienced in programming. Programming learning environments have diverse characteristics. In this research, I identify characteristics of programming learning environments and investigate the learning effects based on the characteristics.

Figure 1-1 shows how my research is related to previous research on programming environments. Although many studies have employed programming environments, the learning effects of each learning environment are unknown.

The research question, solution, and goal of this research are as follows:

1. Research Question 1-1 (RQ1-1): How can novice learners maximize learning effects in programming learning?
 - Goal 1-1 (G1-1): To support selecting an appropriate learning environment for programming learning by novice learners.
 - ✧ Solution 1-1 (S1-1): To clarify the learning effects by grasping the characteristics of the programming learning environment.

On the way to answer RQ 1-1 is to achieve G1-1. As a solution, I investigated mainly S1-1. I created a classification to evaluate multiple environments and identify the characteristics of the programming learning environment. However, classification alone does not elucidate the learning effects. Therefore, I examined the learning effects of the text method and the visual method in the same programming learning environment. Although many text and visual comparative studies exist [6][35][36], investigations that include characteristics such as linguistic representation are scant. Because these characteristics differences should impact the learning effects, I investigated their differences as well as the learning effects of multiple environments.

CHAPTER 1

Specifically, I examined the effect of multiple elements in a programming learning environment on the learning effects. It should be noted that the development of a programming learning environment based on characteristics is future work.

I employed the results of several studies to examine multiple elements because this technique provides stronger evidence of the impact of elements on the learning effects. The subsequent sections will explain my research in more detail.

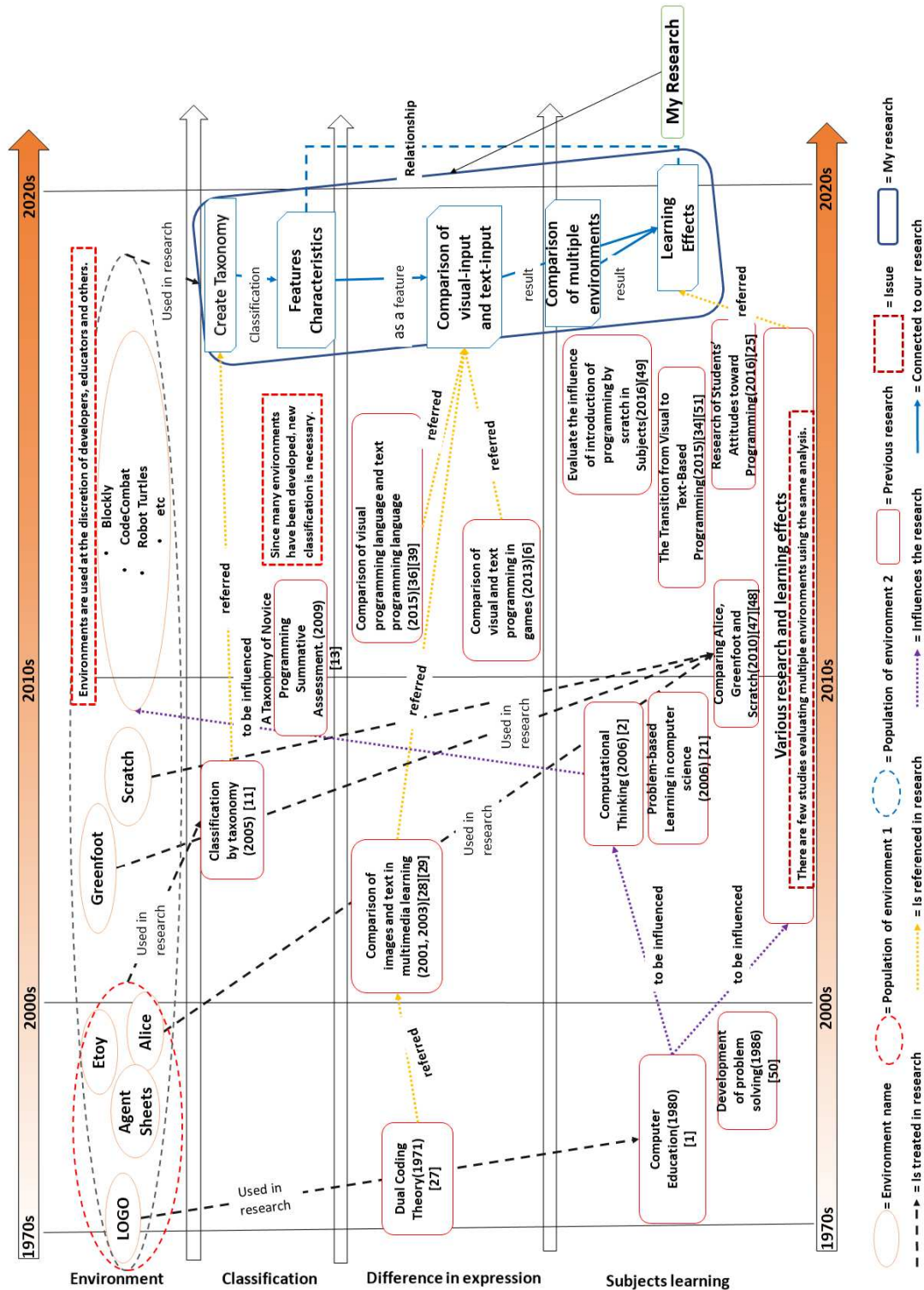


Figure 1-1. Outline of this research

1.1.Programming Learning

Students learn to programme for a variety of reasons. For example, programming learning is used to teach Computational Thinking. The phrase “Computational Thinking” was first used by Seymour Papert in 1980 [1], when he was working on computer education for children using LOGO. In 2008, J. Wing added "abstraction" and "problem-solving" [2]. The effects of programming learning have been extensively studied. Several studies have shown that the learning environment called Scratch [3] improves learners' interest and passion for programming [4][5]. Other studies have shown that using a game called Minecraft tends to improve programming skills [6][7][61]. The results of these studies suggest that the learning effect depends on the method of programming instruction and the learning environment.

In this thesis, I consider the programming learning environment for novice learners and investigate learning effects based on various characteristics.

1.2.Programming Learning Environments

Numerous programming learning environments are utilized for novice learners [3][9][59][69][70]. As examples, Scratch [3][8], Alice [31] and Greenfoot[69] are used in a visual programming language, while CodeCombat [9] and Minecraft Education Edition [10] exist in game software. Previously, Kellaher et al. classified multiple programming environments [11], demonstrating that these environments have unique characteristics. However, the issue is that these environments are used at the discretion of educators and learners. Moreover, it is unclear what kinds of learning effect are derived from the characteristics of each learning environment. Investigating the characteristics of learning environments should reveal the learning effect. This is useful as information to maximize the learning effect of the novice learners.

1.2.1. Method to identify Programming Learning Environments

As the founding premise of this research, I investigated various kinds of programming learning environments. To develop a method for surveying program learning environments described in the literature, I referred to the study by Kai Petersen et al. [12], which is often used for comprehensive literature investigations. First, I employed a Google Custom Search API to search the Web for eight sets of keywords (four sets each from Japanese and English) (Table 1-1). In the table, keywords in the same row have the same meaning in Japanese and English. The top 100 search results for each set of keywords were used, yielding a total of 800 results. I then extracted the programming learning environments by morphological analysis and visual observations, yielding 76 environments for programming learning (Table 1-2).

Table 1-1. Google Keyword Search

Japanese	English
プログラミング 学習 子ども ゲーム	Programming learning game kids
プログラミング 学習 子ども ツール	Programming learning tool kids
プログラミング 教育 子ども ゲーム	Programming education game kids
プログラミング 教育 子ども ツール	Programming education tool kids

Table 1-2. Programming Learning Environments List

No.	Software Name	No.	Software Name	No.	Software Name	No.	Software Name
1	Alice	21	LOGO	41	Programin	61	LEGO MindStorms
2	Ardublock	22	Daisy the Dinosaur	42	RoboMind	62	Romo
3	Blockly	23	Empire of Code	43	Run Marco!	63	Root
4	MOONBlock	24	Erase All Kittens	44	Swift Playgrounds	64	Sphero SPRK
5	Pyonkee	25	Flappy	45	Tech Rocket	65	Vortex
6	Scrach	26	Greenfoot	46	The Foos	66	Wonder Workshop
7	ScratchJr	27	HackforPlay	47	Tickle	67	Arduino
8	SmalRuby	28	Hopscotch	48	Turtle Academy	68	Ichigojam
9	Viscuit	29	JointApps	49	MaKey	69	Java
10	Osmo Coding	30	Junior Coder	50	Osmo Coding	70	JavaScript
11	AgentSheets	31	Kodu Game Lab	51	PETS	71	Python
12	BetaTheRobot	32	Learn Python	52	Puzzlets	72	Ruby
13	BotLogic.us	33	LearnToMod	53	Bitsbox	73	Swift
14	Box Island	34	Lightbot	54	c-jump	74	Tynker
15	Code Monster	35	Minecraft	55	Hello Ruby	75	PROCK
16	Code Studio	36	MinecraftEdu	56	Robot Turtles	76	Algologic
17	Code-Girl Collection	37	Minecraft Education Edition	57	Kano		
18	CodeCombat	38	Move the Turtle	58	Bee-Bot		
19	CodeMonkey	39	Squeak	59	Codie		
20	Crunchzilla	40	Penjee	60	Hackaball		

1.3.Contributions

The contributions of this paper are as follows:

- I provide a taxonomy to qualitatively categorize the programming learning environment.
- I show the difference in learning effects based on the input method of the programming learning environment.
- I show the learning effects as the difference of multiple programming learning environments.
- I show the learning effects derived from the individual characteristics of the programming learning environment.

These contributions will help the novice learners because they assist in selecting the proper programming learning environment.

1.4.Organization of This Thesis

Chapter 1 highlights my research goal. There are over 76 kinds of programming learning environments, leading to issues with programming learning environments.

The rest of this thesis is organized as follows. Chapter 2 explains the taxonomy to evaluate multiple programming environments and shows the classification results based on the proposed taxonomy. Chapter 3 highlights the learning effects by different programming methods (text input and visual input). Chapter 4 investigates the learning effects based on the characteristics in multiple environments. In addition, the correlation between characteristics is shown. Finally, Chapter 5 summarizes this thesis and describes future works.

CHAPTER 2

CLASSIFICATION OF PROGRAMMING LEARNING ENVIRONMENTS

First, I created a taxonomy by defining items for classification of programming learning environments. I then used this taxonomy to classify several environments. Based on the results, I evaluated the characteristics of each environment, including the attributes of the visual language environment and game software. This survey addressed the following research question:

- Research Question 2-1 (RQ2-1): Can a taxonomy group, evaluate, and compare programming learning environments effectively?

The contributions of this research are:

- Development of a taxonomy table for comparison and evaluation of environments based on a standard protocol.
- The taxonomy table aids users in selecting environments with appropriate attributes for the learning objective.

2.1. Background

Caitlin Kelleher et al. [11] investigated dozens of programming environments by classifying them into categories. Furthermore, Shuhaida Sheridan et al. [13] classified learning assessment for novice programming, and then evaluated programming environments using the same taxonomy. Unlike the work of Kelleher et al [11], which included numerous programming environments, this study focuses on programming learning environments for children, with the goal of creating a taxonomy table that is optimized to help users [educators and learners (children)] select the environments referred to in [11]. Additionally, I use our taxonomy table to evaluate the programming learning environments intended for programming education. Because the number of available environments has drastically increased, the environments targeted in this

chapter are visual languages, game software, and other software that work on PCs (including tablets and other devices).

2.2. Creation of Taxonomy

I created a taxonomy table to evaluate program learning environments qualitatively (Table 2-1) by referencing Kelleher et al. [11]. Specifically, I optimized Kelleher's table for learning environments and added the following categories: Game Elements and Requirements. Game elements are added because playing a game is a suitable learning method for programming, especially programming concepts. The number of the games to learn programming has increased. Examples include CodeCombat [9] and Lightbot [14]. This survey considered game elements that deal with games. I used Rule/Restriction, Goal, and Reward (the common parts of the definition by Katie Seaborn et al. [15] and Juho Hamari et al. [16]) to define game elements. From the viewpoint of multi-play, I also added Cooperation [17].

Table 2-1. Taxonomy Table (Optimized Kelleher's taxonomy [11])

Style of programming (C1)	Programming constructs (C2)	Representation of code (C3)
Procedural (i11)	Conditional (i21)	text (31)
Functional (i12)	Loop (i22)	pictures (i32)
Object-based (i13)	Variables (i23)	flow chart (i33)
Object-oriented (i14)	Parameters (i24)	animation (i34)
Event-based (i15)	Procedures/methods (i25)	forms (i35)
Statemachine-based (i16)	User-defined data types (i26)	finite state machine (i36)
	Pre and post conditions (i27)	physical objects (i37)
	Recursion (i28)	
Construction of programs (C4)	Support to understand programs (C5)	Designing Accessible Languages (C6)
typing code (i41)	back stories (i51)	limit the domain (i61)
assembling graphical objects (i42)	debugging (i52)	select user-centered keywords (i62)
demonstrating actions (i43)	physical interpretation (i53)	remove unnecessary punctuation (i63)
selecting/form filling (i44)	liveness (i54)	use natural language (i64)
assembling physical objects (i45)	generated examples (i55)	remove redundancy (i65)
Game elements (C7)	Supporting Language (C8)	Operating Environment (C9)
Rule/Restriction (i71)	Japanese (i81)	Windows (i91)
Goal (i72)	English (i82)	Mac (i92)
Rewards (i73)	Others (i83)	Linux (i93)
Cooperation (i74)		Web (i94)
		iOS (i95)
		Android (i96)
		Others (i97)
Interface (C10)	Experience (C11)	
PC (i101)	unnecessary (i111)	
Tablet(8inch~) (i102)	necessary (i112)	
Smartphone (i103)		
Other Interface (i104)		

2.2.1. Taxonomy details

The taxonomy table divides the 56 items into 11 categories. Each category is explained below.

Style of Programming (C1) indicates the programming style built into the environment. There are six styles: procedural, functional, object-based, object-oriented, event-based, and state machine-based.

Programming Construct (C2) reflects the programming construct that can be learned in an environment. Constructs include conditionals, loops, variables, parameters, procedures/methods, user-defined data types, pre-and-post conditions, and recursions. In this survey, all types of loops were lumped together because they are conceptually identical from the standpoint of teaching. I also included recursion, because some environments teach this concept.

Representation of Code (C3) explains how programs are displayed. Representations include text, pictures, flowcharts, animations, forms, finite state machines, and physical objects.

Construction of Programs (C4) describes how to programs are input. Items include typing code, assembling graphical objects, demonstrating actions, selecting/form-filling, and assembling physical objects.

Support of Program Understanding (C5) focuses on how the environment helps the user comprehend a program. Examples include back stories, debugging, physical interpretations, liveliness, and generating examples.

Designing Accessible Language (C6) represents the functions that make programming languages easier to learn. Functions include limiting the domain, selecting user-centered keywords, removing unnecessary punctuation, using natural language, and removing redundancy.

Game Elements (C7) is a new category representing the game element included in an environment, such as rewards and goals. The presence or absence of such elements influences the learning effect.

Supporting Language (C8) is the language used in each environment. This has been added because the users' understanding of the description of the environments is relevant to the learning effect. Supporting languages are classified as English, Japanese, and others.

Operating Environment (C9) is the platform in which each environment works. I added this category because the way that an environment is launched and used is an important aspect of usability. Operating environments were classified as Windows, Mac, Linux, Android, iOS, Web, and others.

Interface (C10) denotes the device suitable for the environment. This was added for the same reason as Operating Environment. Interfaces are classified as PC, Tablet, Smartphone, and Other.

Experience (C11) indicates whether the environment targets novice programmers. This was added because this research aimed to survey program learning environments for children without programming experience.

2.3.Selection of Environments to Classify

I identified 76 environments based on the method described Chapter 1. This Chapter targets software working on a device such as a PC or a tablet, reducing the number of environments to 43 (Table 2-2). The environments are divided into three attributes (At): visual programming environments (Vi), game software (GM), and other educational software (Ot). Then the environments are classified according to the text from the official website. Although the websites are classified into these three attributes, their definitions are ambiguous. The attributes are characterized using a taxonomy, which should be useful to group future characteristic sets.

CLASSIFICATION OF PROGRAMMING LEARNING ENVIRONMENTS

Table 2-2. Environments survey list

ID	Name	At	ID	Name	At	ID	Name	At
T1	Alice	Vi	T21	Code-Girl Collection	GM	T41	Squeak	Ot
T2	Ardublock	Vi	T22	CodeMonkey	GM	T42	Swift Playgrounds	Ot
T3	Blockly	Vi	T23	Crunchzilla	GM	T43	Tynker	Ot
T4	MOONBlock	Vi	T24	Daisy the Dinosaur	GM			
T5	Pyonkee	Vi	T25	Empire of Code	GM			
T6	Scrach	Vi	T26	Erase All Kittens	GM			
T7	ScratchJr	Vi	T27	Flappy	GM			
T8	SmalRuby	Vi	T28	HackforPlay	GM			
T9	Viscuit	Vi	T29	Junior Coder	GM			
T10	Greenfoot	Vi	T30	Lightbot	GM			
T11	Hopscotch	Vi	T31	Move the Turtle	GM			
T12	Kodu	Vi	T32	Penjee	GM			
T13	LearnToMod	Vi	T33	RoboMind	GM			
T14	Programin	Vi	T34	Run Marco!	GM			
T15	BetaTheRobot	GM	T35	Tech Rocket	GM			
T16	Bo1 Island	GM	T36	The Foos	GM			
T17	BotLogic.us	GM	T37	Tickle	GM			
T18	Code Monster	GM	T38	Turtle Academy	GM			
T19	Code Studio	GM	T39	JointApps	Ot			
T20	CodeCombat	GM	T40	Learn Python	Ot			

2.4.Results and Analysis of Classification by Taxonomy

2.4.1. Analytical method

In this section, I surveyed the features of programming learning environments. As a classification method, two people separately evaluated each environment according to the following process:

- (1) Read the words on the official website of each environment.
- (2) Use each environment.
- (3) Verify the classification in the taxonomy table.
- (4) Cross-check the classification results of the evaluators.

2.4.2. Overall results

Table 2-3 is a taxonomy table, which shows the classifications and attributes of the environments. Furthermore, Figure 2-1 shows the corresponding number of environments for each classification. Several environments have multiple attributes. Additionally, some classifications may be applicable to other attributes (e.g., “robot” or “unplugged tool”). Therefore, additional research is necessary.

For Style of Programming (C1), Procedural, the most basic concept, has the most entries (25 environments). Visual programming environments have been applied to the Object-oriented style of programming. Because Procedural and Object-oriented are basic styles of programming, many environments have been developed for these aspects.

For Programming Constructs (C2), five entries are supported by more than half of the environments: conditionals, loops, variables, parameters, and procedures/methods. All of these are important concepts for programming. A total of 28 environments incorporate conditions and loops as basic programming concepts, indicating that many environments teach the logic of programming.

CLASSIFICATION OF PROGRAMMING LEARNING ENVIRONMENTS

For Representation of Code (C3), 90% of the environments use text. Such environments refer to general languages, allowing users to learn programming in a style that closely resembles regular programming, or to understand programs in a natural language. Additionally, some environments, such as Lightbot, use pictures to represent programs. These environments are more easily intuitively understood than text-based ones.

In Construction of Programs (C4), assembling graphical objects, a way to visualize language, has the most entries. Although some environments require users to type code, many others enable users to input code by dragging and dropping. This is advantageous because these environments were developed for children who may not be proficient at typing or using a keyboard.

In Support to Understand Programs (C5), physical interpretation has the most entries. In this classification, the code is expressed by a specific action such as “walk” or “jump”; again, this is appropriate because these environments were developed for children.

In Designing Accessible Language (C6), the attribute ‘limit the domain’ has the most entries (31 environments). Limiting the domain makes it easier for learners to understand programming.

In Game Elements (C7), many environments include Rules/Restrictions and Goals. At least one game element is present in 24 of the 43 software environments. Therefore, most environments are categorized as game software, enabling users to learn programming by playing a game. The advantage of game software is that users can understand programs by watching an action rather than reading written instructions.

In Supporting Language (C8), English was the most supported (36 environments), largely because many programming learning environments were developed in Europe and North America. Some environments support multiple languages, enabling learners to learn in their own languages, leading to a better understanding of programming.

In Operating Environment (C9), Web has the most entries. Because environments that work on the Web do not require extensive preparation, beginners can more quickly begin to learn to program. Additionally, some applications corresponding to tablets and smartphones are supported by some environments. These environments make it easier to learn programming.

CHAPTER 2

In Interface (C10), PC has the most entries (33 environments), indicating that most environments aim to teach users a general language.

In Experience (C11), over 90% (40 environments) of the environments can be used by beginners (especially children).

