修士論文

ビジュアル型とテキスト型プログラムの相互変換を用いたプログラミング学習支援システム

野口　幹太

主指導教員　　長谷川　忍

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和 6 年 3 月

Abstract

Visual programming can be cited as a learning support for programming beginners. Visual programming includes block-based programming, in which programming is done by combining puzzles and blocks, flow-based programming, in which programming is performed by connecting the nodes of each program, and a unique rule type performs programming using a method different from the type. All three have in common that they are visually easy to understand, easy to operate, less likely to cause errors, and allow learners to select from a set of predetermined code fragments.

On the other hand, examples of text-based programming required in an increasingly information-oriented society include Javascript, Python, Java, C#, and PHP. By learning these skills, learners will be able to handle the front-end to back-end processes in system development, and the demand for human resources in the information society will increase. However, while learning text-based programming allows them to utilize data effectively, it requires writing from scratch and error handling and is characterized by high learning costs in terms of time, teaching materials, and environmental construction.

Although Scratch has been successful as a visual programming tool for elementary school students, there is a gap in connecting it to text programming. Although it is not known exactly where the gap is, it is thought that a major reason for the gap is that it is impossible to convert to text type, and it is difficult to visualize the transition from visual type to text type.

Google Blockly, a block-based visual programming language, is an example of a language compatible with visual and text types. Google Blockly allows learners to convert the created visual program into a text program (JavaScript, Python, PHP, Lua, Dart) and run it. This feature makes it easy for the image of text-based programming to become firmly established, but there is an issue in which beginners who have completed visual programming may not be able to confirm the visual representation during text-based programming. To solve this problem, it is necessary to further solidify the connection between the images of visual and text programs by implementing conversion from text programs to visual programs.

In this study, we focused on the problem that it is difficult for learners who have finished visual programming to transfer to text programming, and in order to solve this problem, in addition to converting existing visual programming to text programming, we Implement the conversion function from a type program to a visual type program. By doing so, we will promote the learning of text-based programming while taking advantage of the characteristics of visual programming, which is easy to see and less likely to cause errors.

To achieve this, we will build an environment that makes it easy to compare visual and text-based programming, implement mutual conversion between visual and text-based

programs, and conduct experiments to demonstrate the usefulness of the developed support system.

We mainly used Google Blockly to build the visual programming environment. The reason is that it is compatible with text-based programs and that learners can add original visual-based programs.

MonacoEditor was used as an editor to construct the text-based programming environment. The reason was that it was necessary to display syntax errors in the program to the learner. The programming language in MonacoEditor was JavaScript. The reason is that it is one of the programming languages compatible with Google Blockly that can be easily executed on a browser.

To convert from a visual type to a text type program, we used the functions of Google Blockly. In Google Blockly, each visual program has a built-in text program, and the built-in programs are combined through connections between visual programs, ultimately completing the entire text program. Using this function, we converted a visual program to a text program.

AST(AbstractSyntaxTree)is a declaration statement, a statement (Statement), an expression (Expression), an operator, an identifier (Identifier), a value (Literal), and a block statement (BlockStatament) that is the body of if, for, function, etc.) is the tree structure data of the program. The text-based program was converted to a visual-based program based on this AST information.

We built an environment that displays visual and text environments within the same interface and allows programming. A visual environment is built on the left half, a text environment is built on the right half, and a conversion button is placed between them, making it easy to compare. Additionally, the system runs on a browser and can be executed, converted, saved, and loaded with the click of a button.

Using the developed system, we conducted experiments with graduate students who do not specialize in information systems. The subjects learned using the mutual conversion function in one of the two learning processes. A mastery comprehension test was conducted before and after learning to measure the subjects' level of understanding of text-based programming, and a questionnaire was administered before and after the experiment. We also collected programming learning logs during the experiment and obtained the error rates of program execution and conversion. In addition, we conducted an experiment on the accuracy of conversion from text type to visual type using reference books.