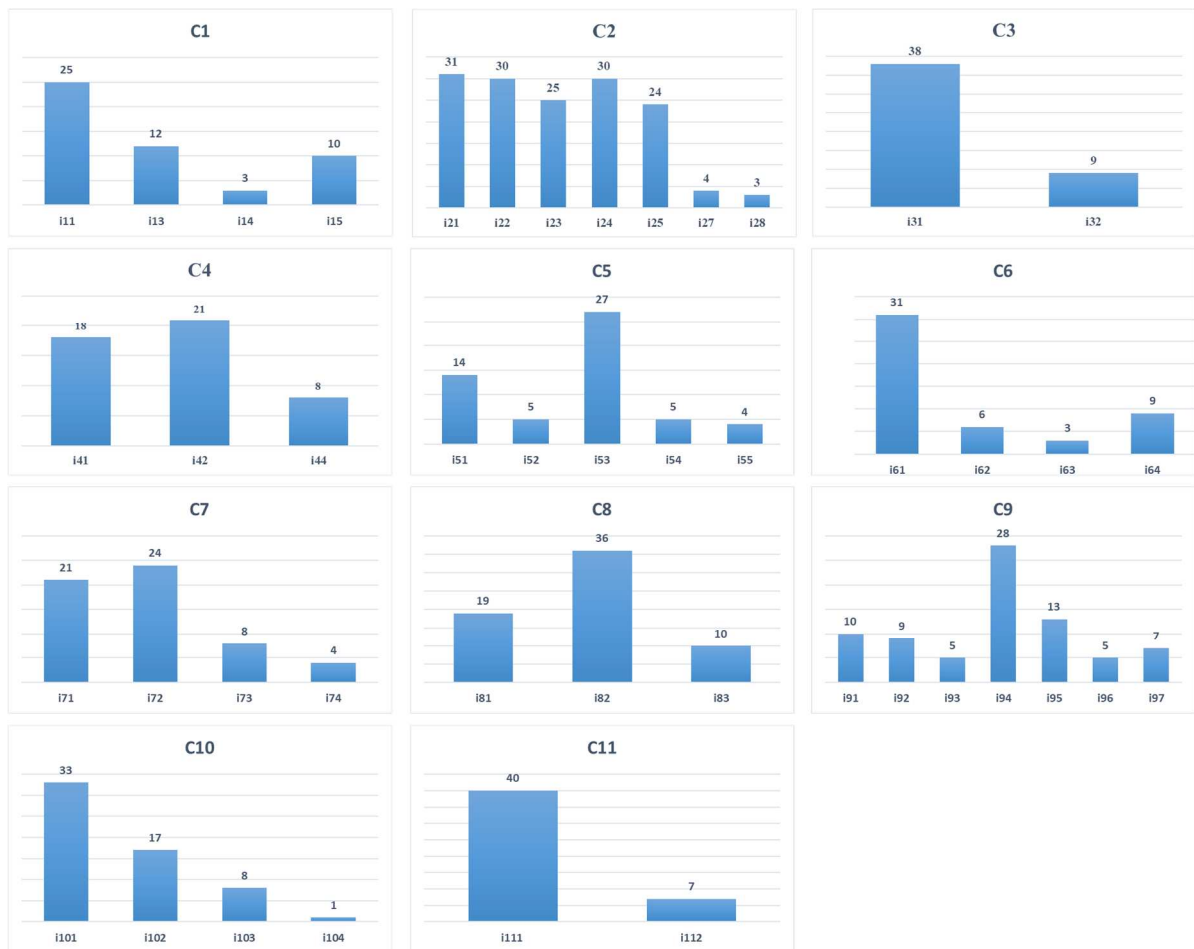


Figure 2-1. Result of Classification

2.4.3. Results of each attribute

2.4.3.1. Visual programming environments

Among the environments I examined, there are 14 visual environments. Many visual programming environments are object-based and include basic concepts such as conditionals, loops, and procedures/methods. Learning to program is easier in a form that is closer to real programming. Accordingly, text is used as a representation of the code. Additionally, as a method of programming, many environments involve assembly of graphical objects, making it possible to program by dragging and dropping. Not all visual environments possess game elements. In other words, these environments are not games, but are instead specialized for creating programs.

2.4.3.2. Game software

In addition, there are 24 game software environments, many of which use a Procedural style of programming or have Rules/Restrictions and Goals among the game elements. These elements clarify the learning goals. Therefore, game software is highly suitable for introductory learning. In addition, I performed a cross-tabulation with Game Elements, a newly added category, and Programming Constructs, the basic goal of programming learning. Table 2-4 shows the results. Many of the environments that have Rules/Restrictions and Goals include conditionals and loops. The reason for this is that showing the action of a conditional in a game helps users to comprehend such concepts. Many games with these game elements, such as CodeCombat [9] and Lightbot [14], are similar to Turtle Graphics. If users (educators and learners) want to learn conditionals and loops, which are important aspects of programming logic, they should select a game.

Table 2-4. Crosstabulation with game elements

<i>Game Elements</i>	Conditional	Loop	Variables	Parameters	Procedures /methods	Pre and post conditions	Recursion
<i>Rule/Restriction</i>	12	11	8	10	9	2	2
<i>Goal</i>	15	13	11	13	12	2	2
<i>Reward</i>	6	6	5	3	5	1	1
<i>Cooperation with Others</i>	2	2	3	3	3	1	0

2.4.3.3. Other Software

Five of the environments are classified as neither game software nor visual language. Many web services gather programming learning applications, and there is an environment for easily developing applications. Five programming expression environments use textual representations. For other items, there are individual characteristics for each environment. Furthermore, it is possible to break down the field of each environment.

2.4.3.4. Summary of Attributes

Each attribute has common characteristics. Table 2-5 shows the characteristics most applicable to each attribute, demonstrating that ambiguous definitions can be determined with this taxonomy. However, this taxonomy is not applicable to one environment.

Table 2-5. Attribute Table

Attribute	Common Characteristics
Vi	Assembling graphical objects or Selecting/form filling Object-based programming
GM	Typing Code or Assembling graphical objects Rule/Restriction, Goal, Rewards
Ot	A collection of various tools without common characteristics

2.5. Discussion

I investigated the following research question:

- RQ2-1: Can a taxonomy group, evaluate, and compare programming learning environments effectively?

I derived a suitable taxonomy table based on Kelleher [11] to compare and evaluate programming learning environments, as demonstrated by the fact that my taxonomy can classify all 43 environments. For example, many environments represent code by text and demand that the code is inputted by assembling graphical objects. Environments with game elements are suitable to improve motivation and teach programming concepts [66][67].

Environments often have common characteristics (Table 2-5). Hence, it is possible to classify environments by attributes. Herein a classification using three attributes (visual, game software and other) is demonstrated. Therefore, it is possible to characterize the attributes of a learning environment by my taxonomy.

Not only is it feasible to evaluate environments using a unified taxonomy, it is also possible to select environments based on learning objectives.

2.6.Related Works

In 2005, Caitlin Kelleher and Randy Pausch surveyed programming learning environments, classified them using their original taxonomy, and created a table to explain environmental attributes [11]. Their survey and taxonomy were highly detailed, and greatly contributed to resolving issues in this field. Due to advances in programming learning environments, a new survey is necessary to improve the taxonomy and incorporate new technology. In addition, the preceding survey targeted all kinds of programming education environments, which would be extremely difficult today due to the greater diversity of environments. Accordingly, our survey specialized in environments categorized as software developed for the purpose of education. This approach provides a taxonomy table suitable evaluation of environments targeting beginners.

2.7.Limitations

One limitation of this study is that the results of evaluation may depend on the evaluator. Although two researchers cross-checked the findings in this survey, repetition and reproduction of the findings with more evaluators will necessary in order to confirm the conclusions.

In addition, the keywords used to extract the environments (Table 1-1) did not cover all environments for beginners. In this search, I targeted "children". However, not all applicable environments may be labeled as "for children". Thus, from the viewpoint of the retrieval method, acquisition of high-quality data is an important goal for future research.

Additionally, because I used the results of a Google search, it is possible that older environments were excluded. Such environments may have greater influence than newer environments. Accordingly, it is important to also classify older environments.

2.8.Conclusion

I surveyed 43 environments with an emphasis on a visual language and software that work alone on PCs or similar devices to create a taxonomy table for programming learning environments. The proposed table can evaluate and compare such environments. Furthermore, my taxonomy can characterize the definition of visual language and game software from their characteristics. The "Other" attribute needs to be divided further. The experiment confirms that the classification and evaluation results are independent of the evaluator. Consequently, this taxonomy table helps users (learners and educators) select the appropriate environment based on their objective.

In the future, more than two people must verify the taxonomy table to verify its reliability. Additionally, I will continue to investigate whether this taxonomy table helps users select the most appropriate environment in actual situations.

CHAPTER 3

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS IN PROGRAMMING METHOD

This chapter investigates learning effects as a function of the characteristics of the same environment. In particular, this chapter focuses on Representation of Code and Construction of Programs because the results in Chapter 2 indicated that many environments use common characteristics. The code is represented by typing or assembling graphical objects, whereas programs are constructed using text or pictures. Herein the differences between visual and text input methods (Representation of Code and Construction of Programs) are investigated in the same Lua programming environment to determine if the input method influences the learning effects. Specifically, a combination of text (Representation of Code) and typing code (Construction of Programs) are compared with a combination of image (Representation of Code) and drag-and-drop (Construction of Programs).

This research examines the following Research Questions (RQs):

- Research Question 3-1 (RQ3-1): Does a visual-based input method induce a different attitude toward programming than a text-based input method?
- Research Question 3-2 (RQ3-2): Does the understanding of programming differ between visual-based and text-based input methods?

RQ3-1 assesses whether a given programming method is suitable for an introductory environment. This RQ can elucidate the attitude of novice learners toward programming, based on the input method. The results should assist in selecting the most suitable method for introductory programming. RQ3-2 evaluates the understanding of programming basics. Furthermore, it examines the understanding of programming concepts by focusing on sequential execution, conditional branching, and repetition. This RQ can reveal which method is most suitable for learning. Because increasing learning efficiency should enhance the

learning effect, these RQs can elucidate the appropriate programming method and environment for introductory education. In addition, the proper learning environment should improve novice learners' motivation to learn.

3.1. Background

3.1.1. Programming learning for novice learners

It is often noted that beginners have difficulty learning to program [11] [18]. Several studies have been conducted to address this issue. Some used a visual method like Scratch, developed at MIT [3][4][8][19], whereas another study used a text method (the C language) [20]. Other studies used both visual and text methods for Project-based Learning for programming based on problem-solving [21][50], as well as Game-based Learning [22][23][24][68]. In addition, some studies investigated attitudes toward programming [25].

Each method has its own learning effect. Some success with novice learners has been reported using these methods. However, it remains unknown which programming input method (visual or text representation of code) is more suitable for novice learners, and the learning effect for each method is also unclear. Based on this situation, this chapter focuses on the input method, and compares the learning effects of both kinds of input methods within the same programming environment. This approach is intended to serve as a reference for educators when selecting an input method for teaching novice learners.

3.1.2. Two input methods

In this section, I compare the learning effects of text and visual inputs in the same programming language. Comparisons of the learning effects of text and visual methods can be traced back to the Dual-coding theory (DCT) proposed by Paivio [26] [27]. In this theory, human information processing can be divided into two systems: language and non-language. Language systems use character information such as characters and voices, whereas non-language systems use sensory information such as images. These features affect recognition by humans [27].

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS IN PROGRAMMING METHOD

Several studies have examined characters and images using DCT. One study investigated the influence of the student's prior knowledge on learning in a computer-based physical lesson as a function of differences in the presentation format (text, images, or animation) [28]. The results revealed that when teaching beginners, images are useful for descriptive and procedural learning. Another study concluded that it is more effective to use images and characters together [29]. Furthermore, Eitel et al. reviewed 42 studies on the presentation order of text and images during learning [30]. The boundary condition to determine whether it is better to use the first process as an image or text is stated as the relative complexity of the image and the text. Unlike this study, which focuses on programming languages, these studies focused on multimedia learning.

A programming language can be expressed as text or visual representations. For example, visual programming languages such as Scratch [3] or Alice [31] use drag-and-drop of visual inputs. A visual language is suitable for initial exposure of novice learners who are unfamiliar with programming languages. Furthermore, text programming languages such as Python and JavaScript receive typed input via the keyboard. Text languages can be more sophisticated than visual languages; however, while a text language is better suited if the purpose is clear, learners must possess sufficient typing skills. In addition, some researchers have investigated the transition to text-based programming from visual-based programming [32][33][34]. Hence, the research results implemented in the field of multimedia are applicable to novice learners of programming.

In a study comparing programming methods, visual methods were noted to be an easy for educators [35]. Studies on programming in higher education have shown that visual-based languages produce better results than alternative approaches [36]. One study developed an extended function of CodeBlock, which expands the visual programming function to Minecraft, and found that this environment resulted in improved recognition of programming. Although this study compared visual programming functions to text environments, no significant difference was detected [6].

Several studies regarding multimedia and programming learning have reported that the visual method is suitable for novice learners. In other words, they suggest that using a visual input method may be more advantageous for novice

learners. However, programming involves both visual information and behavioral aspects, such as input of programs and confirmation of execution results. It is difficult to support all results in the multimedia field. In addition, there is no significant difference in recognition of programming in comparison with the text environment [4]. Consequently, the proper input method for novice learners has not been definitively established. To provide clear answers, this study uses visual inputs and text inputs at the same level of abstraction, built in the same environment. Hence, the comparison is based only on differences in input, with no effect on the environment. In addition, this study strives to include younger participants.

3.1.3. Minecraft and ComputerCraftEdu with Programming Learning

For programming learning, I used Minecraft, an internationally popular sandbox game that involves using various materials to build objects and structures. Minecraft has been used as an educational environment in mathematics and science [37][38].

ComputerCraft is a Minecraft modification (mod) that adds the functions of the Lua programming language. Previous research used a workshop approach to study programming language education using ComputerCraft, based on the revised taxonomy of Bloom [7]. The results revealed that student motivation improved when using ComputerCraft. Consistent with this, another study reported that ComputerCraft is a beneficial tool for programming language education [6].

I used ComputerCraftEdu (CCEdu), the education version of ComputerCraft. The CCEdu has two environments for programming: text-based and visual-based (Figure 3-1 and Figure 3-2). Text-based programming can be controlled in Minecraft using the same method as general text programming, whereas visual-based programming employs illustration blocks. Both environments have the same level of abstraction. For example, the instruction ‘turtle.forward()’ moves the turtle forward. Figure 3-3 shows the relationship between the two methods.

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS IN PROGRAMMING METHOD

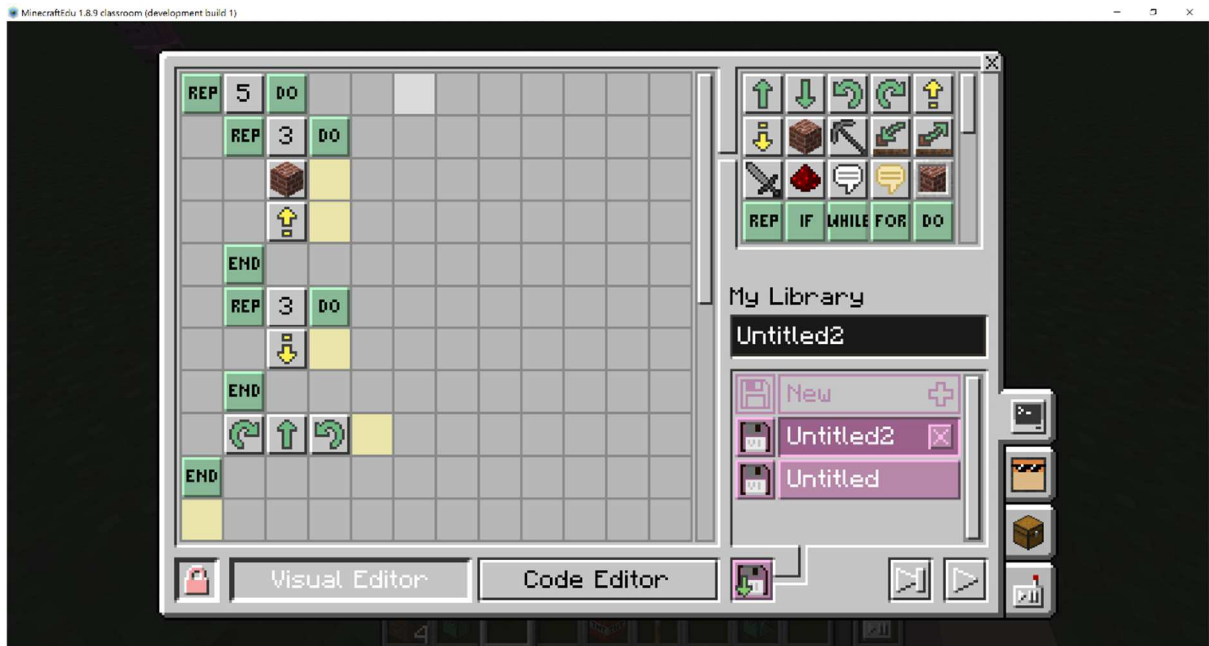


Figure 3-1. Two input method (Visual)

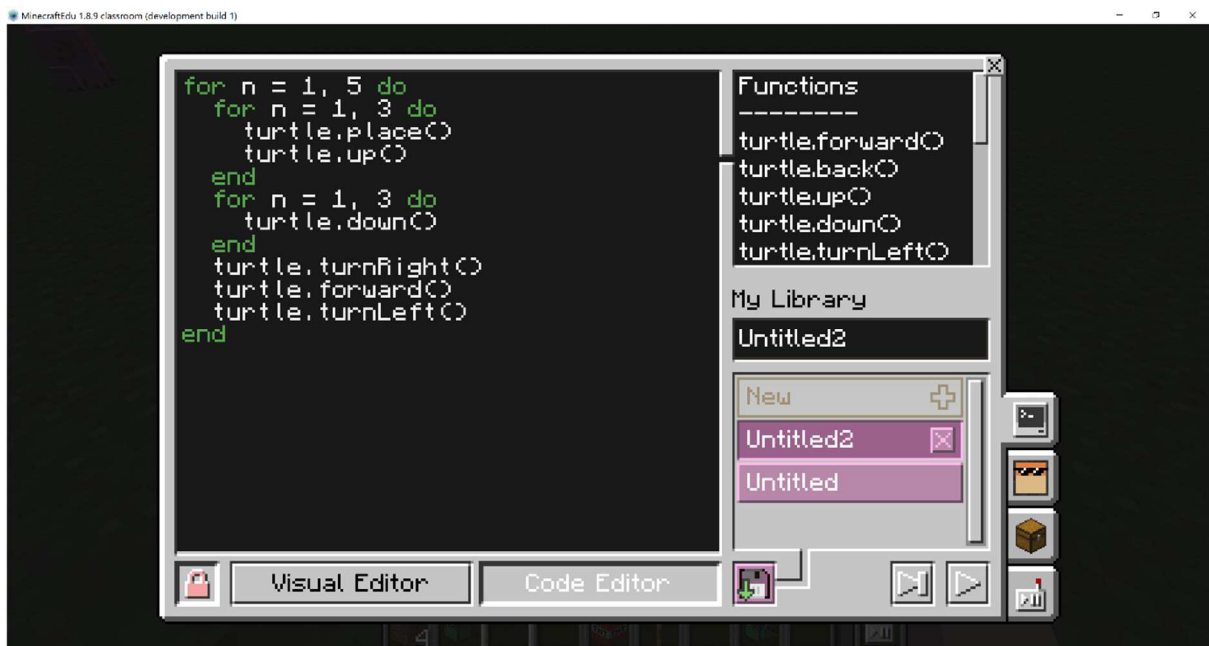


Figure 3-2. Two input method (Text)







instruction	Visual Input	Text Input
Move up Move forward	 	<code>turtle.up()</code> <code>turtle.foward()</code>
Place block Inspect block	 	<code>turtle.place()</code> <code>turtle.inspect()</code>
for sentence		for X = 1 to 5 do end
if sentence		if <code>turtle.detecte()</code> then end

Figure 3-3. Comparison of input methods

3.2.Workshop Design

I implemented two types of workshops (visual-based and text-based), designed for elementary and junior high school students. Each workshop was configured as a short course, and both covered the same contents. Specifically, each workshop consisted of a tutorial, sequential execution, repeat, conditional branching, and a free problem. The order of the workshop contents was as follows:

- (1) Tutorial content focused on operations in Minecraft and ComputerCraftEdu.
- (2) Sequential execution, programming fundamental. The example in the workshop was to move a turtle and place a block in Minecraft. The user learns the turtle instructions for moving forward, back, left, right, up, or down.
- (3) Repetition: loops using the “for” statement to place blocks (Stack and Load Line) using the turtle. Examples included stacking five blocks and creating a staircase pattern.
- (4) Conditionals using the “if” statement to avoid a block. The workshop used two examples: “avoid obstacles” and “remove TNT.”

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS IN PROGRAMMING METHOD

(5) Finally, a free problem was used to assess the students' programming skills. In this problem, the user needed to create one alphabetic character. In addition, to gauge the user's understanding of programming, the workshop included six problems (Table 3-1).

The total time for the workshop was approximately 3.5 hours, allocated as follows: Tutorial (30 minutes), Sequential (50 minutes), Repetition (25 minutes), Conditional (25 minutes), Free problems (30 minutes), and a Break (30 minutes). Although the course was short, it taught the programming concepts of Conditional, Loop, and Sequential were taught.

Table 3-1. Problem Contents

#	Problem Contents	Survey Category
P1	Move the turtle three steps, rotate left, and move two more steps.	Sequential
P2	Add four blocks.	Sequential
P3	Stack eight blocks.	Loop
P4	Create a stairway with eight steps.	Loop
P5	If a TNT block is in front of the turtle, avoid it.	Conditional
P6	If a diamond block is in front of the turtle, mine it.	Conditional

3.3.Experiments

Using comparative experiments based on the "Workshop Design" described in the section 3.2, I investigated whether the text or visual method is more suitable for introductory education. In addition, I developed two hypotheses that correspond to the RQs:

- Hypotheses 3-1(H3-1): Visual input programming lecture induces a larger change in attitude toward programming.
- Hypotheses 3-2(H3-2): Programming is easier to understand using the visual input method.

H3-1, which corresponds to RQ3-1, speculates that the change in attitude toward programming is more significant for the visual input group because the visual input method is more intuitive than the text input method. H3-2 corresponds to RQ3-2. Similar to the rationale for H3-1, I hypothesized that it should be easier to comprehend programming using visual inputs.

3.3.1. Participants

I recruited participants via a website. Participants were primary and junior high school students in Japan ranging in age from 6 to about 15 years old. The application allowed participants to select the course type (visual or text). In each year, 36 students responded to the recruitment targeting novice learners; thus, a total of 72 subjects participated. Based on the participant's preference, they were divided into the Visual Group (VG) and the Text Group (TG). Learners attended the workshop corresponding to their group. VG had 46 participants, whereas TG had 26.

3.3.2. Questionnaire

The same questionnaire was administered twice to assess the change in attitude toward programming: Before Questionnaire (BQ: Q1B–Q10B) and After Questionnaire (AQ: Q1A–Q10A) (Table 3-2). Based on Zorn et al [6], I used five factors to assess attitude: Interest, Difficulty, Usefulness, Fun, and Willingness. Willingness is included because the desire to learn is an important element. Each question was evaluated using the six stages of the Likert scale (1: Strongly disagree, 2: Disagree, 3: Somewhat disagree, 4: Somewhat agree, 5: Agree and 6: Strongly agree). The Likert scale was set to six stages to eliminate an intermediate value, allowing the responses to be divided into "can" and "cannot". For all questions except Q2 and Q7, a higher score in the AQ indicated an improvement. For Q2 and Q7, a lower score in the AQ indicated an improvement. Furthermore, I created two questions (Q11, Q12) to assess the participants' understanding of programming.

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS
IN PROGRAMMING METHOD

Table 3-2. Questionnaire

#	Attitude Toward Programming Question	Survey Category
Q1	Are you interested in programming?	Interest
Q2	Do you think that learning to program is difficult?	Difficulty
Q3	Do you think that knowing how to program is useful?	Usefulness
Q4	Do you think programming is fun?	Fun
Q5	Do you want to learn to program?	Willingness
Q6	Are you interested in the turtle program?	Interest(Turtle)
Q7	Do you think that the learning the turtle program is difficult?	Difficulty(Turtle)
Q8	Do you think that knowing how to turtle program is useful?	Usefulness(Turtle)
Q9	Do you think turtle programming is fun?	Fun(Turtle)
Q10	Do you want to learn to turtle program?	Willingness(Turtle)
Understanding Programming questions		
Q11	What is a conditional?	Conditional
Q12	What is a loop?	Loop

3.3.3. Analysis Method

To determine the appropriate analysis method, I tested the normality of the results of each questionnaire using the Shapiro–Wilk test. In this test, which evaluates the normality of a given distribution, a p-value ≤ 0.05 indicates lack of normality. In all populations, I was unable to confirm that the data follows a normal distribution (Table 3-3). Hence, I adopted the Wilcoxon signed-rank test and the Wilcoxon rank–sum test. Unlike t-tests, these tests can be used without an assumption of normality. The Wilcoxon signed-rank test is used to test for significant differences between two groups with correspondence, whereas the rank–sum test is used to test for significant differences between two groups without correspondence.

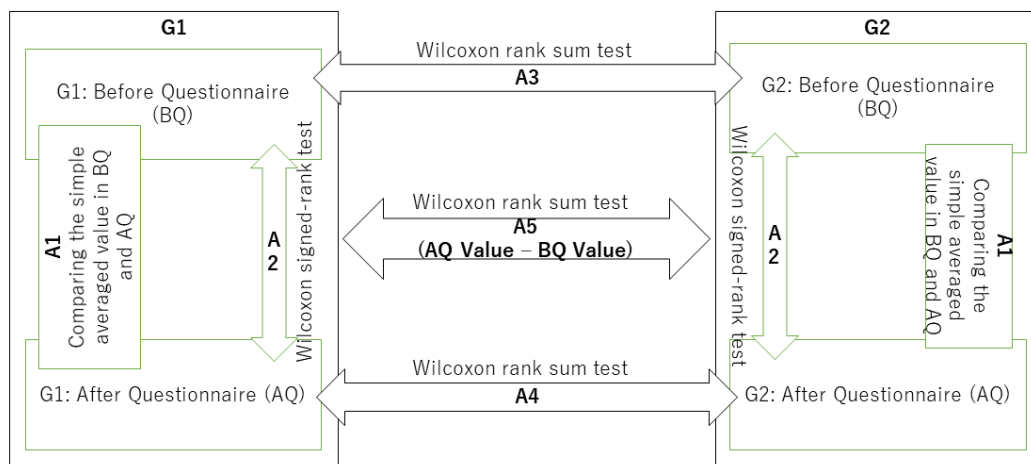
The number of valid responses was 38 (VG) and 26 (TG). To address RQ3-1, I evaluated the following

- (A1) Analyze the change in a simple averaged value
- (A2) Implement a Wilcoxon signed-rank test in BQ and AQ, by group
- (A3) Implement a Wilcoxon rank–sum test for the results of BQ, by group
- (A4) Implement a Wilcoxon rank–sum test for the results of AQ, by group
- (A5) Implement a Wilcoxon rank–sum test for the change from BQ to AQ.

Figure 3-4 shows the details of the analysis.

Table 3-3. Result of the Shapiro-Wilk test

	VG				TG			
	BQ		AQ		BQ		AQ	
	W	p	W	p	W	p	W	p
Q1	0.8411 49	8.11E- 05	0.7024 22	1.80E- 07	0.6994 44	5.21E- 06	0.5938 14	2.51E- 07
Q2	0.9240 76	0.0131 1	0.8829 25	0.0008 7	0.8894 36	0.0091 65	0.8841 48	0.0070 52
Q3	0.6482 26	2.67E- 08	0.6240 26	1.21E- 08	0.6684 89	2.02E- 06	0.5451 2	7.30E- 08
Q4	0.7570 55	1.56E- 06	0.6612 49	4.15E- 08	0.6425 64	9.52E- 07	0.5273 94	4.76E- 08
Q5	0.7762 91	3.56E- 06	0.6643 08	4.61E- 08	0.6346 14	7.60E- 07	0.6907 53	3.97E- 06
Q6	0.8040 84	1.27E- 05	0.6649 48	4.71E- 08	0.7606 26	4.04E- 05	0.6659 6	1.88E- 06
Q7	0.9042 73	0.0033 72	0.9111 77	0.0053 5	0.8610 33	0.0023 46	0.8696 28	0.0035 03
Q8	0.7966 19	8.93E- 06	0.6734 67	6.34E- 08	0.6963 93	4.73E- 06	0.7815 64	8.69E- 05
Q9	0.7698 49	2.69E- 06	0.6907 09	1.17E- 07	0.7082 25	6.88E- 06	0.5792 84	1.72E- 07
Q10	0.8146 39	2.11E- 05	0.6989	1.58E- 07	0.6755 84	2.50E- 06	0.7028 7	5.81E- 06



- (A1) Analyze the change as the simple averaged value (Table3)
- (A2) Implement a Wilcoxon signed-rank test in the BQ and AQ of each group
- (A3) Implement a Wilcoxon rank sum test for the results of BQ in G1 and G2
- (A4) Implement a Wilcoxon rank sum test for the results of AQ in G1 and G2
- (A5) Implement a Wilcoxon rank sum test for the change in BQ and AQ

Figure 3-4. Analysis Method

3.4.Result

3.4.1. Attitude Toward Programming

3.4.1.1. Questionnaire result

Figures 3-5 (VG) and 3-6 (TG) show the results of the questionnaires (Q1–Q10) using violin plots. A violin plot expresses the distribution of data, allowing the distribution density, average value, and median value to be confirmed. Thus, it is possible to verify the change in the value of the Likert scale before and after the workshop, as well as the distribution density. The green lines in the plot (Figures 3-5 and 3-6) show the average values. After the workshop, the results for most categories improved in the VG group. On the other hand, the results in the TG group decreased to an overall negative attitude, except for those related to interest in programming (Q1, Q6) and difficulty of programming (Q2, Q7), which showed improvement. The change in values was larger in VG than TG. Hence, visual inputs may be more suitable for novice learners than text inputs. However, the TG had a larger improvement in the difficulty of programming than VG. In addition, there was no difference between BQ and AQ in VG, because the answers regarding difficulty were largely positive in BQ.

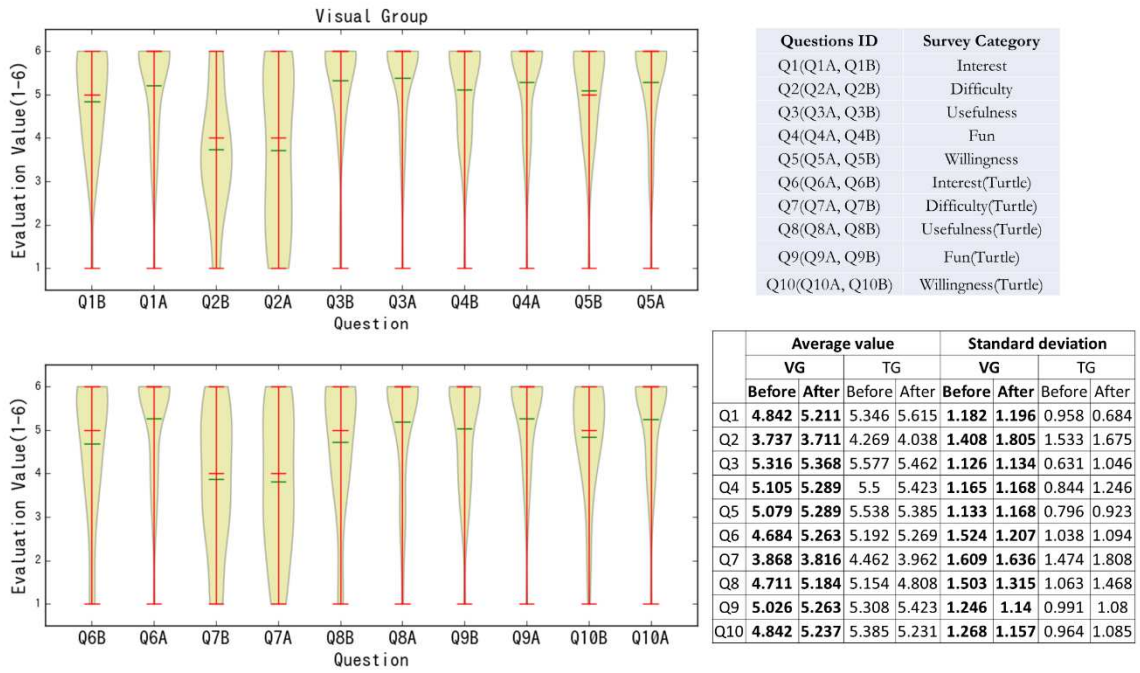


Figure 3-5. Result of VG

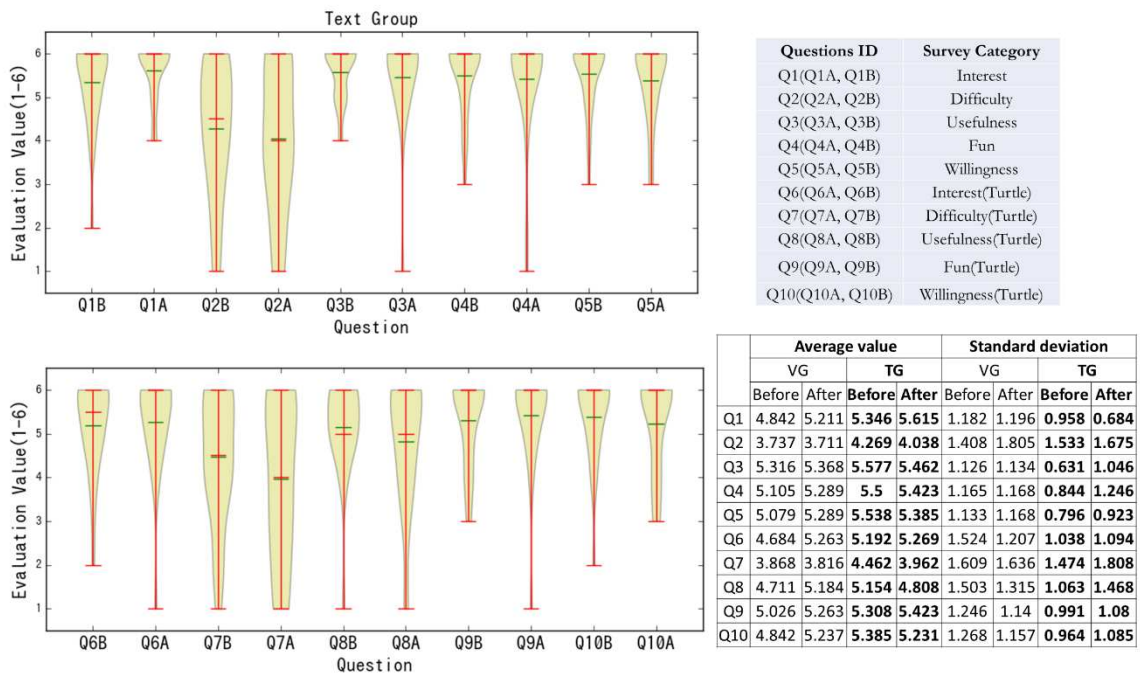


Figure 3-6. Result of TG

3.4.1.2. Analysis of the results

I analyzed the questionnaire results using the Wilcoxon signed-rank test and the Wilcoxon rank-sum test (p -value < 0.05). Table 3-4 shows the results for A1, whereas Table 3-5 shows the results for A2–A3. According to A1, VG improved in all categories. In particular, the attitude towards turtle programming improved, and the interest in turtle programming improved by about 0.6 points. However, some of the learners reported a lower value for attitude after the lecture. It is possible that some learners became bored with programming or were more absorbed in playing the game than programming. In TG, some categories also improved to a positive attitude, whereas others decreased to a negative attitude. The large amount of input necessary to program may inspire a negative attitude. Attitudes regarding the interest and difficulty of programming became positive. Furthermore, attitudes regarding the interest, difficulty, and fun of turtle programming improved.

In VG, the results of A2 revealed a statistically significant difference in the AQ for Q1 (interest in programming, $p = 0.029$), Q6 (interest in turtle programming, $p = 0.008$), and Q8 (usefulness of turtle programming, $p = 0.045$), suggesting that the workshop increased interest in programming. On the other hand, there was no significant difference in TG. In both groups, the responses tended to differ significantly from the turtle programming-specific questions. In VG, the responses were more significant regarding the attitude toward programming than TG.

A3 involved a Wilcoxon rank-sum test of the BQ responses between the two groups. There were no significant differences, but marginal differences were observed for interest in programming ($p = 0.079$) and willingness to engage in turtle programming ($p = 0.069$). The marginal differences are attributed to the negative values in the BQ in VG.

The A4 analysis was the same as the A3 analysis, except that the AQ results were compared. The results were statistically insignificant.

The A5 analysis was carried out on the change in value. The change in the usefulness of turtle program was marginally significant ($p = 0.069$).

Overall, VG had a larger positive change in attitudes toward programming than TG. However, both VG and TG exhibited increase in interest in programming.

CHAPTER 3

After the workshop, both groups reported that programming is difficult. TG showed a very slight improvement in comparison to VG [TG (A1: -0.231) vs. VG (A1: -0.026)], but the difference was insignificant. However, the results imply that the text method has a larger effect on decreasing the difficulty level of programming than the visual input method.

Regarding the usefulness of programming, VG exhibited an improvement, whereas TG did not. However, the results did not differ significantly. The text input had more input responses than the visual input, which may have contributed to the decrease in TG.

Regarding the fun of programming, VG slightly increased, whereas TG slightly decreased. However, the difference was not significant. Similar to above, text input had more input responses than visual input, which have contributed to the decrease in TG.

As for willingness to engage in programming, VG improved, whereas TG did not. However, the difference was not significant. The decline in willingness in TG could be attributed to the decline in the fun of programming.

Both VG and TG exhibited increased interest in turtle programming, and the response for VG was statistically significant. Therefore, VG had greater interest in manipulating turtles using programming.

VG and TG both indicated that turtle programming was easier after the workshop, but the results were statistically insignificant. However, based on the results of A1, the value of evaluation changed considerably for TG. Therefore, TG tended to feel that programming is easier.

VG exhibited increase in the usefulness of turtle programming, whereas TG exhibited decrease. The difference was significant for VG (A2). Thus, the visual expression affected the evaluation: VG intuitively understood the turtle instructions from the illustration.

VG exhibited an improved willingness to use turtle programming, whereas TG did not. However, the difference between the two groups was not significant.

Based on these results, VG exhibited the most improvement, and the differences were often more significant than those in TG. These observations confirm hypothesis H3-1, which speculates that visual-based programming is adequate for introductory programming learning by novice learners. In addition, some

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS IN PROGRAMMING METHOD

learners in both groups exhibited reduced values, but the differences were not significant. In particular, many learners in VG commented that programming difficulty increased after the workshop, whereas many learners in TG indicated decrease in usefulness and willingness after the workshop. Because TG requires more input, it is possible that the learners had to take more time to program. Furthermore, the degree of difficulty for programming was more likely to change to a positive value for TG.

Table 3-4. Result of Arithmetic average (A1)

#	VG				TG			
	Before	After	CV ¹	Evaluati on	Before	After	CV	Evaluati on
Q1	4.842	5.211	0.368	Improve ment	5.346	5.615	0.269	Improve ment
Q2	3.737	3.711	-0.026	Improve ment	4.269	4.038	-0.231	Improve ment
Q3	5.316	5.368	0.053	Improve ment	5.577	5.462	-0.115	Degrada tion
Q4	5.105	5.289	0.184	Improve ment	5.500	5.423	-0.077	Degrada tion
Q5	5.079	5.289	0.211	Improve ment	5.538	5.385	-0.154	Degrada tion
Q6	4.684	5.263	0.579	Improve ment	5.192	5.269	0.077	Improve ment
Q7	3.868	3.816	-0.053	Improve ment	4.462	3.962	-0.500	Improve ment
Q8	4.711	5.184	0.474	Improve ment	5.154	4.808	-0.346	Degrada tion
Q9	5.026	5.263	0.237	Improve ment	5.308	5.423	0.115	Improve ment
Q10	4.842	5.237	0.395	Improve ment	5.385	5.231	-0.154	Degrada tion

¹CV = Change Value

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS
IN PROGRAMMING METHOD

Table 3-5. Result of A2 - A5

#	A2(VG)		A2(TG)		A3		A4		A5	
	S	p	S	p	S	p	S	p	S	p
Q1	32	0.029 **	5.5	0.143	-1.76	0.079 *	-1.05	0.293	0.40	0.687
Q2	157.5	0.891	27	0.338	-1.54	0.124	-0.71	0.477	0.96	0.339
Q3	53	0.672	13	0.863	-0.52	0.603	-0.04	0.967	0.03	0.978
Q4	23	0.2	18	1	-1.24	0.214	-0.57	0.566	0.05	0.956
Q5	22.5	0.183	20	0.427	-1.61	0.107	-0.01	0.989	1.00	0.315
Q6	30.5	0.008 **	35	0.439	-1.1	0.271	0.4	0.692	1.09	0.274
Q7	144.5	0.873	62	0.178	-1.46	0.145	-0.41	0.682	0.51	0.613
Q8	35	0.045 **	27.5	0.185	-0.9	0.371	1.13	0.257	1.82	0.069 *
Q9	42	0.151	42	0.5	-0.74	0.46	-0.51	0.613	0.1	0.924
Q10	34	0.070 *	42	0.5	-1.82	0.069 *	0.05	0.956	1.35	0.176

* = Significant trend, ** = Significant difference

3.4.2. Understanding Programming

3.4.2.1. Problem results and analyses

I used tests and questionnaires to confirm the comprehension level of novice learners. There were six questions (Table 3-1) and one free problem. Each learner self-declared when a problem was complete, and then took a screenshot to confirm the solution. In addition, I acquired the source code as part of the answer. Figure 3-7 shows the response rate. A low response rate was a problem. There was not any difference in P1 by the group. For P2, the percentage of correct answers was higher for VG than TG. This difference is attributed to the amount of input required to program. TG returned a higher percentage of correct answers than VG for P3, which was about loop statements, indicating that the operation amount (i.e., input amount) of VG had increased. Consequently, the correct answer rate decreased for VG. The result of P4 was the same as that of P3. On the other hand, P5, which was about conditional statements, had the opposite result, i.e., VG had a higher percentage of correct responses than TG. Because complicated conditional expressions had to be input for TG, it was more difficult to obtain a correct response in TG than in VG. The result of P6 was the same as that of P5.

In the free problem, the student was required to create a single letter of the alphabet. Figure 3-8 shows the answer to the free problem. Both groups utilized many iterations, indicating that a conditional branch is a difficult concept to understand. The differences between groups were statistically insignificant, confirming that the abstraction level of the visual language was similar to that of the text language. However, some learners in both groups were unable to solve the free problem.

3.4.2.2. Description formula questionnaire result and analysis

Q11 and Q12 used the description formula questionnaire (Table 3-2). Table 3-6 shows the answers to the questionnaire. The answers were grouped into four categories: "Explain in relation to game events (CTG1)", "Explain the action in words (CTG2)", "Associate with a programming language (CTG3)", "Unanswered · Unknown · Other (CTG4)". "Explain in relation to game events" indicates that an answer was created in association with Minecraft, e.g., "Avoid certain blocks the using turtle". Many responses to Q11 and Q12 by the VG group fit into this category, but this response was rare in TG. It is possible that VG applied this category more often because the expression of the programming language used for visual input was easy to imagine as an event of the game. "Explain the action in words" indicates that the answer was explained using words without being related to game events. In VG, many learners' responses to Q11 and Q12 fit into this group. Even in TG, few learners fit into this group. "Associated with a programming language" indicates that the answer was derived from the programming language, e.g., "for x = 1, ~ do ~ end". Responses for both groups fit this category, but more from TG. It is possible that TG grasped the meaning of the question in the programming language. "Unanswered · Unknown · others" indicates users who did not respond or indicated that they were unsure. Impressions include, "I do not know" and "It is difficult". This category applied to learners in both groups, but more from TG. Thus, TG may have had more difficulty verbalizing concepts or understanding programming.