As a result of the mastery level comprehension test, we were able to show that learning using the system is significant in text-based programming learning with a p-value = 0.0009119, but a comparison between with and without mutual conversion shows that the p-value =0.89, the result is not significant. Errors occurred with a probability of

approximately 8% when running text-based programs, resulting in many definition errors. The conversion accuracy was poor for function call statements in text-based programs.

There are still areas for improvement in this system. In order to make it easier to compare visual and text types, the colors of characters and blocks in both types should be unified, and the number of programming languages that the system can support should also be increased. Also, since only certain visual type programs can be converted from text type programs, it is necessary to implement a function that automatically generates visual type programs from text type programs or the AST of text type programs.

目次

図目次

表目次

第 1 章　序論

1.1　研究背景

高等学習指導要領の改訂により 2022 年度からの高校教育の「情報」科では、共通必修科目「情報 I 」と選択科目「情報II」が新設されることになり，すべての生徒がプログラミングやネットワーク、データベースの基礎等について学習することになった[1]．またプログラミング教育必修化に伴い，2024 年度からの大学入学共通テストでは「情報 I 」が出題されることとなった[2]．こういった流れにより，プログラミング学習支援の需要は今後も増加すると予想される．

プログラミング初学者の学習支援としてビジュアル型プログラミングが挙げられる[3,4,5].ビジュアル型プログラミングとは，パズルやブロックを組み合わせることでプログラミングを行うブロック型[3]，各プログラムのノードをつなげてプログラミングを行うフロー型[4]，ブロック型やフロー型とは違う手法を使用してプログラミングを行う独自ルール型[5]がある．これらの 3 つは共通点として視覚的に分かりやすい，操作が簡単，エラーが起きにくい，決められたコードの断片の中から選ぶという特徴がある．

一方で，情報化が進む社会において求められるテキスト型プログラミングの例として Javascript，Python,Java,C#,PHP などが挙げられる[6]．これらを学習することで，システム開発におけるフロントエンドからバックエンドの工程をこなすことができ，情報社会においての人材需要が高まる．しかしながら，テキスト型プログラミングを習得することでデータを上手く活用できるようになる一方で，ゼロからの記述及びエラー処理を必要とし，時間や教材や環境構築に関する学習コストが高いという特徴を持っている．

総務省が教育関係団体(NPO 法人・民間教育事業者・教育関係機関(原則として学校を除く.))学識経験者及び民間企業等に行ったプログラミング人材育成の在り方に関する調査研究報告書[7]によると，ビジュアル型プログラムである Scratch[3]を選択している業者が多く，Scratch は小学生向けのビジュアル型プログラミングとして成功しているが，テキスト型プログラミングへ繋げるにはギャップがあると述べている．具体的にどこにギャップがあるかはこの報告書では述べられていないが，ギャップの大きな原因は Scratch ではテキスト型への

変換が行えず，ビジュアル型からテキスト型へのイメージが湧きにくいためだと考えられる．

ビジュアル型とテキスト型で互換性があるものとしてブロック型のビジュアル型プログラミング言語である Google Blockly[8]が挙げられる．Google Blockly では，作成したビジュアル型プログラムをテキスト型プログラム(JavaScript,Python,PHP,Lua,Dart)に変換し，実行することができる．この特徴によりテキスト型プログラムへのイメージが定着しやすいが，ビジュアル型プログラミングを終えた初学者がテキスト型プログラミング中にビジュアル型での表現を確認したい場合に確認できない課題がある[9]．この課題を解決するためにテキスト型プログラムからビジュアル型プログラムへの変換を実装することで，ビジュアル型とテキスト型プログラムのイメージの結びつきをより凝固にする必要がある．

1.2 研究目的

本研究では, ビジュアル型プログラミングを終えた学習者がテキスト型プログラミングに移行するのが難しいという課題に着目し，この課題を解決するために既存のビジュアル型からテキスト型プログラムの変換に加えて，テキスト型プログラムからビジュアル型プログラムへの変換機能を実装する．それにより，ビジュアル型の視覚的に見やすい，エラーが起きにくいという特徴を活かしつつ，テキスト型プログラミングの学習の促進を行う．

この実現のために①ビジュアル型とテキスト型プログラミングが比較しやすい環境を構築し，②ビジュアル型とテキスト型プログラムの相互変換を実装し，③開発した支援システムの有用性を示す実験を行う．

1.3 本論文の構成

本論文の構成を以下に示す．

第 1 章　序論

本研究の背景，目的について述べる．

第 2 章　関連研究

本研究の関連研究について示す.