3.4.2.3. Summary of results

The results do not confirm H3-2. The results for the loop problem (Q3 and Q4) were better for TG than for VG. On the other hand, the results for the conditional problem showed the opposite trend (Q5 and Q6). In addition, the manipulated variables and input quantities in each input method may influence the correct answer rate.

Q11 and Q12 reveal that the type of response related to programming concepts differs according to the programming input method. It is possible that the expression method of the programming language has a significant influence in this regard. Because both groups responded similarly to the question about the description formula, I believe there no substantial difference in the degree of understanding of programming as a function of the input method.

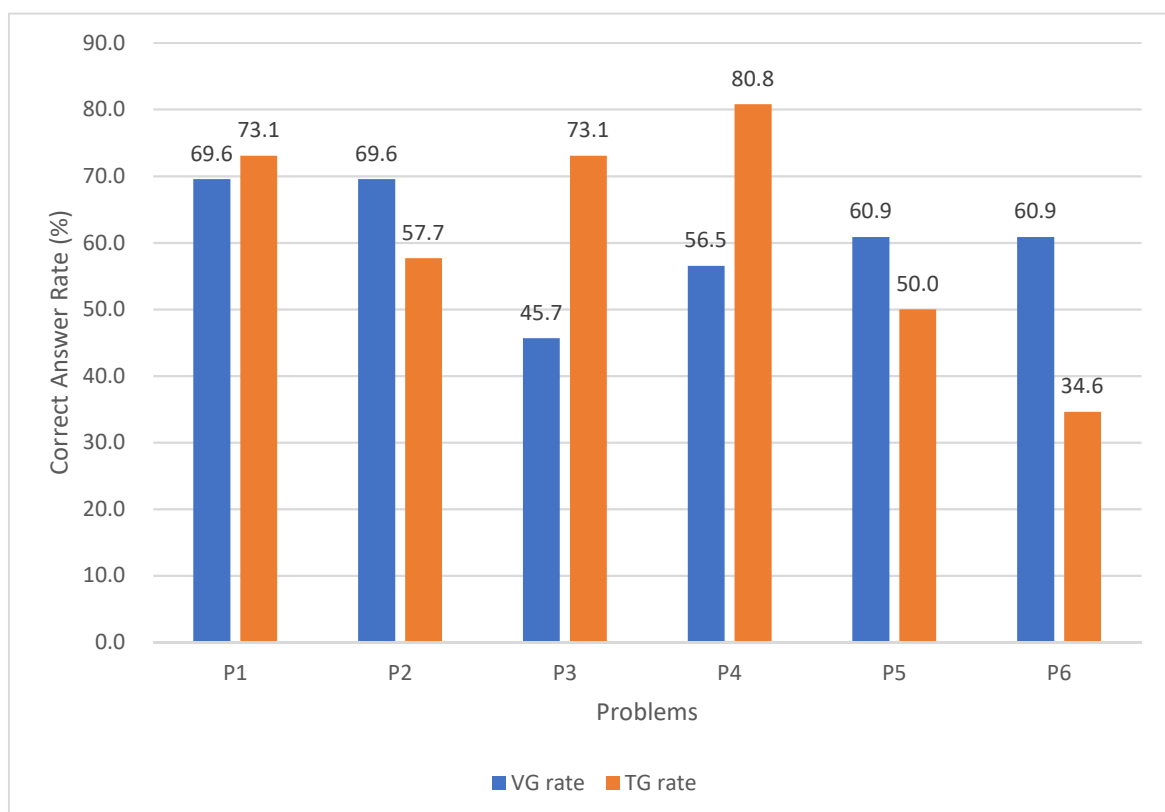


Figure 3-7. Six problems Response Rate

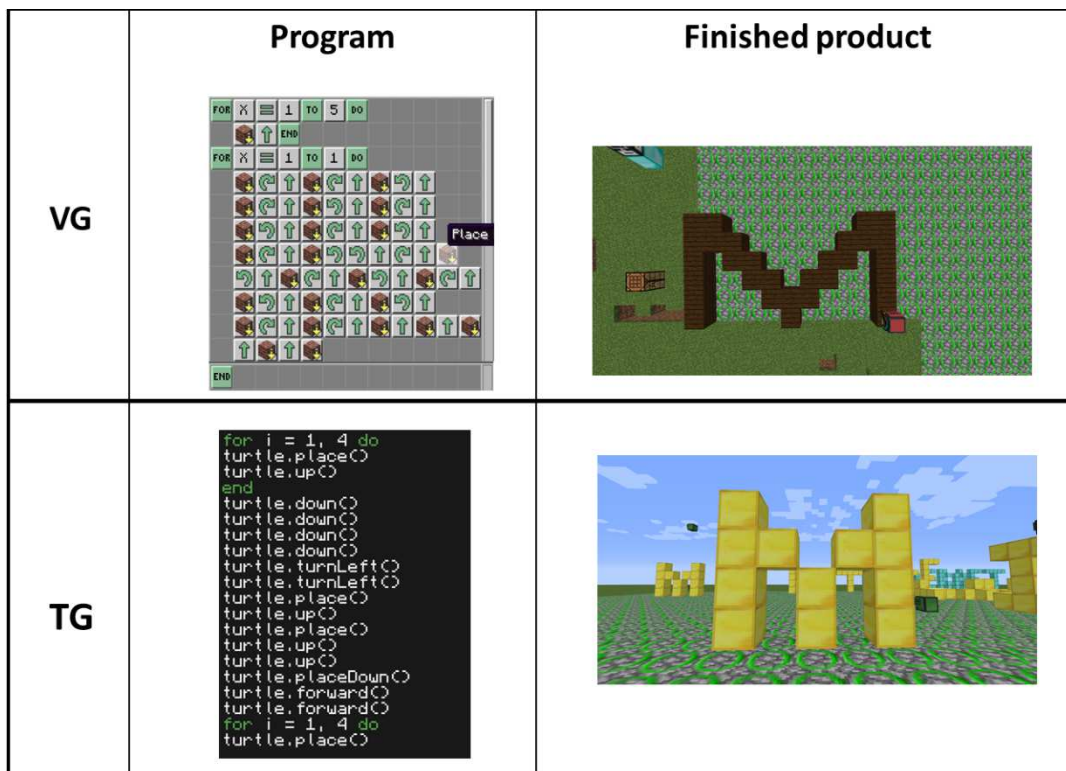


Figure 3-8. Result of Free Problem

Table 3-6. Description formula questionnaire result

	What is a conditional?			What is a loop?		
	VG (%)	TG (%)	Answer Example	VG (%)	TG (%)	Answer Example
CTG1	19.57	3.85	Avoid certain blocks the using turtle	17.39	0	Process to stack many blocks
CTG2	36.96	26.92	If there is ~, run the program.	34.78	15.38	Repeat as many times as it was said
CTG3	13.04	23.08	if ~ then ~ end	10.87	30.77	for x = 0, ~ do ~ end
CTG4	30.43	46.15	I am difficult	36.96	53.85	I do not know

3.5.Discussion

3.5.1. Result of RQ3-1

- RQ3-1: Does the visual-based input method induce a different attitude toward programming than the text-based input method?

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS IN PROGRAMMING METHOD

- H3-1: The visual input programming lecture induces a larger change in attitude toward programming.

In RQ3-1, there is a difference between VG and TG in terms of visual expression. The results of this study also differ from those of previous research. Zorn et al. used a mod of CodeBlock for a student lecture course in 2013 [6]. Their research, which compared the learning effect of block programming to that of text programming, found that block programming increases student interest. In our research, VG exhibited statistically significant differences in the interest and usefulness of turtle programming. VG also exhibited statistically significant differences in the interest of programming overall. These results indicate that visual inputs are likely to increase interest in programming. In addition, our results showed that VG increased usefulness and willingness. This may be because VG is more intuitive than TG. Also, because a keyboard was not used in VG, less time is necessary to see results.

TG had no statistically significant difference in some of the analyses. However, the change in the arithmetic mean indicated that the difficulty of programming improved more in TG than in VG. A previous comparison study that investigated programming difficulty [39] revealed that a novice cannot distinguish the cause of programming difficulties because they do not recognize challenges that arise due to differences between interfaces. By contrast, our study revealed a difference in attitude. This difference may be due to the fact that text input is a more realistic programming method than visual input. Learners may have a prejudice that text is more representative of programming, which is perceived as more difficult. However, our workshop interposed games, creating the possibility that learners would feel that programming is easy. Consequently, programming difficulty exhibited a larger improvement in TG.

The visual input method improved attitudes towards programming to a greater extent than the text input method. Although the differences were not statistically significant, I can confirm H3-1. Thus, from the standpoint of the attitude of novice learners toward programming, the visual input method is more suitable. However, the text input method decreases the difficulty level more than the visual input method. Consequently, the text input method can be adapted to novice learners. Data accumulated in future studies should further distinguish between the two input methods.

3.5.2. Result of RQ3-2

- RQ3-2: Does the understanding of programming differ between visual-based and text-based input methods?
 - H3-2: Programming is easier to understand using a visual input method.

For RQ3-2, the low response rate was an issue for both groups. VG had a high percentage of correct answers regarding the conditional problem. This is attributed to the fact that the visual method requires less input to create the conditional program. Furthermore, visual input allows the conditional to be viewed as images instead of text. The score exhibited a larger improvement in VG than in TG. In addition, some VG learners could not solve the free problem. More answers used loops than conditional branching, suggesting that loops are conceptually simpler than conditional branching. TG had a high percentage of correct answers for the loop problem because less input is required to create loops with text inputs. Hence, the score exhibited a larger improvement in TG than in VG.

Many responses in VG used the same loop for the free problems. Based on these findings, it can be assumed that both groups are influenced by the operations and input quantity in the environment. In addition, the results also support the notion that a loop is a simpler concept than conditional branching. This result suggests that the expression of a programming language influences learners' understanding level if DCT is considered [27][30].

Furthermore, the rate of correct response to the problem regarding programming indicates that both methods are useful. As discussed above, the results indicate that loops are a simpler concept than conditional branching. Therefore, both methods can be applied to novice learners. The programming input method and input quantity may influence the rate of correct answers to the problem about the understanding of programming (Table 3-1). In the questionnaire about programming concepts, VG exhibited a larger improvement than TG. Consistent with previous research [28][35], this result suggests that visual inputs are beneficial for novice learners.

The answer to RQ3-2, H3-2, cannot be confirmed using the results of this study. The two groups exhibited a clear difference in their understanding of programming. The rate of correct response to the problem regarding

programming indicates that both methods are useful. Thus, both methods can be applied to novice learners.

3.6.Limitations

This research had several limitations, some of which could be addressed by future research. Here, I note five limitations:

- 1) The response rates to the problems confirming the degree of understanding (P1–P6) were low due to self-assessment. Although implementing a paper test could increase the response rate, it may not resolve this issue. I am currently considering other options.
- 2) Because participants were recruited via the Internet, there was a difference in the number of participants in the two groups. Participants selected their group (visual or text) when volunteering for the study. This difference is likely due to the perception, at the time of recruitment, that the text method would more difficult. To address this imbalance, in the future each group should have roughly the same number of participants.
- 3) Because the participants were recruited via the Internet, learners were able to select the programming method, and could register for either the visual or text workshop. In the future, participants should be randomly assigned to each method.
- 4) Novice learners were recruited online. However, some participants may have had some previous exposure to programming, which may have affected the results, especially the understanding of programming concepts. In the future, filtering and other adjustments will be conducted to decrease the exceptions of participants.
- 5) The small population size may have affected our results. In the future, more experimental data should be accumulated.

3.7. Conclusion

I investigated whether a text or visual input method is better for novice learners. In the field of DCT and multimedia, it has been reported that visual expressions, as well as the application of text and images in a balanced manner, are effective for novice learners. Hence, it may be beneficial to teach programming visually. However, programming involves behavioral aspects, such as entering and executing programs. Because information is acquired by more than just vision, programming differs in several aspects of multimedia learning. Some previous studies have applied and compared programming learning methods for novice learners, but their results did not clarify whether visual or text input is more suitable for novice learners. Therefore, we compared the learning effects of two input methods for novice learners using ComputerCraftEdu in Minecraft. The visual input method resulted in a larger change in attitude. Significant differences were noted, especially in regard to interest in programming (including Turtle programming). Although text input tended to make programming less difficult, the difference was not significant.

In rate of correct response to the problem assessing the understanding of programming (Table 3-1), there was a difference between conditional branching and loops. The rate of correct response to the conditional problem was higher for visual input, whereas the rate for the loop problem was higher for text input. I speculate that the differences are influenced by operations and input quantity, but additional studies will be necessary to definitively determine the cause. In addition, differences were observed in the questionnaire results regarding the programming concept. VG tried to explain the concept by applying it to a specific action, whereas many TG in tried to explain the concept in relation to programming. Thus, the expression method of programming language may influence the perception of concepts.

The overall results indicate that the visual input method is better suited for an introduction to programming. These results coincide with the DCT, implying that it is easier to use a visual input method. However, the comparison results suggested that actions change the learning effect. Hence, from the viewpoints of the amount of operations and input in the programming environment, the text input method can be used for programming learning by novice learners. In the future, I plan to investigate the learning effect from the perspective of behavior recognition. Furthermore, I plan to collect and analyze additional data, as well

COMPARISON OF LEARNING EFFECTS OF TEXT AND VISUAL REPRESENTATIONS
IN PROGRAMMING METHOD

as determine the correlation between attitudes and understanding of programming.

CHAPTER 4

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF PROGRAMMING LEARNING ENVIRONMENTS

This chapter investigates the characteristics of multiple environments, as well as the learning effect of each environment. In this chapter, I consider the learning effect based on the characteristics of programming constructs and game elements, in addition to the characteristics discussed in Chapter 3

Each environment has unique characteristics. Several studies have evaluated various environments [53][54][55][56][57], but the learning effects due to the characteristics of a given environment have yet to be sufficiently examined.

To address this issue, I investigated the following Research Questions (RQs):

- Research Question 4-1 (RQ4-1): Is there a difference in characteristics between programming environments?
- Research Question 4-2 (RQ4-2): Does the programming environment influence the learning effect?
- Research Question 4-3 (RQ4-3): Is there a relationship between the characteristics of an environment and the learning effect?

RQ4-1 determines whether each environment has unique characteristics. Because the most appropriate environment for the intended purpose can be selected based on the desired characteristics, RQ4-1 should enhance the effectiveness of applying environments. RQ4-2 evaluates the influence of each environment on the learning effect. RQ4-3 elucidates how learning effects are related to the characteristics of each environment. Understanding the learning effect from these perspectives will aid in selection of the appropriate environment based on learning objectives and goals.

4.1. Background

Programming learning for beginners has been conducted using various learning environments. For example, Scratch [3][8] is used in a visual programming language, while CodeCombat [9] and Minecraft Education Edition [11] exist within game software. These environments have different characteristics, including program expression and programming method. For example, program expression can be text, visual, etc. A previous study on multimedia learning revealed that learner recognition and learning effects differ between text expression and image expression [30].

In addition, these environments differ widely in terms of developers' intentions and learning objectives. Although many researchers have investigated programming learning environments (e.g., evaluation of a single environment [40] and comparisons between text and visual languages [36]), few studies have compared programming learning environments in multiple fields. Therefore, the types of learning effects that depend on the characteristics of the programming learning environment remain unknown.

In this research, I evaluated environments with three different programming methods [visual programming languages, game software, and physical environments (tangible [72] and unplugged)] in the same framework, using a workshop.

4.2. Programming Learning Environments

I selected six environments that are commonly available in Japan.

4.2.1. Scratch

Scratch [71] (Sc, Figure 4-1) is a visual language used to create stories, games, and animations. This globally popular environment was developed by the MIT Media Laboratory. Some previous studies [4][5] have used this environment.

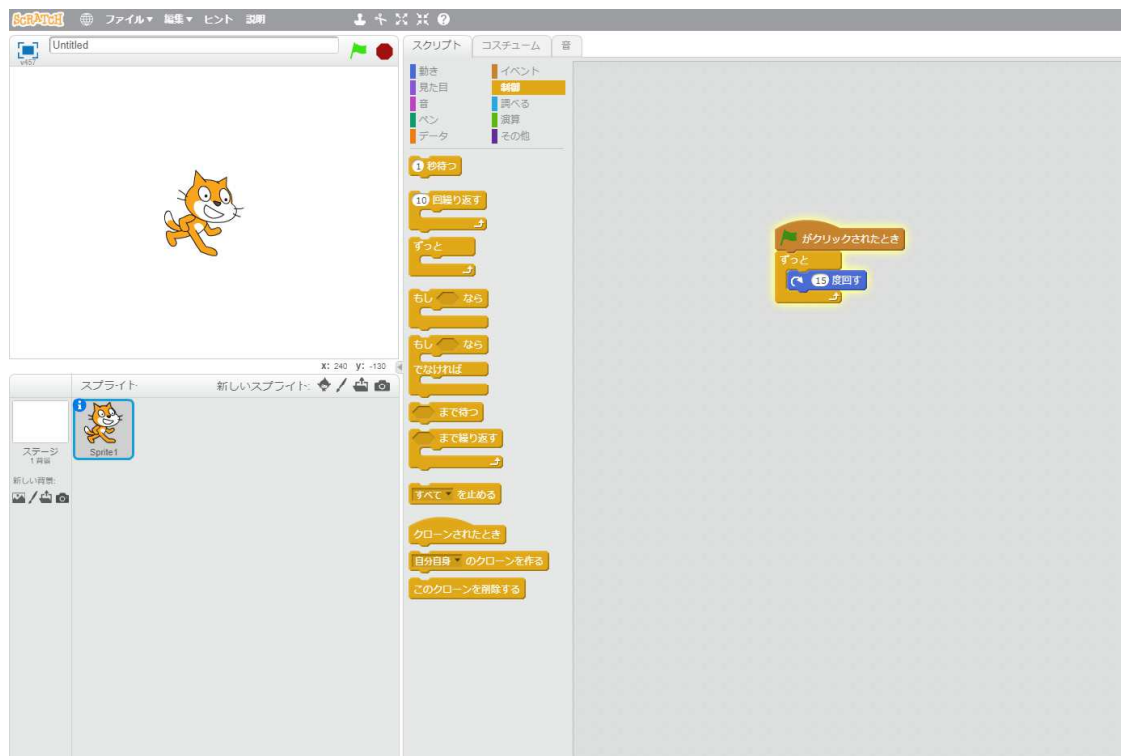


Figure 4-1. Scratch [71]

4.2.2. Viscuit

Viscuit [41] (Vi, Figure 4-2) is a Visual Programming Language and Environment developed by Digital Pocket in Japan. It can control a written illustration using a special form of programming called "glasses".

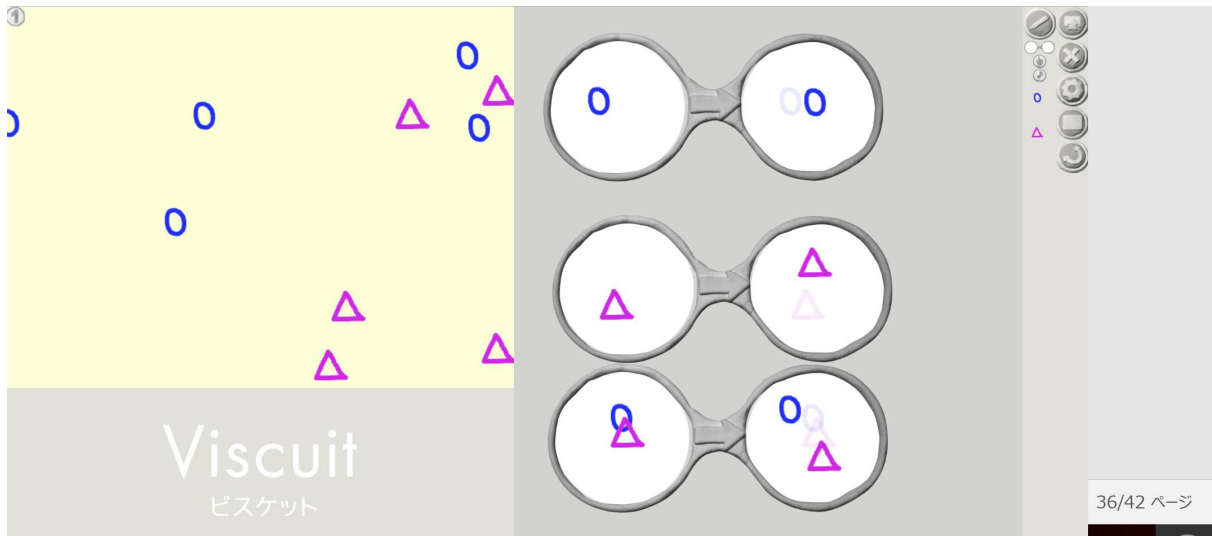


Figure 4-2. Viscuit [41]

4.2.3. CodeMonkey

CodeMonkey [42] (CM, Figure 4-3) is game software used to program the behavior of a monkey collecting bananas. This game uses a programming language called CoffeeScript.



Figure 4-3. CodeMonkey [42]

4.2.4. Lightbot

Lightbot [14] [60] (Li, Figure 4-4) is game software used to program the behavior of a robot to achieve a goal. It teaches the concept of recursion as a "Loop".