第 3 章　提案手法

本研究で用いた手法について述べる.

第 4 章　開発システム

開発したシステムについて述べる.

第 5 章　評価実験

開発したシステムの評価実験について述べる.

第 6 章　終章

本研究のまとめと今後の展望について述べる.

第 2 章　関連研究

本章では，プログラミング学習支援としてビジュアル型とテキスト型プログラムの変換を行っている関連研究について紹介する．

2.1　プログラムの片側変換を利用した学習支援

本節では，ビジュアル型とテキスト型プログラムの片側変換を利用したプログラミング学習支援を行っている関連研究について述べる．

山梨らはビジュアル型プログラムのブロックの構成が不完全な状態からテキスト型プログラムに変換されたプログラムでは構文誤りが発生してしまう問題を解決するために，構文誤りを含んでいるテキスト型プログラムの欠損部分を仮トークンで充足し，ビジュアル型プログラムへの変換を行った[10]．

Google 社が提供する Google Blockly はドラッグ&ドロップブロックを使用するオープンソースのビジュアルプログラミングエディタである[8]. Blockly はビジュアル型プログラムからテキスト型プログラム(JavaScript,Python,PHP,Lua,Dart)に変換でき，両型のプログラムをブラウザ上で実行することができる．

末吉らはビジュアル型プログラミングでプログラミングを学んだ初学者もテキスト型プログラミングを改めて学びなおさなければならないという問題を解決するために，ビジュアル型プログラムからテキスト型プログラムである C 言語に変換できる学習支援システムを Web ブラウザ上で構築した[11]．

本研究では，ビジュアル型とテキスト型の表現の違いを認識させ，テキスト型をビジュアル型へ変換することで視覚的に理解しやすくするためにビジュアル型とテキスト型プログラムの片側変換だけでなく，相互変換を用いることでテキスト型プログラミングの学習支援を行う．

2.2　プログラムの相互変換を利用した学習支援

本節では，ビジュアル型とテキスト型プログラムの相互変換を利用したプログラミング学習支援を行っている関連研究について述べる．

山梨らは,ビジュアル型プログラミング言語はプログラミング初学者にとって有用であるが，テキスト型プログラミングを学べないという課題を解決するた

めに両型の相互変換機能と両型の表現の間に両表現を併用して記述できる記述形式を取り入れたシステムを作成した[9].そのシステムでは，テキスト型からビジュアル型プログラムへの変換において命令列記入時にリアルタイム変換行い，文法エラーを含んでいるプログラムが変換できない仕様になっている．しかし，テキスト型プログラミングのデバッグ処理の支援を行っていない.

松澤らはビジュアル型プログラミングによる入門学習を終えた学習者に対して C や Java などのテキスト記述型言語へのシームレスな移行が考慮されていないという問題を解決するためにビジュアル型言語とテキスト型言語(Java)の併用開発環境「BlockEditor」を構築した[12]．BlockEditor はビジュアル型とテキスト型プログラムを相互変換する機能を有しており，両型での実行も可能となっている．しかし，ビジュアル型とテキスト型プログラミングを行うインターフェイスが別になっており，両型を比較しにくい設計になっている．また，テキスト型からビジュアル型プログラムへの変換において，文法エラーを含んでいるプログラムが変換できないが，テキスト型プログラミングのデバッグ処理の支援を行っていない.