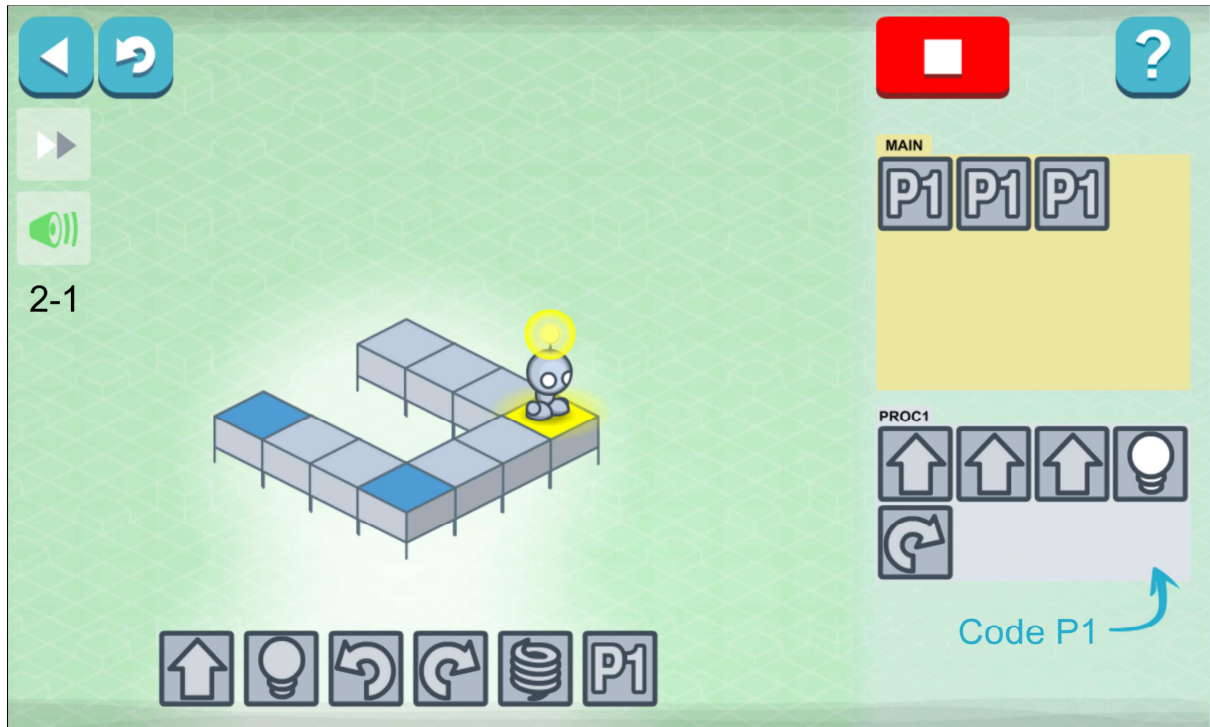


Figure 4-4. Lightbot [14]

4.2.5. OSMO Coding

Osmo Coding [43] (OC, Figure 4-5) is a tangible device. It uses physical blocks for programming to control characters via an iPad application.

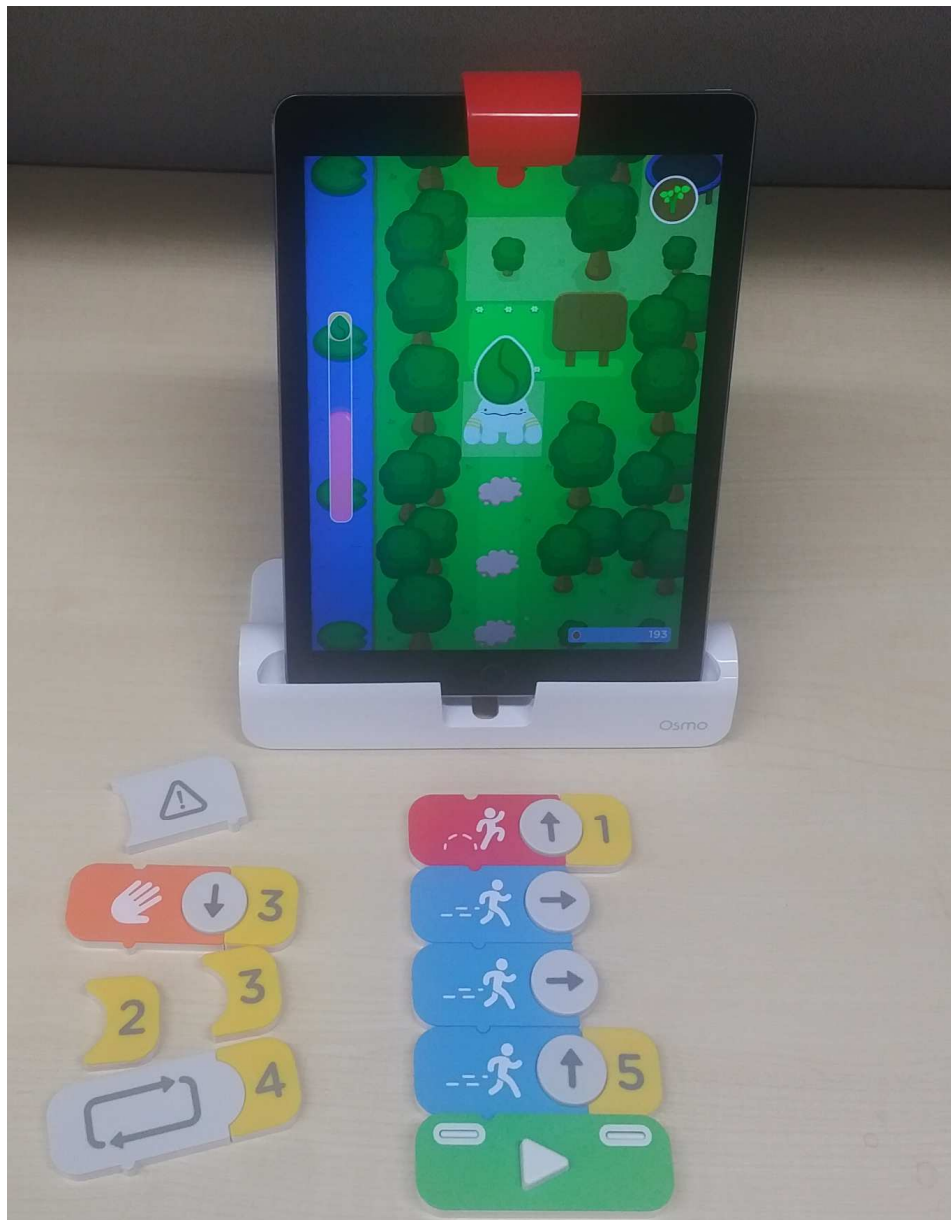


Figure 4-5. OSMO Coding [43]

4.2.6. Robot Turtles

Robot Turtles [44] (RT, Figure 4-6) is a board game in an unplugged tool. The purpose is to create a program to manipulate the turtle and collect jewels.



Figure 4-6. Robot Turtles [44]

4.3. Classification

These six environments can be divided into three fields, based on their characteristics: visual programming environment, game software, and physical environment. In addition, I qualitatively evaluated the environments based on the taxonomy described in Chapter 2. The results are shown in Table 4-1.

The visual programming environment uses a visual programming language within a programming method with a drag-and-drop feature. This feature allows content to be freely created. Viscuit and Scratch are visual programming environments, and the main difference between them is the expression of code. Scratch is expressed in text, whereas Viscuit is expressed in images.

Game software is software with game elements, including Rules/Restrictions, Goals, Rewards, and Cooperation [15][16][17]. Lightbot and CodeMonkey are game software, and these environments differ in both the expression of code and the programming method. Lightbot expresses code in images, and programming is performed by drag-and-drop. In contrast, CodeMonkey uses text to express code, and programming is performed by typing the code.

A physical environment is one that allows programming using physical cards or blocks. OSMO Coding and Robot Turtles are examples of physical environments that differ in the location of the program execution results. In OSMO Coding, the result of programming is reflected in the software, i.e., the program works in a virtual space. On the other hand, the execution result of Robot Turtle is reflected by the behavior of a piece on a board game. In other words, the program works in real space.

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF PROGRAMMING LEARNING ENVIRONMENTS

Table 4-1. Classification Result

Category	Characteristic	Sc	Vi	CM	Li	OC	RT
style of programming	procedural			x	x	x	x
	functional						
	object-based	x	x				
	object-oriented						
	event-based	x					
	statemachine-based						
programming constructs	conditional	x		x	x	x	
	loop	x	x	x		x	
	variables	x		x			
	parameters	x		x		x	
	procedures/methods	x		x	x		x
	user-defined data types						
	pre and post conditions						
	recursion				x		
Representation of code	text	x		x			
	pictures		x		x		
	flow chart						
	animation						
	forms						
	finite state machine						
	physical objects					x	x
Construction of programs	typing code			x			
	assembling graphical objects	x	x		x		
	demonstrating actions						
	selecting/form filling			x			
	assembling physical objects					x	x
Support to understand programs	back stories			x		x	x
	debugging						
	physical interpretation	x		x	x	x	x
	liveness		x				
Designing Accessible Languages	generated examples						
	limit the domain	x	x	x	x	x	x
	select user-centered keywords						
	remove unnecessary punctuation						
	use natural language						
Game elements	remove redundancy						
	Rule/Restriction			x	x	x	x
	Goal			x	x	x	x
	Rewards			x		x	
	Cooperation			x			x
Supporting Language	Japanese	x		x		x	
	English	x		x	x	x	x
	others			x			
Operating Environment	Windows	x	x		x		
	Mac	x	x		x		
	Linux	x					
	Web	x	x	x	x		
	iOS				x	x	
	Android				x		
	others				x		x
Interface	PC	x	x	x	x		
	Tablet(8inch~)	x	x	x	x	x	
	Smartphone				x		
	others						x
Experience	unnecessary	x	x	x	x	x	x
	necessary						

4.4.Experiments

4.4.1. About Experiments

To evaluate the six environments, I focused on the understanding of basic programming concepts (sequential execution, repetition, conditional) and the influence of applied skills (especially, abstraction and problem solving) in a workshop. In addition, I researched attitudes toward programming using a questionnaire and an eight-point learning comprehension test (programming basics and applied programming test).

4.4.2. Questionnaire and test


I conducted a questionnaire and a test to analyze the learning effect.

4.4.3. Learning comprehension test

The test to investigate the influence of the environment on the understanding of programming consisted of eight questions:

- Sequential: one question
- Repetition: three questions
- Conditional: two questions
- Free description problem: two questions

Figure 4-7 and 4-8 show the types of questions asked.

6							
5							
4							
3							
2							
1							
	1	2	3	4	5	6	

Q1 Please freely draw a line so that the robot passes through all the squares. At first it is facing right.

Q2 Please explain with a simple program why you drew such a line.

Figure 4-8. Free description problem

4.4.4. Questionnaire about the attitude toward programming

This questionnaire was conducted before and after the workshop to investigate changes in attitudes toward programming: fun (Q1A, Q1B), difficulty (Q2A, Q2B), usefulness (Q3A, Q3B), willingness (Q4A, Q4B), and interest (Q5A, Q5B)]. Responses were on a six-stage Likert Scale (1: Strongly disagree, 2: Disagree, 3: Somewhat disagree, 4: Somewhat agree, 5: Agree and 6: Strongly agree). Based on Chapter 3, the questionnaire consisted of the following five questions:

- Q1: Do you think programming is fun?
- Q2: Do you think programming is difficult?

- Q3: Do you think programming is useful?
- Q4: Do you want to learn programming?
- Q5: Are you interested in programming?

4.5. Workshop

The workshop system was organized by two to four persons, including lecturers and assistants. The learners were elementary students in grades 3 to 6, except for learners using Robot Turtles, who were in grades 1 to 3 at an elementary school where the environment was announced officially as a subject. The teaching materials included online environments, handouts, etc.

4.5.1. Schedule of the workshop

The workshop lasted 90 minutes with the following format:

1. Pre-Questionnaire: 2 min;
2. Pre-Test: 5 min;
3. Workshop Time: 75 min;
4. Post-Test: 5 min;
5. Post-Questionnaire: 3 min (+5 additional minutes allowed)

4.5.2. Number of students and effective questionnaire responses

Fifty-nine students participated in the workshop using the following environments: Scratch (10 people), Viscuit (9), CodeMonkey (9), Lightbot (7), OSOMO Coding (16), and Robot Turtles (8). The numbers of valid responses to the test and questionnaire were as follows:

- Learning comprehension test: 45 people

- Questionnaire of attitude toward programming: 49 people
- Questionnaire on impressions: 49 people

4.6.Results and Analysis

4.6.1. Learning comprehension test

4.6.1.1. Overall test results

First, I analyzed three groups: visual programming environment, game software, and physical environment. Figures 4-9 – 4-11 show the learning comprehension test results, by group. Each group exhibited improved learning comprehension after the workshop. For each result, the prior and posterior scores were evaluated using the Wilcoxon signed-rank test (confidence interval 95%; $p < 0.05$ indicates a significant difference). Table 4-2 summarizes the results.

The visual group improved as a whole, with a Wilcoxon signed-rank test p-value of about 0.08. Although the difference was not significant, the trend indicates that the workshop was effective. However, a few learners had reduced scores after the workshop. One reason for a lower score might be that learners became tired of learning in the visual programming language and stopped taking the test.

The game software group exhibited a large improvement in learning. The Wilcoxon signed-rank test had a p-value of about 0.006, which is statistically significant. Game elements provide an explanation for the significant difference: because the goal in a game is clear, the students are engaged until the test was complete. However, it is possible that the scores improved because the problems asked in the test were similar to those in the game software.

Similarly, the learning effect improved in the physical environment group after the workshop, although the change was not significant (Wilcoxon signed-rank test p-value of 0.28). The scores of some learners declined after the workshop, possibly because of the difference in work volume due to physical intervention.

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF PROGRAMMING LEARNING ENVIRONMENTS

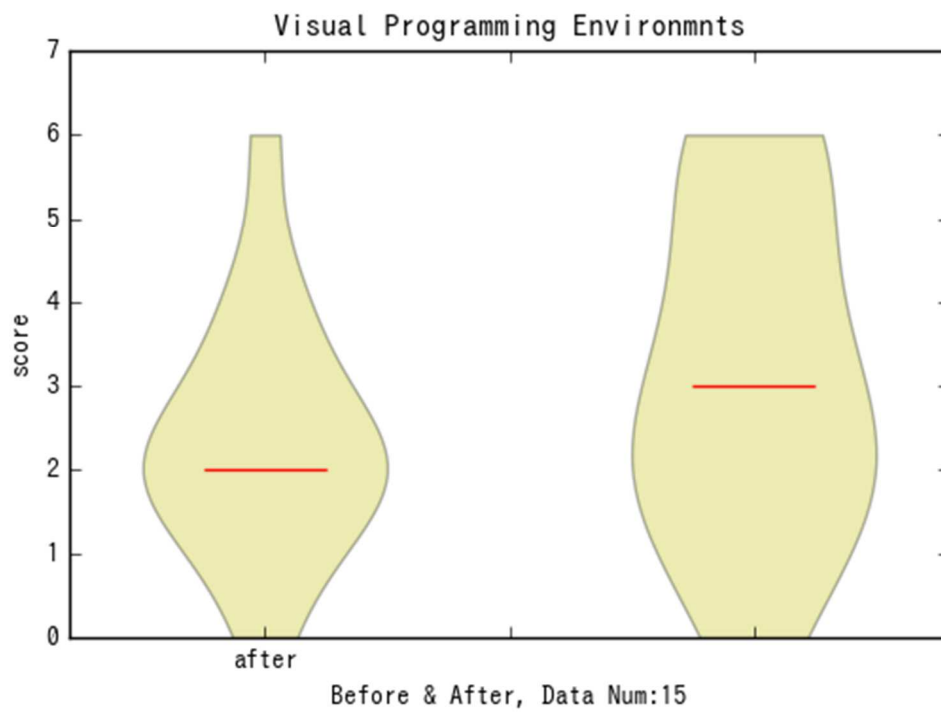


Figure 4-9. Results of Visual Programming

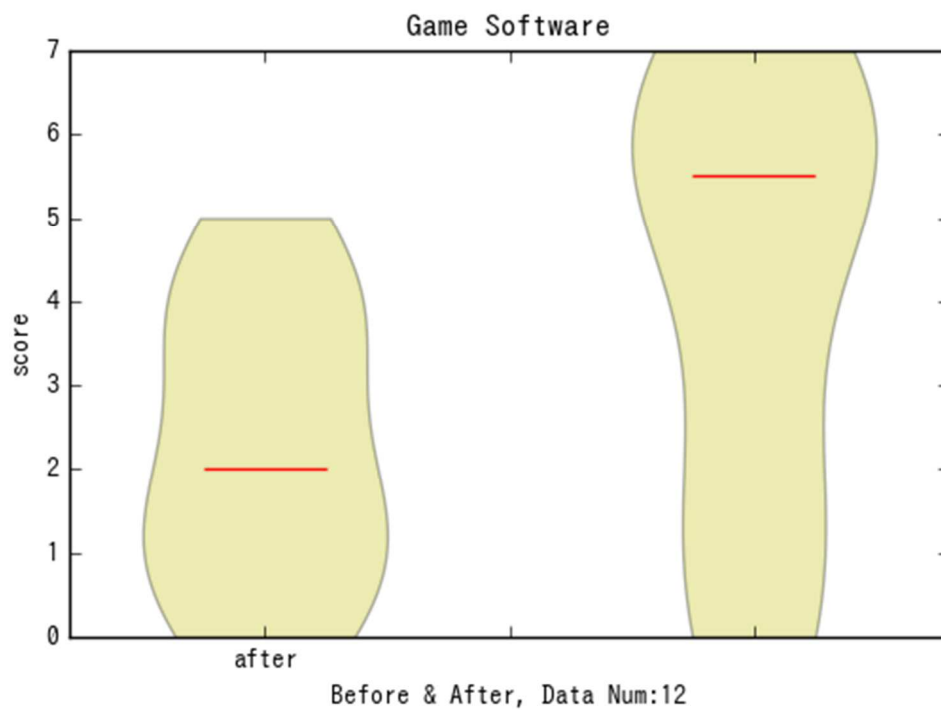


Figure 4-10. Results of Game Software

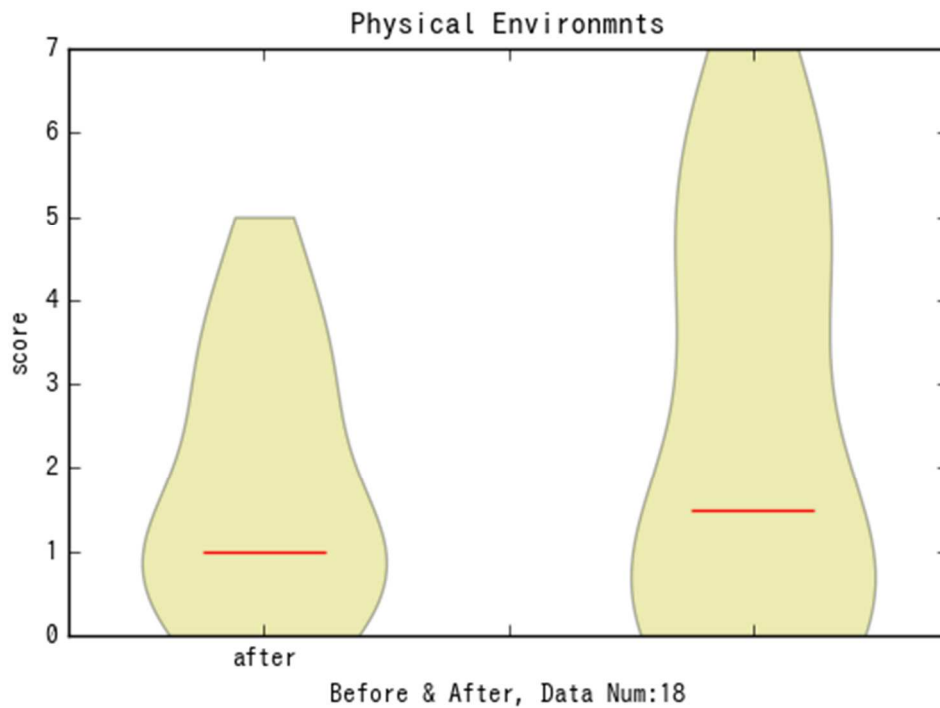


Figure 4-11. Results of Physical Environment

Table 4-2. Results of The Significant Difference Test (Learning Comprehension Test)

Category	Statistics	p-value
Visual Programming Environments	17.5	0.0834 *
Game Software	0	0.0059 **
Physical Environments	30	0.2752

** Significant difference, * Significant trend

4.6.1.2. Programming applied test

Two patterns emerged in the responses to the free description questions. The descriptive patterns were either U-shaped (Figure 4-12, left) or spiral-shaped (Figure 4-12, right). Because both were correct due to problem solving, it is possible that learners improved their problem-solving abilities and explanatory skills. The spiral type can be simply described using a small number of procedures and components. Therefore, the improvement may have been due to an enhanced abstraction ability. Interestingly, only the Viscuit participants

responded using a spiral, suggesting that Viscuit may have features not found in the other environments.

None of the learners could explain the program prior to the workshop, and only a small number could after workshop. Furthermore, the differences among the environments were not significant. For example, learners felt that they "wanted to proceed until hitting the wall".

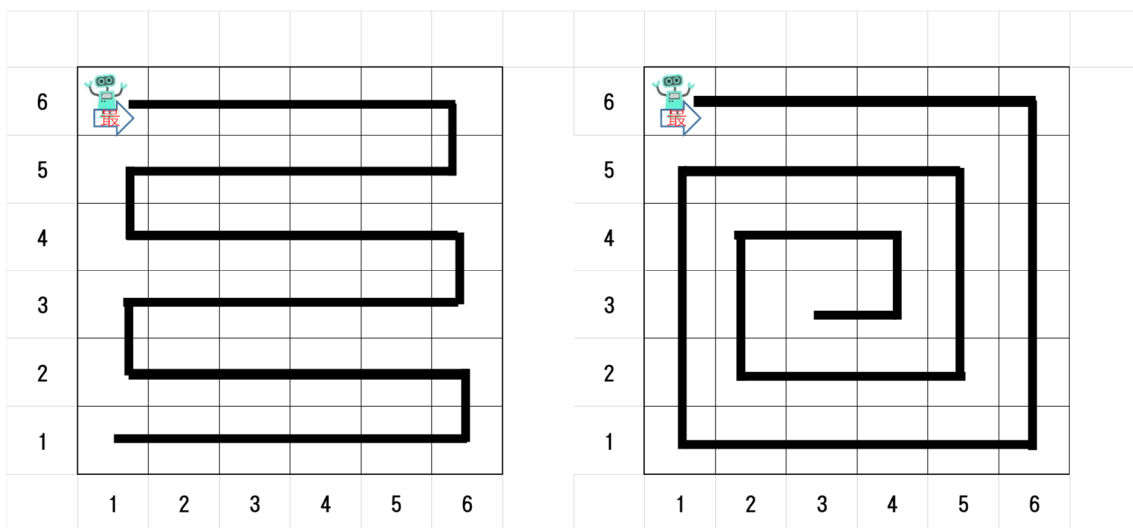


Figure 4-12. Results of the Free Description Problem

4.6.2. Attitude toward programming

Figures 4-13 – 4-15 show the results of the questionnaire regarding attitudes about programming, by the group. Table 4-3 shows the results of the Wilcoxon signed-rank test.

If the environment included game elements, interest in programming improved in the after workshop, likely because games are more fun than physical environments with game elements. We evaluated the significance of the difference using the Wilcoxon signed-rank test. The p-value for interest in the game software group is about 0.06, indicating a significant trend. From a comprehensive viewpoint, game elements make programming seem more interesting.

Visual programming languages tend to decrease the difficulty of programming; learners can easily create software by visual programming because it is consistent with the general image of programming. The visual programming group had Wilcoxon signed-rank test p-value of approximately 0.09, a slightly significant trend. Both the game software and physical environment groups felt that programming was more difficult after the workshop. For the game software group, the p-value was about 0.07.

The visual programming language and physical environment group indicated that the workshop did not increase their perception of the usefulness of programming. However, the game software group reported increased value of usefulness after the workshop. This difference may be because the game software is instantaneously executed, yielding a concrete result. However, the Wilcoxon signed-rank test indicated an insignificant difference between the groups.

Each group exhibited a similar willingness to learn, and the Wilcoxon signed-rank test indicated no significant differences. This workshop included a short introduction, which had a negligible effect on willingness. Depending on the environment, some learners reported decrease in willingness after the workshop; the reasons for this need to be considered further.

Each group reported a slight improvement in interest in programming. Although the students only studied programming for a short time, their interest improved. However, the Wilcoxon signed-rank test did not confirm a significant difference.

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF PROGRAMMING LEARNING ENVIRONMENTS

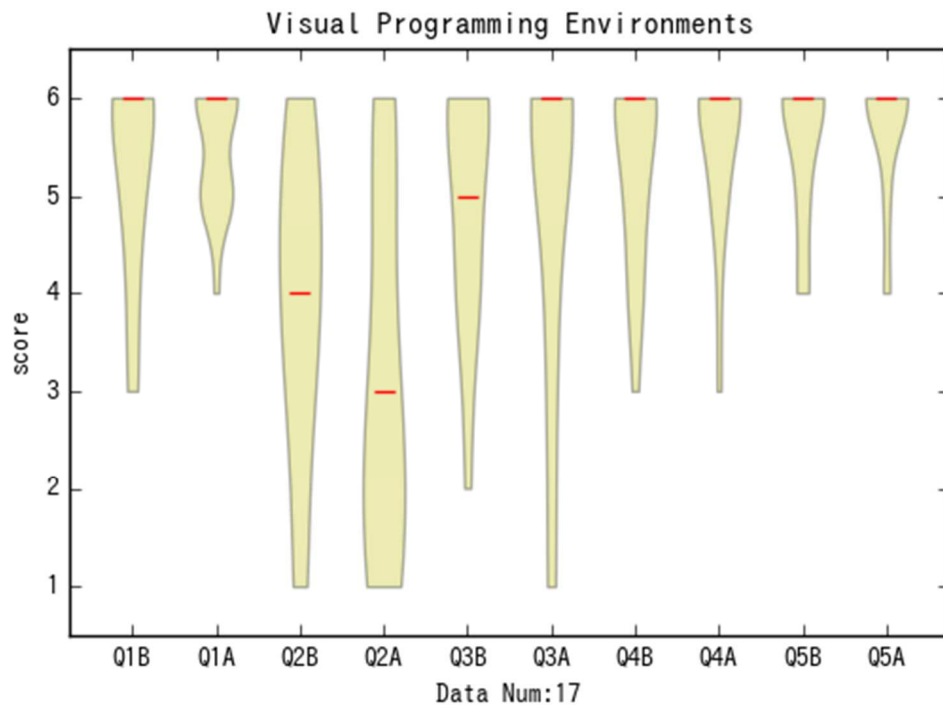


Figure 4-13. Results of visual programming language

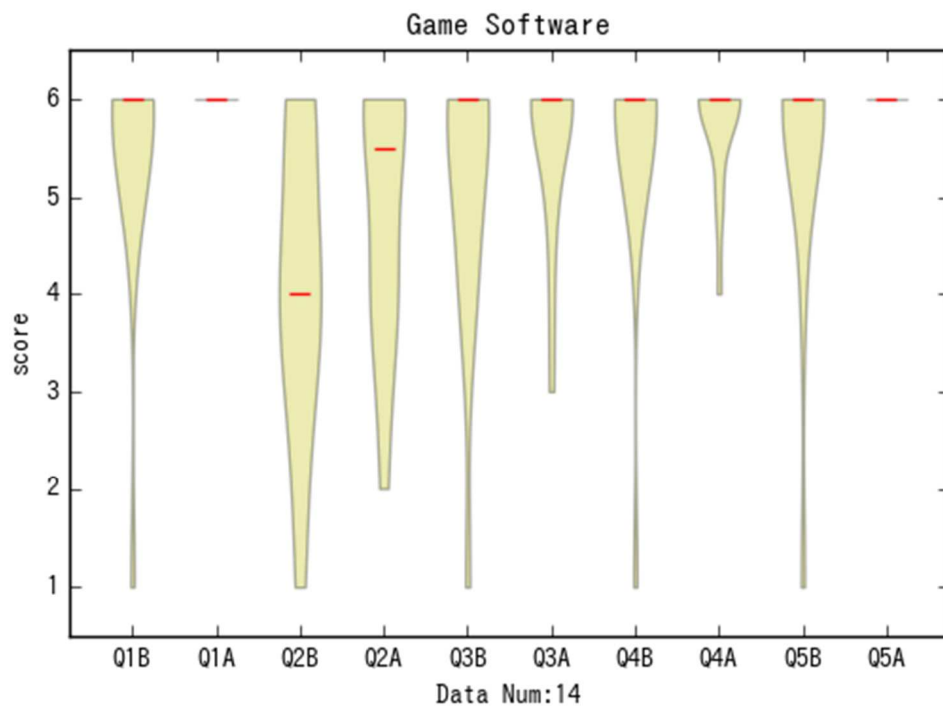


Figure 4-14. Results of game software

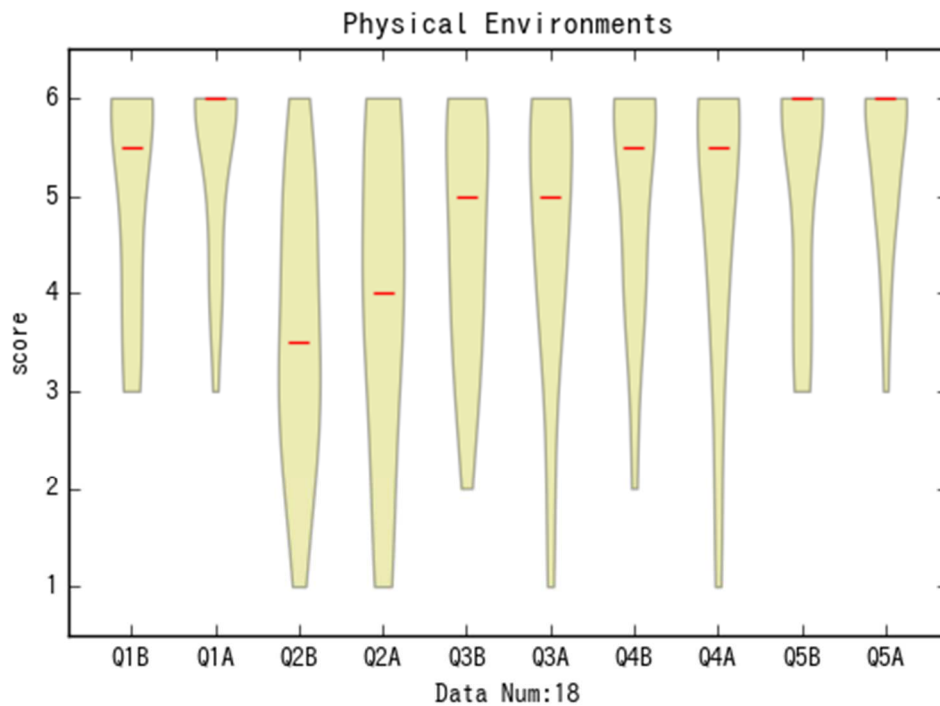


Figure 4-15. Results of the physical environment

Table 4-3. Results of The Significant Difference Test (Attitude Toward Programming)

	Visual language		Game Software		Physical environment	
	Statistics	p-value	Statistics	p-value	Statistics	p-value
Q1	6.000	0.160	0.000	0.059*	2.000	0.131
Q2	17.500	0.087*	0.000	0.066*	26.000	0.522
Q3	11.000	0.608	1.000	0.285	30.500	0.813
Q4	5.500	0.279	7.500	1.000	4.500	0.854
Q5	6.000	0.317	0.000	0.109	2.500	0.157

* Significant trend

4.6.3. Comparison of the characteristics in individual Environments

Table 4-4 overviews the characteristics of each environment. In addition, the questionnaire results on the impressions about each environment are considered.

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF PROGRAMMING LEARNING ENVIRONMENTS

The tools are divided into populations to analyze each characteristic individually using the before questionnaire/test results and the after questionnaire/test results. In addition, the results of the comprehension test and the questionnaire on attitudes towards programming are analyzed separately. Table 4-5 shows the analysis results of the relationship between each characteristic and the comprehension test, while Table 4-6 shows the relationship between each characteristic and the attitude questionnaire. These tables use the average point change (Ac) and the p value (p) of the Wilcoxon code rank test for each population. Table 4-7 summarizes the results of the learning effects for each characteristic (Programming constructs, Representation of Code, Construction of Programs, and Game elements) of the programming environments. These results are analyzed using the average value of the understanding of a programming concept (excluding the free description problem) and the results of the attitude questionnaire. "x", "xx", and "xxx" denote a change in the mean value, a significant trend in the significant difference test, and significance in the significant difference test, respectively. Programming constructs promote the understanding of each programming concept. In particular, the characteristic of a loop helps comprehend the concept of iteration. Moreover, the characteristic of recursion may promote the understanding of iteration and conditional branching.

In Representation of Code, text representation reduces programming difficulty. The Wilcoxon signed-rank test, which was conducted using the attitude questionnaire results in an environment where the representation of text is given as a population, indicates that the change in the degree of difficulty shows a significant trend. Hence, the representation of text reduces the difficulty level. The Wilcoxon signed-rank test, which was conducted with the attitude questionnaire results in an environment where the representation of a picture is given as a population, indicates that fun and willingness exhibit significant trends, and interest displays a significant difference. The Wilcoxon signed-rank test, which was conducted with the attitude questionnaire results in an environment where the representation of physical objects is given as a population, does not show a significant difference, indicating that more data is necessary to confirm whether physical objects improve fun and interest in programming.

In Construction of Programs, assembling graphical objects may improve attitudes other than usefulness. Combining selecting/format filling and typing

code may prevent increase in difficulty and a reduction in usefulness. In addition, assembling physical objects improves fun and interest.

Game elements improve usefulness, interest, and fun for programming. The Wilcoxon signed-rank test, which was conducted with the attitude questionnaire results in an environment where the game elements are given as a population, shows significant differences in fun and interest. Moreover, combining game elements with elements of physical objects may affect the difficulty level and usefulness.

Supporting the results of Chapter 3, elements related to problem solving and neutralization abilities are expressions of codes and construction of programs. Characteristics such as liveliness and generated examples in Support may also be influential because these factors confirm the execution result of a program by the motion of a picture. This leads to an understanding of programming. Therefore, such characteristics may lead to abstraction and problem solving. However, the results may depend on the tool. Consequently, teacher's teaching methods and teaching materials may also be involved.

Finally, the learning effects derived from each characteristic are summarized below:

- Programming Constructs
 - Conditional
 - ✧ It is suitable to learn conditional.
 - Loops
 - ✧ It is suitable to learn loops.
 - Recursion
 - ✧ It is suitable to learn loops.
- Representation of Code
 - Text
 - ✧ It alleviates difficulties in programming.
 - ✧ It may improve fun and interest of programming.
 - Pictures

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF PROGRAMMING LEARNING ENVIRONMENTS

- ◇ It improves interest in programming.
- ◇ It improves fun and willingness of programming.
- ◇ It may reduce difficulties in programming and improve usefulness.
- Physical objects
 - ◇ It may improve fun and interest of programming.
- Construction of Programs
 - Typing code
 - ◇ It may improve fun and interest of programming.
 - ◇ It may have an effect when combined with selecting/form filling.
 - Assembling graphical objects
 - ◇ It may improve fun, difficulty, usefulness, willingness, and interest of programming
 - Selecting / form filling
 - ◇ It may improve fun and interest of programming.
 - ◇ It may have an effect when combined with typing code.
 - Assembling physical objects
 - ◇ It may improve fun and interest of programming.
- Game Elements
 - Rule/Restriction
 - Goal
 - Rewards
 - ◇ It improves fun and interest of programming.
 - ◇ It may improve usefulness of programming.
 - ◇ It is effective to use game elements in combination with other game elements and others characteristics.

4.6.3.1. Scratch

Scratch tended to improve the rate of correct responses in the learning comprehension test. In the free description test, many learners described the pattern as U-shaped. Additionally, after the workshop, the perception of the difficulty of programming was remarkably reduced.

In this method, the programming method involves dragging and dropping a block. Hence, action is validated immediately after execution. This method is considered to contribute to the reduction of "difficulty," as assembling graphical objects is a major element of this environment. Furthermore, impressions of "making things" and "making apps" are observed. Accordingly, learners can quickly visualize movement using illustrations. Furthermore, the high degree of freedom in this style of programming seems to contribute to such impressions.

4.6.3.2. Viscuit

This environment tended to improve the rate of correct responses in the learning comprehension test. Both U-shaped and spiral responses were provided in the free descriptions. It is possible that this environment stimulates creativity. The spiral shape can be described simply, using only a few procedures and components. Hence, the ability to abstract problems improved after the workshop.

Common learner's impressions included "moving a picture" and "glasses," possibly because movements with "eyeglasses" are intuitive.

4.6.3.3. CodeMonkey

This environment tended to improve the correct answer rate of the learning comprehension test. In the free description test, many learners described the pattern as U-shaped. In addition, many of the learners tried to explain programs in the free description, indicating that they had thought about and then solved the problem independently. Thus, this environment improved explanation skills.

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF PROGRAMMING LEARNING ENVIRONMENTS

In addition, there was also trend toward improvement in attitudes toward programming. Interestingly, the perceived difficulty of programming did not change after the workshop, possibly because the programming method was easy, combining keyboard input and form selection.

One learner commented, "There were various programs, and I learned something very interesting". This environment contains a collection of problems, allowing the learner to progress continuously without a large gap in difficulty level. This environment seemed to lead to continuous enthusiasm and fun, and it was easy to express the goals and rules of the game elements.

4.6.3.4. Lightbot

This environment tended to improve the rate of correct responses in the learning comprehension test because it helped the learner comprehend different programming concepts. In the free description test, many learners described the pattern as U-shaped. This environment is a simple puzzle game, which can be operated intuitively using a tablet (or smart phone). The learner sees the program that he or she creates as the movements of a robot, promoting the understanding of programming concepts.

One learner commented that it is "easier to learn with the feeling of a game." This "game sensation" improves the learners' motivation and promotes their understanding of programming.

4.6.3.5. OSMO Coding

This environment tended to improve the rate of correct responses in the learning comprehension test. In the free description test, many learners described the pattern as U-shaped. Although major features are not found for specific matters, each subject is approached in a balanced manner. Because the environment is a tangible device, it is considered to be effective for continuous learning without decrease in motivation. However, due to the relationship between the physical block and the software element, the workload may increase, causing learners to quit.

In addition, learners' impressions often included the word "move," e.g., "move the computer" or "move it as instructed", which may be related to assembling and programming the blocks.

4.6.3.6. Robot Turtles

This environment tended to improve the rate of correct responses in the learning comprehension test. In the free description test, many learners described the pattern as U-shaped. The environment is unplugged, and learners can work in groups. Group learning can increase the diversity of knowledge and promote comprehension by enabling students to share the programs they create. Cooperation with others also invokes a game element. Impressions suggested that learners believed that programming could be optimized, as noted in responses such as a "faster way to go forward."

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF
PROGRAMMING LEARNING ENVIRONMENTS

Table 4-4. Feature Table of the Environments

	Programming constructs						Attitude toward programming			
	Sequen- tial	Lo- op	Condi- tional 1	Condit- ional 2	Free Descri- ption (line)	Free Descri- ption (Descri- ption)	Fun	Diffi- culty	Usefu- lness	Willin- gness
Sc			x	x	x			xx		
Vi					x	xx		x		
C M				x	x	x				
Li		x	x	xx	xx					
OC										
RT		x	x							

x = Characteristic and feature, xx = Strong Characteristic and feature

Table 4-5. Analysis of test results

		Sequential		Loops		Conditional	
		Ac	p	Ac	p	Ac	p
Programming constructs	conditional loops	-0.105	0.134	0.368	0.029	0.579	0.00014
	recursion	-0.143	0.317	1	0.157	0.857	0.034

Table 4-6. Analysis of attitude questionnaire

		Fun		Difficulty		Usefulness		willingness		Interest	
		Ac	p	Ac	p	Ac	p	Ac	p	Ac	p
Representation of code	text	0.125	0.317	-0.563	0.084	-0.063	0.655	-0.188	0.257	0.063	0.785
	pictures	0.643	0.084	0.571	0.339	0.571	0.268	0.643	0.084	0.643	0.034
	physical objects	0.429	0.109	0.357	0.473	0.000	1.000	-0.357	0.414	0.286	0.257
Construction of programs	typing code	0.125	0.317	0.000	nan	0.000	nan	-0.250	0.157	0.250	0.317
	assembling graphical objects	0.188	0.317	-0.688	0.135	0.063	0.915	0.188	0.408	0.125	0.480
	selecting/form filling	0.125	0.317	0.000	nan	0.000	nan	-0.250	0.157	0.250	0.317
	assembling physical objects	0.429	0.109	0.357	0.473	0.000	1.000	-0.357	0.414	0.286	0.257
Game Elements	Rule/Restriction Goal Rewards	0.500	0.016	0.536	0.080	0.214	0.484	-0.071	0.608	0.429	0.048

QUANTITATIVE EVALUATION OF THE LEARNING EFFECT EVALUATION OF
PROGRAMMING LEARNING ENVIRONMENTS

Table 4-7. Analysis of the learning effects

Category	Characteristics	Sequential	Loops	Conditional	Fun	Difficulty	Usefulness	willingness	Interest
programming constructs	conditional			xxx					
	loops		xxx						
	variables								
	parameters								
	procedures/methods								
	user-defined data types								
	pre and post conditions								
	recursion		x						
Representation of code	text				x	xx			x
	pictures				xx	x	x	xx	xxx
	flow chart								
	animation								
	forms								
	finite state machine								
	physical objects				x				x
Construction of programs	typing code				x				x
	assembling graphical objects	x	x	x	x	x	x	x	
	demonstrating actions								
	selecting/form filling	x						x	
	assembling physical objects	x						x	
Game elements	Rule/Restriction				xxx		x		xxx
	Goal								
	Rewards								
	Cooperation with Others								

x = Average value changes, xx = Significant trend, xxx = Significant difference

4.7.Discussion

In this section, each RQ is discussed.

4.7.1. Answer of RQ4-1

- RQ4-1: Is there a difference in characteristics between programming environments?

Each environment had unique characteristics (e.g., programming method and expression of programming language), confirming RQ4-1. Table 4-1 shows the qualitative characteristics of the programming environments. As noted in Chapter 2 and in Kelleher et al. [11], some environments share common characteristics. For example, visual programming environments employ a programming method using a drag-and-drop feature. Similarly, in some programming languages, a physical object can be touched by hand. Game software shares common elements (i.e., game elements). The attributes of each environment can be classified from the classification results.

The learning effect depends on the characteristics of the environment. The 43 environments were divided into six categories based on their attributes (Table 4-4). This analysis confirms that all programming learning environments have unique characteristics.

4.7.2. Answer of RQ4-2

- RQ4-2: Does the programming environment influence the learning effect?

Each environment displayed its own learning effect. Due to the small sample size, however, RQ4-2 must be investigated further. In particular, a difference in the learning effect was observed in the free description problem. However, the influence of each environment on the response to the free description problem must be further evaluated. This is obvious from the fact that there were two answers (Figure 4-12).