Microsoft 社が提供している MicrosoftMakeCode はビジュアル型プログラムゲームやマイコンを動作させることで学習支援を行っているプラットフォームである[13]．MicrosoftMakeCode ではビジュアル型とテキスト型プログラムの相互変換することができるが，プログラミング画面をビジュアル型かテキスト型のどちらかに変更する必要があり，両型を比較しにくい設計となっている．また，ゲームやマイコンを動かす内部関数を理解していないとテキスト型プログラムでの編集が満足に行うことができない.

Michael らは，ビジュアル型プログラミングからテキスト型プログラミングへの移行支援として，従来のテキスト構文を備えた教育用プログラミング言語である Grace[14]をタイルベースとしたエディタ Tiled Grace を開発した[15].しかし，Tiled Grace はビジュアルビューとテキストビューに切り替えることでビジュアル型とテキスト型のどちらでもコーディングを行うため，インターフェイス内で同時に両型の比較を行うことができない.

本研究では，ビジュアル型とテキスト型プログラミング環境を一つのインターフェイス内で完結することによって，ビジュアル型とテキスト型プログラムの比較が行いやすいシステムを目指す．また，テキスト型プログラミングのデバッグ処理支援として構文検証システムを導入することで，デバッグ処理やビジ

ュアル型プログラムへの変換を行いやすくする．そして，内部関数を使用しない初学者向けのプログラミングを対象とすることで，実行結果も分かりやすく表示する．

第 3 章　提案手法

本章では，本研究で提案するビジュアル型とテキスト型プログラムの相互変換システムについて述べる．

3.1　概要

本研究では，ビジュアル型とテキスト型プログラムの相互変換を用いることで，ビジュアル型とテキスト型プログラムのイメージの結びつきを凝固にし，テキスト型プログラミングの習得を支援するシステムの構築を行う．そのためにビジュアル型とテキスト型プログラミングを比較しやすい環境を構築し，ビジュアル型とテキスト型プログラムの相互変換を実装する必要がある．

このシステムを実現にするために以下のシステムの要件を満たす必要がある．

① 比較しやすい環境：

ビジュアル型とテキスト型プログラムが同一のインターフェイスで入力/実行/保存/読込ができる．

② 相互変換機能：

ビジュアル型とテキスト型プログラムの相互変換機能を実装し，テキスト型からのビジュアル型への変換エラーを防ぐためにテキスト型プログラミングに構文検証部を導入する．

先述したシステムの構成する 2 つの要素を踏まえ，考案したシステムの全体図を図 3.1 に示す．

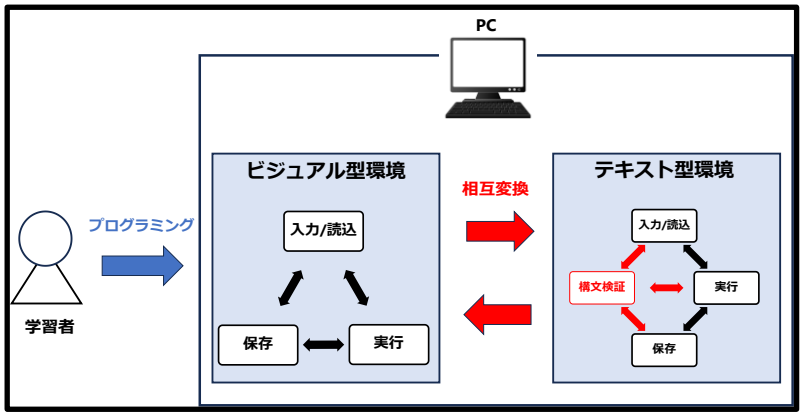

図 3.1：システム全体図

3.2　ビジュアル型環境

ビジュアル型プログラミング環境の構築には，主に Google Blockly[8](図 3.2)を利用した．理由としては，テキスト型プログラムへの互換性があるのとオリジナルのビジュアル型プログラムを追加することができるからである．