The questionnaire revealed a difference in attitude regarding the "difficulty" of programming; this is also evident from the results in Figure 4-13 – 4-15. Other attitudes exhibited trends toward improvement. In Chapter 3, it was

demonstrated that visual programming environments improve attitudes toward programming.

4.7.3. Answer of RQ4-3

- RQ4-3: Is there a relation between the characteristics of an environment and the learning effect?

The learning effects of each environment are based on its unique characteristics, confirming RQ4-3. Table 4-1 lists the qualitative characteristics of the programming environment. RQ4-2 reveals that the learning effects depend on the environment. In particular, factors that influence the learning effects include representation of code and construction of programs. Representations of images and texts affect recognition in multimedia research [28][29]. The amount of work (e.g., typing the code) in a programming learning environment impacts the learning effects. The difference in work may influence learners' attitudes toward programming. Juho Hamari and Veikko Eranti reported that game elements impact attitudes toward programming [14].

Each environment also has its own characteristics (Table 4-4). For example, spiral-type answers are found in the free description problem with Viscuit, suggesting that Viscuit helps cultivate abstraction skills. As shown in Table 4-7, the learning effects are easily obtained by characteristics. Programming construct characteristics affect the outcome of each programming concept (loops and conditions). This suggests that characteristics play an important role in understanding the concept of programming, although this finding is a natural result. Moreover, multimedia research reveals that there are differences in the learning effects in the representation of the code [29][30].

This chapter reveals a difference in expression of three patterns of text, picture, and physical objects, which influence the attitude toward programming. The advantage of text representation is that 'code meaning' can be understood by looking at it. The representation of a picture affects the attitude of programming when the illustration used is more relevant to the program's movement, increasing interest in particular. Chapter 3 reveals that a difference in representation affects the learning effects. In addition, the influence of the

representation is also mentioned in a multimedia study [30]. Hence, it is obvious that the difference in expression impacts attitude.

Construction of Programs also affects attitudes towards programming. Typing code is keyboard input. Hence, if learners do not know how to type, this input may reduce willingness. However, by combining selecting/form filling, it may be possible to prevent the decrease in willingness. Assembling graphical objects involves drag and drop, making it relatively easy to program. Thus, graphical objects have the potential to improve the attitude towards programming. Assembling physical objects may not be effective, depending on the tool. The programming method is the easiest. However, it is possible that the learning effect may decrease because the relation between reality and virtual is weak. Furthermore, it is possible that programming is done without a computer (unplugged).

Furthermore, the characteristics of the representation of code and programming constructs may be closely related. For example, when the representation is text, assembling graphical objects tends to make programming feel easier. Even in the case of pictures, interest may be enhanced by assembling physical objects.

As described in previous studies, game elements improve fun and interest [22][23]. It is obvious that these characteristics impact the learning effects. Therefore, the characteristics of each environment may be related to the learning effects.

These unique features may enhance the learning effects according to the intended purpose.

4.8.Related Works

Kelleher et al. [11] qualitatively investigated and categorized multiple programming environments. However, to assess the characteristics and learning effects of these environments, a quantitative investigation is necessary. This research focused on quantitative evaluation with the goal of clarifying the learning effect of environmental characteristics.

Paul Gross and Kris Powers [18] summarized evaluations of programming environments for beginners. Furthermore, they created a rubric to ascertain the

quality of their evaluations, and assessed courses using several different environments. By contrast, our research analyzed the environments themselves and investigated the learning effects of environmental characteristics. By combining their contributions with ours, it may be possible to realize a more systematic evaluation.

4.9.Limitations

I noted the following limitations:

- 1) The population size is small and the number of participants in each environment is biased.
- 2) Some of the test problems were similar to those within the environments.
- 3) It is possible that the learning effects of environmental characteristics depended on the instructor's teaching method.

The bias in the number of learners weakens the statistical validity of this research. To address this problem, we need to accumulate additional data and analyze the data further. The purpose of this research was to investigate environmental characteristics. However, it is possible that the learning effect in each environment depended on the lecturer in charge of the workshop. To solve this problem, the workshop design must be generalized. In future initiatives, we will design a more general workshop.

4.10. Conclusion

I conducted a quantitative evaluation of six programming learning environments, using a workshop approach. The elements of classification influenced the learning effect. All environments improved the result of a learning comprehension test. However, when the software involved physical elements, learners could become bored as the workload increases. Students in three groups (visual programming language, game software, and physical environment) exhibited differences in attitudes toward programming. The use of a visual programming language tended to decrease the perceived difficulty of programming. Although environments with game elements tended to make

CHAPTER 4

programming more fun, they also increased the perceived difficulty of programming.

In the future, I plan to increase both the number of environments and the number of learners. I also plan to design a workshop that is independent of the lecturer and the setting where learning takes place.

CHAPTER 5

CONCLUSION

5.1. Summary

This research investigates the characteristics of the programming learning environment in an effort to determine the learning effects based on characteristics. In Chapter 2, I created a taxonomy table for programming learning environments. This table can classify the programming learning environment, confirming that each environment has unique characteristics.

In Chapter 3, I focused on two different methods in the same environment. Specifically, I examined the learning effects for text-input and visual-input (Representation of Code and Construction of Programs) methods. The method influences not only the attitude towards programming, but also the understanding of programming, demonstrating that the programming method influences the learning effects.

In Chapter 4, I classified the characteristics of six environments. These environments are divided into three categories: visual language, game software, and physical tools (unplugged and tangible device). Furthermore, I examined the learning effects of each category. Similar to the environment, each category influences the learning effects. In particular, characteristics such as Representation of Code (text, image, or physical), Construction of Programs (typing or drag and drop), and Game elements lead to large differences in the learning effects.

Chapter 2 categorizes various environments by characteristics. Chapters 3 and 4 investigate the relationship between the characteristics and the learning effects quantitatively. Moreover, the results show that grasping the characteristics of each environment may maximize the learning effects. The results in Table 2-3 and Table 4-7 assist novice learners in choosing a proper environment. For example, since the representation of the image improves the attitude toward programming on the whole, it is excellent for learning at the very beginning. Assembling graphical objects is a feature seen in visual languages. This environment is excellent for cultivating creativity because it makes programming easy. The environment with game elements makes

programming more interesting. In addition, it is most suitable for learners who wish to acquire logical thinking and problem solving skills. Hence, RQ 1-1 is affirmatively answered. I am convinced that this research will greatly benefit programming learning.

5.2.Future research

Figure 5-1 overviews my future research. The three main areas are to propose and create a programming learning environment, optimize the characteristics and functions of the taxonomy table, and create guidelines to select the appropriate programming learning environment. These future activities are not intended to provide a list of “good” and “bad” environments nor are they designed to simply compare different environments. Instead, they are designed to highlight the merits and demerits of different environments, allowing learners and educators to select the environment to maximize the learning effects based on the learning objective.

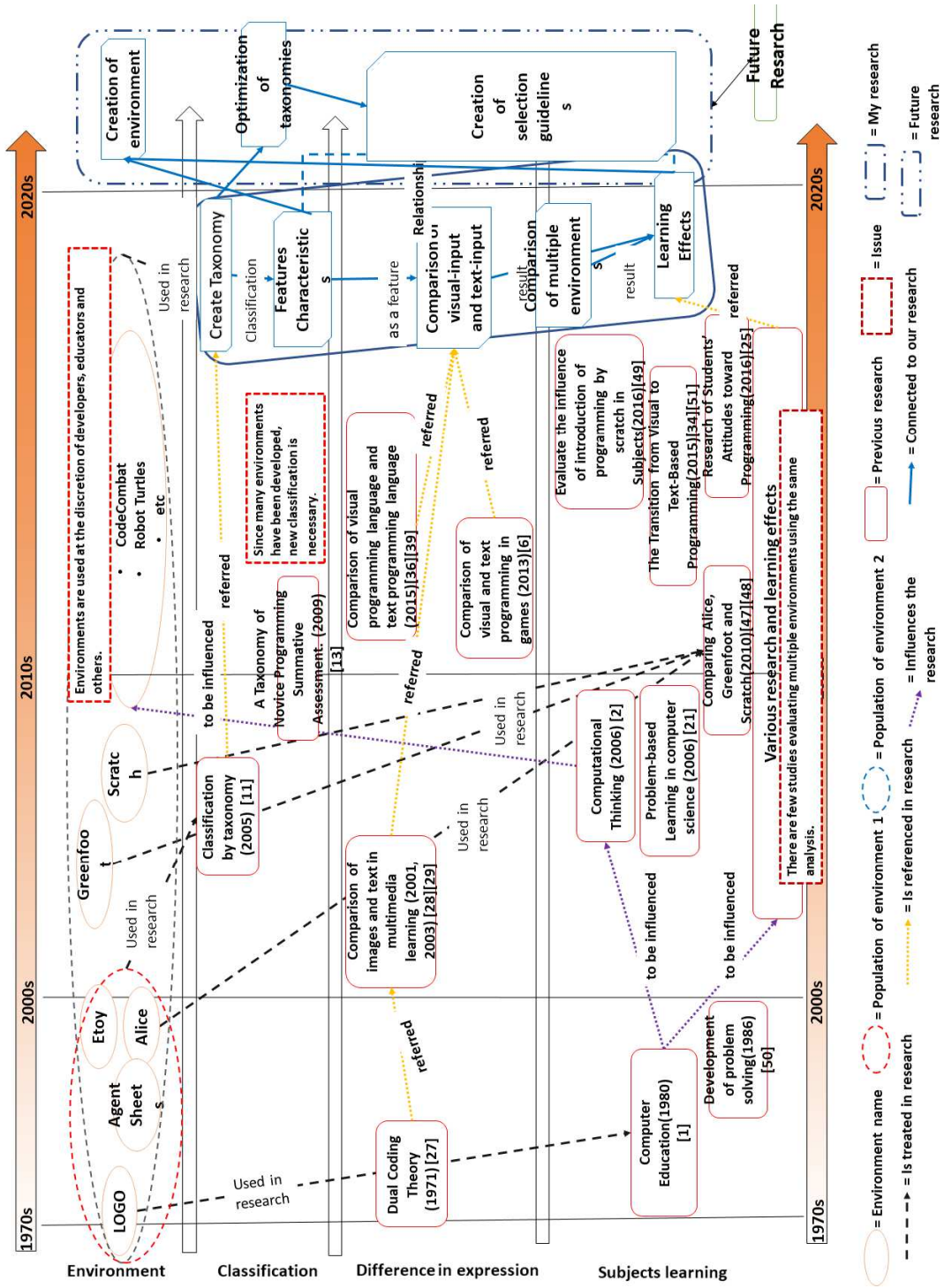


Figure 5-1. Future Research

5.2.1. Propose and Create a Programming Learning Environment

As a future task, I propose an environment to predict learning effects from characteristics.

I would like to obtain the following learning effects:

- Continue fun and interest in programming
- Promote the understanding of programming concepts

According to Chapters 3 and 4, the following characteristics may influence the learning effects:

- Rule/Restriction and Reword (Game Elements): It is possible to promote the understanding of programming concepts while enhancing learners' enjoyment of programming.
- Typing Code and Selecting/form filling (Construction of Programs): It can reduce the input procedure more than assembling graphical objects. Depending on the learner, it may even reduce the programming difficulty.
- Text and Picture (Representation of Code): Combining text representation and image representation may promote understanding of programming.

I propose expanding an existing environment. In this thesis, I used Code Connection [62] (CC) of the programming environment in Minecraft Education Edition [10] (MEE). In MEE, it is possible to add to an existing programming environment called CC. In CC, MakeCode [58], Scratch, and Tynker [59] can be used for programming. An example of MakeCode programming is shown in the Figure 5-2. Either a visual programming language or JavaScript can be used to program in MakeCode.

As an implementation method, I tried to combine Python in a programming language with CC. Python is well utilized in programming learning [45][46][52][64][65]. I created a prototype of the proposed environment with a simple Python library and Web application using the API of publicly available CC [63] (Fig. 5-4). However, image representations cannot be implemented in this prototype. Figure 5-5 shows the basic specifications of this environment.

This prototype can control Minecraft using a simpler code by eliminating the complexity seen in MakeCode (JavaScript). For example, Fig. 5-5 compares programs that stack blocks on MEE. The prototype environment provides easier-to-understand instructions and a library of Python that works on a PC when Python is installed. In this case, the language for image expression cannot be used, but it is easy to shift to full-fledged programming.

In this research, I tried to develop an environment that considers the learning effects. The proposed environment is an extension based on an existing environment. In this work, only partial environments or prototypes are implemented. Currently, I am working on expanding the function of this environment. Moreover, I plan to investigate whether the anticipated learning benefits are obtained based on the characteristics of the prototype environment using a workshop.

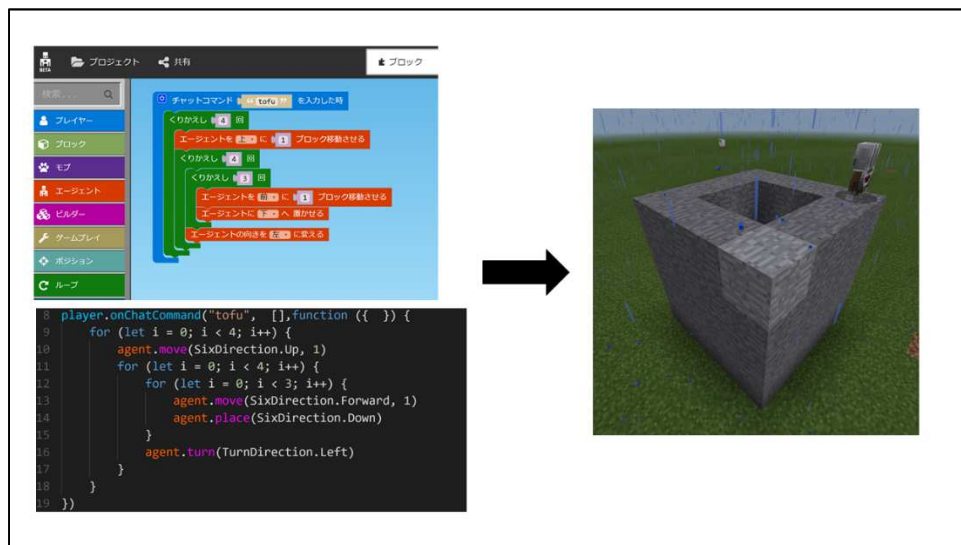


Figure 5-2. MakeCode [58] for MEE

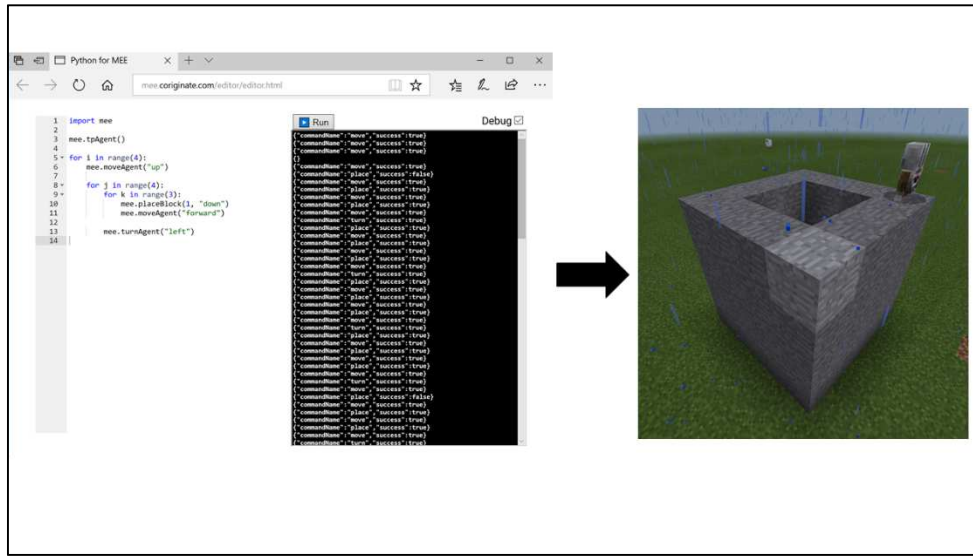


Figure 5-3. Python Environments for MEE

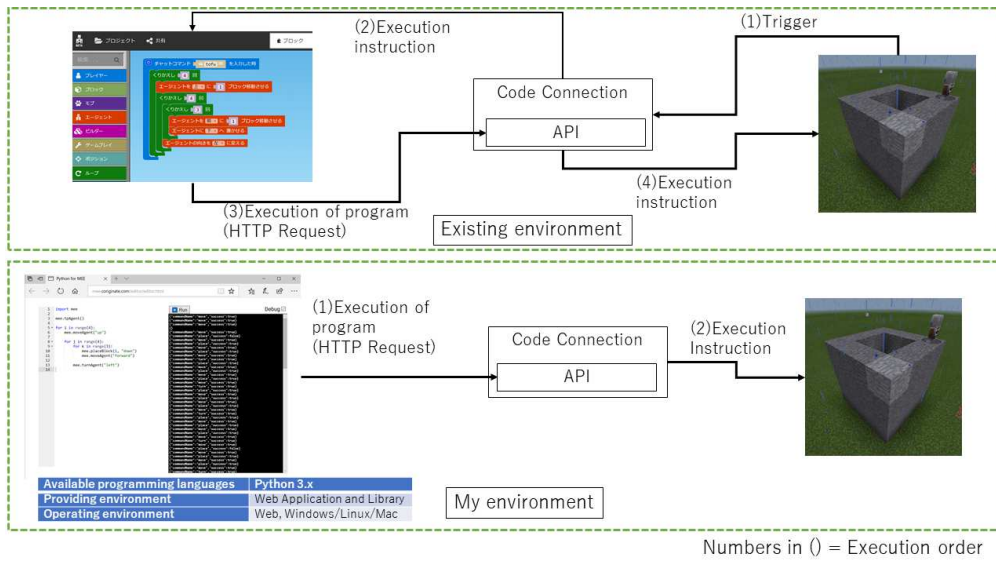


Figure 5-4. Basic specifications

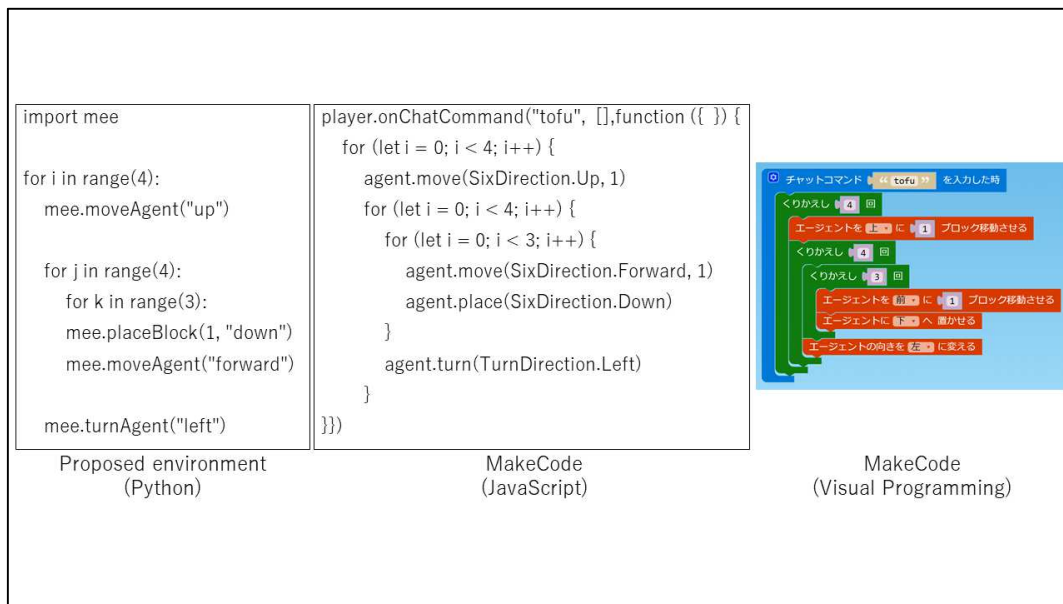


Figure 5-5. Comparison of source code

5.2.2. Other future research

In the future, I plan to increase the survey environment, as well as optimize the characteristics and functions of the taxonomy table.

Then, I will create guidelines to select the appropriate programming environment based on the learning effect. These guidelines will associate attributes and learning effects (Fig. 5-6). In addition, this research demonstrates the usefulness of guidelines and provides developers with guidance in the creation of programming learning environments. These endeavors will not only help learners and educators select a more appropriate environment, but will also facilitate the design of programming learning.

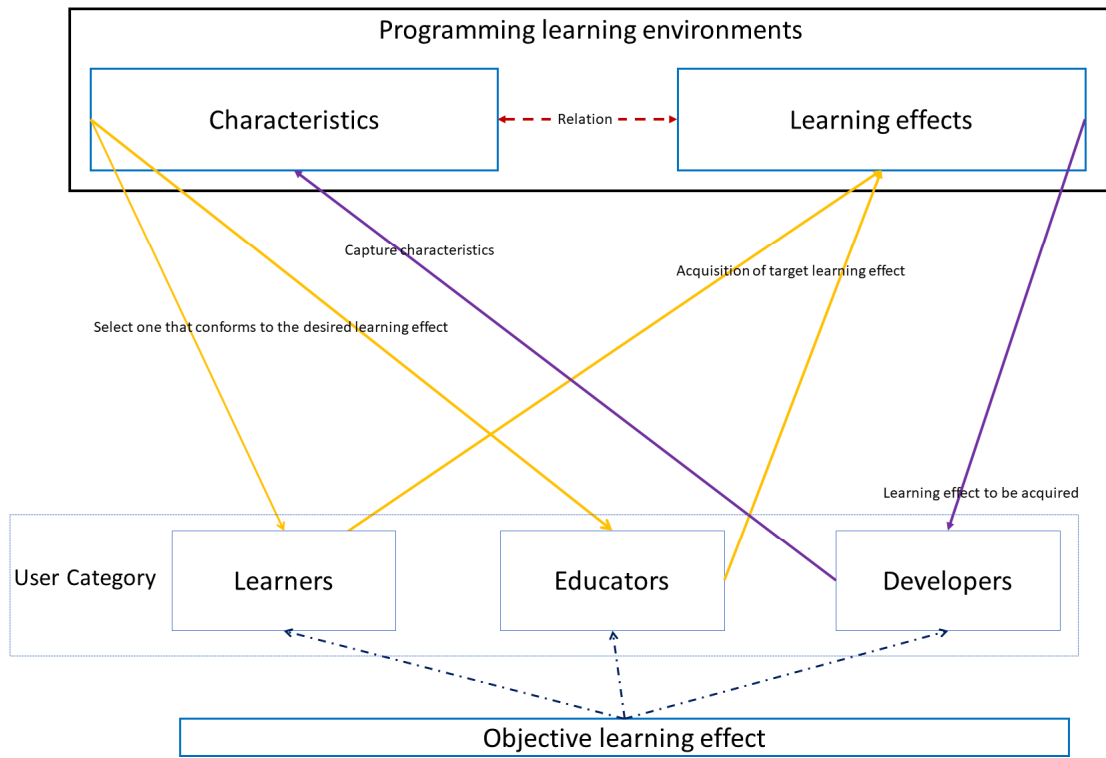


Figure 5-6. About Guideline

REFERENCES

- [1] Papert Seymour, *Mindstorms: Children, computers, and powerful ideas*, Basic Books, Inc., 1980.
- [2] Jeannette M. Wing, *Computational Thinking*, *Communications of the ACM*, Vol.49(3), ACM, 2006, pp. 33-35.
- [3] John Maloney; Mitchel Resnick; Natalie Rusk; Brian Silverman and Evelyn Eastmond, *The Scratch Programming Language and Environment*, *Transactions on Computing Education*, Vol.10(4), ACM, 2010, Article No. 16.
- [4] David J. Malan and Henry H. Leitner, *Scratch for Budding Computer Scientists*, *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, ACM, 2007, pp. 223-227.
- [5] Chiung-Fang Chiu, *Introducing Scratch as the Fundamental to Study App Inventor Programming*, *Learning and Teaching in Computing and Engineering (LaTiCE)*, 2015 International Conference on, IEEE, 2015, pp. 219-220.
- [6] Christopher Zorn; Chadwick Wingrave; Emiko Charbonneau and Joseph J. Laviola, *Exploring Minecraft as a Conduit for Increasing Interest in Programming*, *The 8th International Conference on the Foundations of Digital Games*, *Foundations of Digital Games*, 2013, pp. 352-359.
- [7] Brett Wilkinson; Neville Williams and Patrick Armstrong, *Improving Student Understanding, Application and Synthesis of Computer Programming Concepts with Minecraft*, *The European Conference on Technology in the Classroom*, IAFOR, 2013
- [8] Mitchel Resnick; John Maloney; Andrés Monroy-Hernández; Natalie Rusk; Evelyn Eastmond; Karen Brennan; Amon Millner; Eric

Rosenbaum; Jay Silver; Brian Silverman and Yasmin Kafai, Scratch: Programming for All, Communications of the ACM, Vol. 52(11), ACM, 2009, pp. 60-67.

- [9] CodeCombat Inc, CodeCombat, Retrieved from <https://codecombat.com/>, Accessed on June, 2016.
- [10] Microsoft, Minecraft Education Edition, Retrieved from <https://education.minecraft.net/>, Accessed on June, 2016.
- [11] Caitlin Kelleher and Randy Pausch, Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers, ACM Computing Surveys, Vol. 37(2), ACM, 2005, pp. 83-137.
- [12] Kai Petersen; Robert Feldt; Shahid Mujtaba and Michael Mattsson, Systematic Mapping Studies in Software Engineering. Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, ACM, 2008, pp. 68-77.
- [13] Shuhaida Shuhidan; Margaret Hamilton and Daryl D'Souza, A Taxonomic Study of Novice Programming Summative Assessment, Proceedings of the Eleventh Australasian Conference on Computing Education, Vol.95, ACM.2009, pp. 147-156.
- [14] Daniel Yaroslavski, Lightbot, Retrieved from <https://lightbot.com/>, Accessed on June, 2016.
- [15] Katie Seaborn and Deborah I. Fels, Gamification in Theory and Action: A survey, International Journal of Human-Computer Studies, Vol.74, Elsevier, 2015, pp. 14-31.
- [16] Juho Hamari and Veikko Eranti, Framework for Designing and Evaluating Game Achievements, Proceedings of the 2011 DiGRA International Conference: Think Design Play, DiGRA/Utrecht School of the Arts, 2011.

- [17] Jesper Juul, *Half-real: Video Games Between Real Rules and Fictional Worlds*. The MIT Press, 2011.
- [18] Paul Gross and Kris Powers, *Evaluating Assessments of Novice Programming Environments*, Proceedings of the First International Workshop on Computing Education Research, ACM, 2005, pp. 99-110.
- [19] José-Manuel Sáez-López; Marcos Román-González and Esteban Vázquez-Cano, *Visual Programming Languages Integrated Across the Curriculum in Elementary School: A Two Year Case Study Using "Scratch" in Five Schools*, *Computers & Education*, Vol.97, Elsevier, 2016, pp. 129-141.
- [20] Daisuke Saito and Tsuneo Yamaura, *A New Approach to Programming Language Education for Beginners with Top-Down Learning*, *International Journal of Engineering Pedagogy*, Vol.3(S4), IGIP, 2013, pp. 16-21.
- [21] Jackie O'Kelly and J. Paul Gibson, *RoboCode & Problem-based Learning: A Non-prescriptive Approach to Teaching Programming*, Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ACM, 2006, pp. 217-221
- [22] Ju Long, *Just for Fun: Using Programming Games in Software Programming Training and Education -A Field Study of IBM RoboCode Community*, *Journal of Information Technology Education*, Vol.6, Informing Science Institute, 2007, pp. 279-290.
- [23] Hewijin Christine Jiau; Jinghong Cox Chen and Kuo-Feng Ssu, *Enhancing Self-motivation in Learning Programming Using Game-based Simulation and Metrics*, *IEEE Transactions on Education*, Vol.52(4), IEEE, 2009, pp. 555-562.
- [24] Vasilateanu Andrei; Pavaloiu Bujor Ionel and Wyracic Sebastian, *A Science Fiction Serious Game for Learning Programming Languages*, Proceedings of the 12th International Scientific

Conference "eLearning and Software for Education" , Vol.1, "Carol I" National Defence University, 2016, pp. 561-564.

- [25] Jie Du; Hayden Wimmer and Roy Rada, "Hour of Code": Can It Change Students' Attitudes Toward Programming?, *Journal of Information Technology Education: Innovations in Practice*, Vol.15, Informing Science Institute, 2016, pp. 53-73.
- [26] Allan Paivio, *Imagery and verbal processes*, Psychology Press, 2013.
- [27] James M. Clark and Allan Paivio, *Dual Coding Theory and Education*, *Educational Psychology Review*, Vol.3(3), Springer, 1991, pp. 149-210.
- [28] L ChanLin, *Formats and Prior Knowledge on Learning in a Computer-based Lesson*, *Journal of Computer Assisted Learning*, Vol.17(4), Wiley, 2001, pp. 409-419.
- [29] Richard E. Mayer, *The Promise of Multimedia Learning: Using the Same Instructional Design Methods Across Different Media*, *Learning and instruction*, Vol.13(2), Elsevier, 2003, pp. 125-139.
- [30] Alexander Eitel and Katharina Scheiter, *Picture or Text First? Explaining Sequence Effects when Learning with Pictures and Text*, *Educational Psychology Review*, Vol.27(1), Springer, 2015, pp. 153-180.
- [31] Wanda P. Dann and Randy Pausch, *Learning to Program with Alice (w/CD ROM)*, Prentice Hall Press, 2011.
- [32] Michael Kölling; Neil C. C. Brown and Amjad Altadmri, *Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming*, *Proceedings of the Workshop in Primary and Secondary Computing Education*, ACM, 2015, pp. 29-38.
- [33] Essi Lahtinen; Kirsti Ala-Mutka and Hannu-Matti Järvinen, *A Study of the Difficulties of Novice Programmers*, *Proceedings of the*

10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ACM, 2005, pp. 14-18.

- [34] David Bau; Anthony Bau; Matthew Dawson and C. Sydney Pickens, Pencil Code: Block Code for a Text World, Proceedings of the 14th International Conference on Interaction Design and Children, ACM, 2015, pp. 445-448.
- [35] David Weintrop and Uri Wilensky, To Block or Not to Block, That is the Question: Students' Perceptions of Blocks-based Programming, Proceedings of the 14th International Conference on Interaction Design and Children, ACM, 2015, pp. 199-208.
- [36] David Weintrop and Uri Wilensky, Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs, Proceedings of the Eleventh Annual International Conference on International Computing Education Research, ACM, 2015, pp. 101-110.
- [37] Jessica D. Bayliss, Teaching Game AI Through Minecraft Mods, Games Innovation Conference (IGIC), 2012 IEEE International, IEEE, 2012,
- [38] Colin Gallagher, An Educator's Guide to Using Minecraft in the Classroom: Ideas, Inspiration, and Student Projects for Teachers, Peachpit Press, 2014,
- [39] Thomas W. Price and Tiffany Barnes, Comparing Textual and Block Interfaces in a Novice Programming Environment, Proceedings of the Eleventh Annual International Conference on International Computing Education Research, ACM, 2015, pp. 91-99.
- [40] Ana M Pinto-Llorente; Sonia Casillas Martín; Marcos Cabezas González and Francisco José García-Peñalvo, Developing Computational Thinking via the Visual Programming Tool: Lego Education WeDo, Proceedings of the Fourth International Conference

on Technological Ecosystems for Enhancing Multiculturality, ACM, 2016, pp. 45-50.

- [41] Digital-Pocket, Viscuit, Retrieved from <http://www.viscuit.com/>, Accessed on June, 2016.
- [42] J21 Corporation. CodeMonkey, Retrieved from <https://codemonkey.jp/>, Accessed on June, 2016.
- [43] Tangible Play, OSMO Coding, Retrieved from <https://www.playosmo.com/ja/coding/>, Accessed on June, 2016.
- [44] Thinkfun, Robot Turtles, Retrieved from <http://www.thinkfun.com/products/robot-turtles/>, Accessed on June, 2016.
- [45] Austin Cory Bart; Javier Tibau; Eli Tilevich; Clifford A. Shaffer and Dennis Kafura, BlockPy: An Open Access Data-Science Environment for Introductory Programmers, *Computer*, Vol.50(5), IEEE, 2017, pp. 18-26.
- [46] Linda Grandell; Mia Peltomäki; Ralph-Johan Back and Tapio Salakoski, Why Complicate Things?: Introducing Programming in High School Using Python, *Proceedings of the 8th Australasian Conference on Computing Education*, Vol.52, ACM, 2006, pp. 71-80.
- [47] Ian Utting; Stephen Cooper; Michael Kölling; John Maloney and Mitchel Resnick, Alice, Greenfoot, and Scratch -- A Discussion, *ACM Transactions on Computing Education*, Vol.10(4), ACM, 2010, Article No. 17.
- [48] Sally Fincher; Stephen Cooper; Michael Kölling and John Maloney, Comparing Alice, Greenfoot & Scratch, *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*, ACM, 2010, pp. 192-193.
- [49] Jesús Moreno León; Gregorio Robles and Marcos Román-González, Code to Learn: Where Does It Belong in the K-12 Curriculum?,

Journal of Information Technology Education: Research, Vol.15, Informing Science Institute, 2016, pp. 283-303.

- [50] Noreen M. Webb; Philip Ender and Scott Lewis, Problem-solving Strategies and Group Processes in Small Groups Learning Computer Programming, American Educational Research Journal, Vol.23(2), SAGE, 1986, pp. 243-261.
- [51] Yoshiaki Matsuzawa; Takashi Ohata; Manabu Sugiura and Sanshiro Sakai, Language Migration in Non-cs Introductory Programming Through Mutual Language Translation Environment, Proceedings of the 46th ACM Technical Symposium on Computer Science Education, ACM, 2015, pp. 185-190.
- [52] Atanas Radenski, "Python first": A Lab-based Digital Introduction to Computer Science, Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ACM, 2008, pp. 197-201.
- [53] Dylan J. Portelance; Amanda L. Strawhacker and Marina Umaschi Bers, Constructing the ScratchJr Programming Language in the Early Childhood Classroom, International Journal of Technology and Design Education, Vol.26(4), Springer, 2016, pp. 489–504.
- [54] Ryan Andrew Nivens and Rosemary Geiken, Using a Computer Science-based Board Game to Develop Preschoolers' Mathematics, 13th International Congress on Mathematical Education, Society of Didactics of Mathematics, 2016, Poster presentation.
- [55] Tihomir Orehovački and Snježana Babić, Inspecting Quality of Games Designed for Learning Programming, International Conference on Learning and Collaboration Technologies, Springer, 2015, pp. 620-631.
- [56] Lindsey Ann Gouws; Karen Bradshaw and Peter Wentworth, Computational Thinking in Educational Activities: An Evaluation of the Educational Game Light-bot, Proceedings of the 18th ACM

Conference on Innovation and Technology in Computer Science Education, ACM, 2013, pp. 10-15.

- [57] Susan H. Rodger; Maggie Bashford; Lana Dyck; Jenna Hayes; Liz Liang; Deborah Nelson and Henry Qin, Enhancing K-12 Education with Alice Programming Adventures, Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education, ACM, 2010, pp. 234-238.
- [58] Microsoft, MakeCode for Minecraft, Retrieved from <https://minecraft.makecode.com/>, Accessed on September, 2017.
- [59] Tynker, Tynker, Retrieved from <https://www.tynker.com/>, Accessed on September, 2017.
- [60] Daniel Yaroslavski, How Does Lightbot Teach Programming?, Retrieved From http://lightbot.com/Lightbot_HowDoesLightbotTeachProgramming.pdf, Accessed on June, 2016.
- [61] Alexander Repenning; David C. Webb; Catharine Brand; Fred Gluck; Ryan Grover; Susan Miller; Hilarie Nickerson and Muyang Song, Beyond Minecraft: Facilitating Computational Thinking Through Modeling and Programming in 3d, IEEE Computer Graphics and Applications, Vol.34(3), IEEE, pp. 68-71.
- [62] Microsoft, Code Connection, Retrieved from <https://education.minecraft.net/support/knowledge-base/connecting-code-connection-minecraft/>, Accessed on September, 2017.
- [63] Microsoft, Code Connection: API Documentation, Retrieved from <http://aka.ms/mee-ccapi>, Accessed on September, 2017.
- [64] Craig Richardson, Learn to Program with Minecraft: Transform Your World with the Power of Python, No Starch Press, 2015
- [65] Jason Briggs, Python for Kids: A Playful Introduction to Programming, No Starch Press, 2013.

- [66] Marina Papastergiou, Digital Game-based Learning in High School Computer Science Education: Impact on Educational Effectiveness and Student Motivation, *Computers & Education*, Vol.52(1), ACM, 2009, pp. 1-12.
- [67] Tamotsu Mitamura; Yasuhiro Suzuki and Takahumi Oohori, Serious Games for Learning Programming languages, *Systems, Man, and Cybernetics (SMC)*, 2012 IEEE International Conference on, IEEE, 2012, pp. 1812-1817.
- [68] Cagin Kazimoglu; Mary Kiernan; Liz Bacon and Lachlan Mackinnon, A Serious Game for Developing Computational Thinking and Learning Introductory Computer Programming, *Procedia-Social and Behavioral Sciences*, Vol.47, Elsevier, 2012, pp. 1991-1999.
- [69] University of Kent; oracle, Greenfoot, Retrieved from <https://www.greenfoot.org/door>, Accessed on September, 2017.
- [70] Michael Kölling, The Greenfoot Programming Environment, *ACM Transactions on Computing Education*, Vol,10(4), ACM, 2010, Article No. 14.
- [71] MIT Media Lab, Scratch - Imagine, Program, Share, <https://scratch.mit.edu/>, Accessed on September, 2017.
- [72] Hiroshi Ishii, The Tangible User Interface and Its Cvolution, *Communications of the ACM*, Vol.51(6), ACM, 2008, pp. 32-36.

RESEARCH ACHIEVEMENT

Journals

- Daisuke Saito; Hironori Washizaki and Yoshiaki Fukazawa, Comparison of Text-Based and Visual-Based Programming Input Methods for First-Time Learners, Journal of Information Technology Education: Research, Vol. 16, Informing Science Institute, Jun. 2017, pp. 209-226.
- Daisuke Saito and Tsuneo Yamaura, A New Approach to Programming Language Education for Beginners with Top-Down Learning, International Journal of Engineering Pedagogy, Vol. 3(S4), International Society of Engineering Education, Dec. 2013, pp. 16-21.

International Conferences

- Daisuke Saito; Ayana Sasaki; Hironori Washizaki; Yoshiaki Fukazawa and Yusuke Muto, Quantitative Learning Effect Evaluation of Programming Learning Tools, Teaching, Assessment, and Learning for Engineering (TALE), 2017 IEEE International Conference on. IEEE, Dec. 2017, pp. 209-216, Hongkong, China.
- Daisuke Saito; Ayana Sasaki; Hironori Washizaki; Yoshiaki Fukazawa and Yusuke Muto, Program Learning for Beginners: Survey and Taxonomy of Programming Learning Tools, Engineering Education (ICEED), 2017 IEEE 9th International Conference on. IEEE, Nov. 2017, pp. 137-142, Ishikawa, Japan.
- Daisuke Saito; Hironori Washizaki and Yoshiaki Fukazawa, Analysis of the Learning Effects Between Text-based and Visual-based

Beginner Programming Environments, Engineering Education (ICEED), 2016 IEEE 8th International Conference on, IEEE, Dec. 2016, pp. 208-213, Kuala Lumpur, Malaysia.

- Daisuke Saito; Hironori Washizaki and Yoshiaki Fukazawa, Influence of the Programming Environment on Programming Education, Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, ACM, Jul. 2016, pp. 354-354, Arequipa, Peru.
- Daisuke Saito; Hironori Washizaki and Yoshiaki Fukazawa, Work in progress: A Comparison of Programming Way: Illustration-based Programming and Text-based Programming, Teaching, Assessment, and Learning for Engineering (TALE), 2015 IEEE International Conference on, IEEE, Dec. 2015, pp. 220-223, Zhuhai, China.
- Daisuke Saito; Akira Takebayashi; Tsuneo Yamaura; Hironori Washizaki and Yoshiaki Fukazawa, An Evaluation and Result of a Workshop Using Minecraft for ICT Education. Replaying Japan 2015: 3rd International Japan Game Studies Conference, May. 2015. Kyoto, Japan.
- Daisuke Saito and Tsuneo Yamaura, Applying the Top-down Approach to Beginners in Programming Language Education, Interactive Collaborative Learning (ICL), 2014 International Conference on, IEEE, Dec. 2014, pp. 311-318. Dubai, UAE.
- Daisuke Saito; Akira Takebayashi and Tsuneo Yamaura, Minecraft-based Preparatory Training for Software Development Project, Professional Communication Conference (IPCC), 2014 IEEE International, IEEE, Oct. 2014, pp. 1-9, Pittsburgh, USA.
- Daisuke Saito; Akira Takebayashi; Taiki Nizuma; Renato Nojiri and Tsunao Yamaura, Minecraft-based Communication Learning to Elementary School Students and Junior High School Students,

Replaying Japan 2014: 2nd International Conference on Japanese Game Studies, Aug. 2014, pp. 30, Edmonton, Canada.

- Daisuke Saito and Tsuneo Yamaura, A New Approach to Programming Language Education for Beginners with Top-down Learning, Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on, IEEE, Aug. 2013, pp. 752-755, Bali, Indonesia.

Domestic Conferences

- 佐々木 綾菜; 鷺崎 弘宜; 齋藤 大輔; 深澤 良彰; 武藤 優介; 西澤 利治, 小学校におけるプログラミング教育において活用可能なループリックの提案, 日本デジタル教科書学会第6回年次大会, 日本デジタル教科書学会, 2017年8月, pp. 33-34, 東京都.
- 齋藤 大輔; 佐々木 綾菜; 鷺崎 弘宜; 深澤 良彰; 武藤 優介. 初学者向けプログラミング学習ツールにおけるゲームソフトウェアの調査と分類, 日本デジタルゲーム学会 2016年度 年次大会, 日本デジタルゲーム学会, 2017年3月, pp. 51-54, 愛知県.

Lectures

- 齋藤 大輔; 鷺崎 弘宜, Python を含む複数のプログラミング言語の初学者向け学習環境の特性・特徴の分析, PyCon JP 2017, 2017年9月8日, 東京都.
- 齋藤 大輔, 第四次産業革命としごとの在り方」- こども向けプログラミング学習の観点から-, 2017年度 第1回しごと能力研究学会部会・研究会, しごと能力研究学会, 2017年8月5日, 宮城県.
- Daisuke Saito, Learn to Program with Minecraft: A Comparison of the Effects of Learning with Programming Methods, PCS-J 2nd

Technical Meeting and General Assembly 2015, IEEE Professional Communication Society Japan Chapter, 2015 年 12 月 19 日, 長野県.

Books

- 齋藤大輔, Minecraft で楽しく学べる Python プログラミング, ソーテック社, 2017 年 6 月 10 日, ISBN: 978-4800711656.
- 松尾 高明; 齋藤 大輔; ナポアン; nishi, みんな大好き! マインクラフトるんるんプログラミング! コマンドブロック編, ソシム, 2017 年 3 月 21 日, ISBN: 978-4802610780