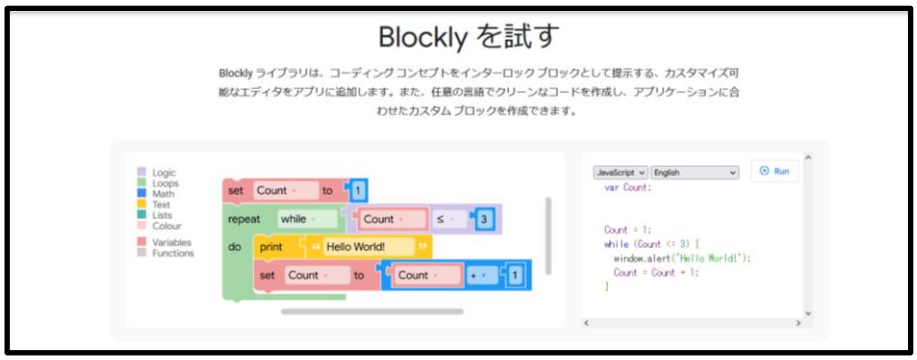

図 3.2：Google Blockly [8]

Google Blockly[8]ではブロックの凹凸及び内部の入力を利用して，プログラミングを行う．組み合わせの種類は左右の凹凸と上下の凹凸の二種類あり，制御構文や関数文といった命令文を内包しているものは内部に上下の凹凸がある．凹凸の例を図 3.3 に示す．

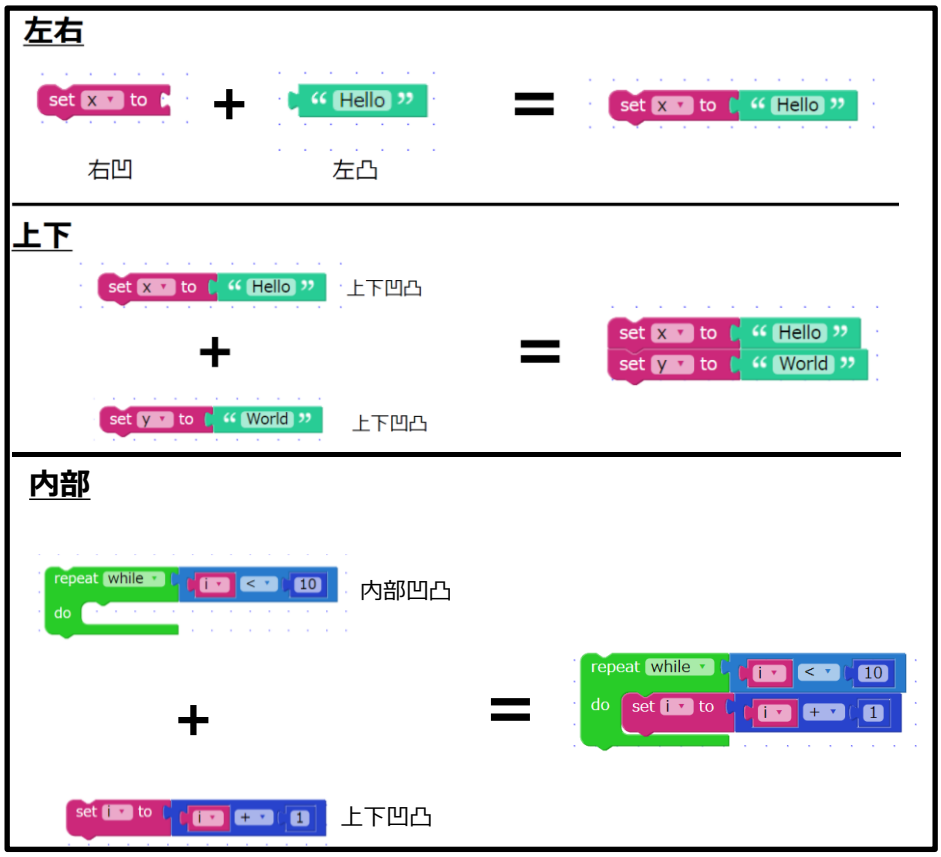

図 3.3：ビジュアル型の凹凸の組み合わせの例

Google Blockly[8]では数値や文字列といったデータの入力はブロック内にてキーボード入力を行う必要があり，両方ともに左凸ブロックとなっている．また，エラーを防ぐために数値ブロックには文字列や全角数字が入力できない仕様になっている．数値と文字列のブロックの例を図 3.4 に示す．

図 3.4：数値と文字列ブロックの例

Google Blockly 内の表示言語は日本語が読みづらかったため，英語にした．

ビジュアル型プログラムの実行に関しては，ビジュアル型プログラムを一度テキスト型プログラムに変換し，テキスト型プログラムを実行する形で実装した．ビジュアル型プログラムの保存/読込に関しては，ビジュアル型プログラムを xml 形式で保存/読込を行う形で実装した．

3.3 テキスト型環境

テキスト型プログラミング環境の構築には，エディタとして MonacoEditor[16]を利用した．理由としては，後述する構文検証部に必要であったためである. MonacoEditor 内のプログラミング言語は. JavaScript とした．理由としては，Google Blockly[8]と互換性がある中でブラウザ上での実行が行いやすい言語だからである．

テキスト型プログラムの実行の関しては，そのままテキスト型プログラムを実行する形で実装した．テキスト型プログラムの保存/読込に関しては，テキスト型プログラムを js 形式で保存/読込を行う形で実装した．

テキスト型プログラムの変換エラーを防ぐため，デバッグ処理を支援するためにテキスト型環境に構文検証部を導入した. 構文検証には, MonacoEditor[16]と JavaScript 構文解析ライブラリ esprima[17]を掛け合わせた Syntax Validator[18]を用いて行う．Syntax Validator(図 3.5)では，構文エラーがあるとエディタ内の該当箇所が赤い波線でマークアップされ，赤いテキストボックスが下部に表示される. 構文エラーがない場合は, 緑のテキストボックスが下部に表示される．これらの機能をテキスト型環境に導入した．

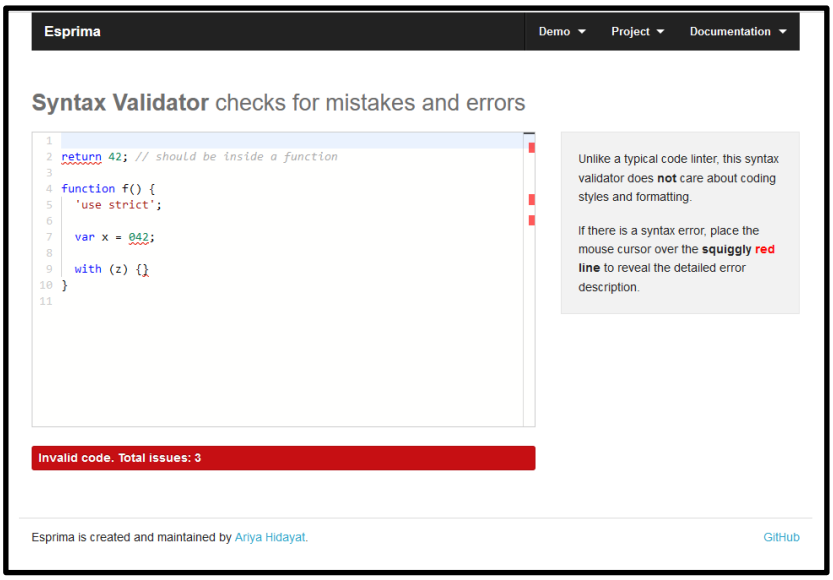

図 3.5：Syntax Validator[18]

3.4　ビジュアル型からテキスト型への変換

ビジュアル型からテキスト型プログラムへの変換には，Google Blockly[8]の機能を用いた．Google Blockly では，各ビジュアル型プログラムにテキスト型プログラムが内蔵されており，ビジュアル型プログラム同士の繋がりによって内蔵プログラムが組み合わさり，最終的に全体のテキスト型プログラムが完成する．この機能を用いて，ビジュアル型プログラムをテキスト型プログラムに変換した．変換例として，while 文単体と while 文に条件と動作を加えたものの変換例を図 3.6 に示す．

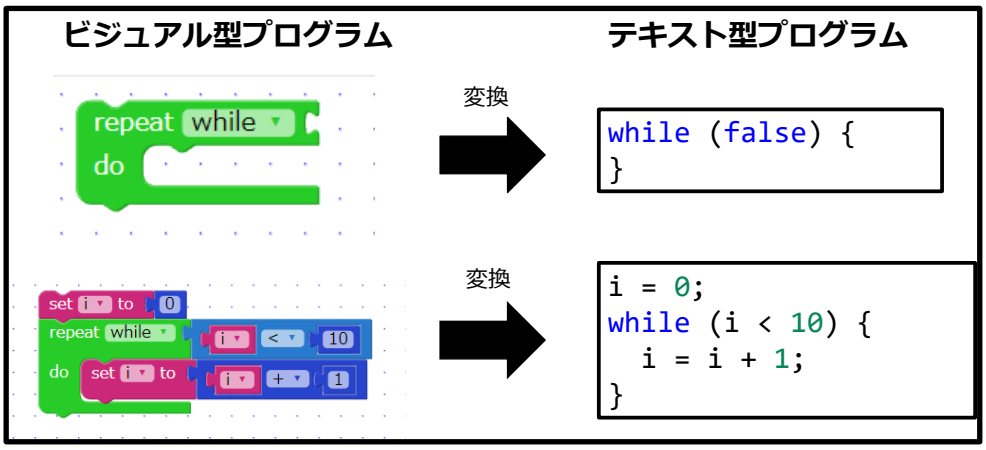

図 3.6：ビジュアル型からテキスト型への変換例

3.5　テキスト型からビジュアル型への変換

3.5.1　変換方法

プログラムは宣言文(Declaration)，命令文(Statement)，式(Expression)，演算子(Operator)，識別子(Identifier)，値(Literal)，および if や for,関数等の本体となるブロック命令文(BlockStatament)から構成される．これらの構成要素を木構造のデータ構造で表したものを AST (AbstractSyntaxTree)と言われる[19].

テキスト型からビジュアル型プログラムへの変換はこの AST を探索し，AST の情報を元にビジュアル型プログラムへ変換する処理を実装した．変換の流れを図 3.7 に示す.

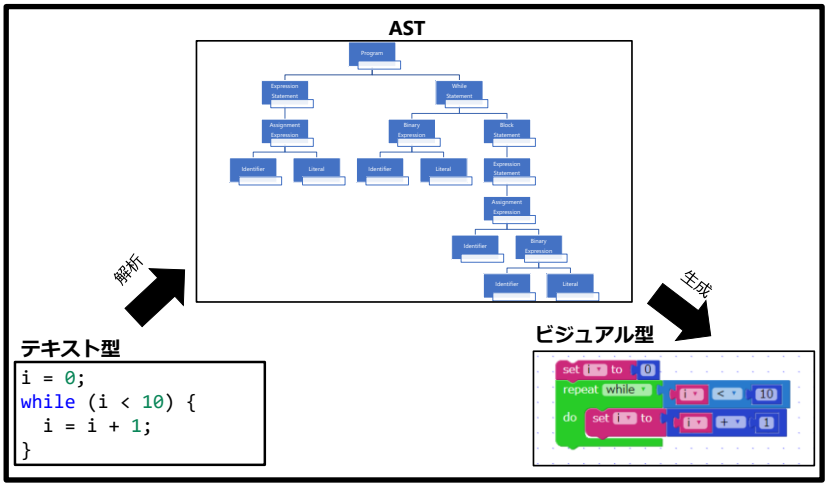

図 3.7：ビジュアル型への変換の流れ

構文解析ライブラリ esprima[17] で JavaScript のソースコードを AST に変換し、AST 探索ライブラリ estraverse[20]で AST の探索を行った．探索は一番上のノードから最深部のノードまで左節点を優先に探索を行う前順走査を行った．後順を行わない理由は二つあり，一つは estraverse では探索している子ノードの親ノードまでの情報しか取得できなかったからである．二つ目の理由は AST の構造は決まった構造をしているからである.

変換例として下記のテキスト型プログラム(While 文)の AST 探索結果及び生成結果を図 3.8 示す．図 3.8 の AST には大きく分けて二つの構成要素がある．一つ目は ExpressionStatement であり，ExpressionStatement では変数:i(identifer)とデータ:0(Literal)が AssinmentExpression にて代入式:i=0 とな

っている．二つ目は WhileStatement であり，WhileStatement では条件式は BinaryExpression に含まれており，変数:i とデータ:10 が BinaryExpression にて比較式:i<10 となっている．条件を満たしている間に動作する内容は BlockStatement({}) に含まれており，BlockStatement 内にある ExpressionStatement では，変数:i とデータ:1 が BinaryExpression に加算式:i+1 となり，変数:i と加算式:i+1 が AssinmentExpression にて代入式:i=i+1 となっている．

・While 文(10 回ループ)

```
i = 0;
while (i < 10) {
  i = i + 1;
}
```

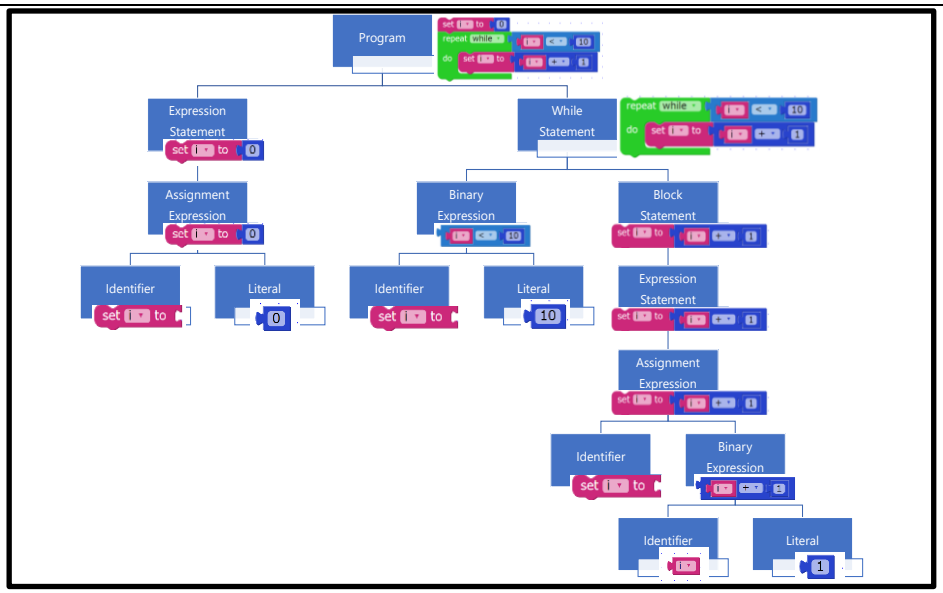

図 3.8：while 文 AST 探索/生成結果

3.5.2　変換の制限

本システムでのテキスト型からビジュアル型への変換には制限が存在する．本研究では，ビジュアル型プログラミング終えた学習者のテキスト型プログラミングへの移行を目的としているため，前提条件としてビジュアル型プログラムとの互換性があるテキスト型プログラムを変換の対象としている．そのため，ビジュアル型プログラムで再現できないテキスト型プログラムは変換を行うことができない．ビジュアル型プログラムで再現可能/不可能プログラム文法を表

3.1 に示す．表 3.1 に含まれていないプログラム文法(クラスやメソッド処理など)については再現ができない．

表 3.1：ビジュアル型プログラムへの変換可否リスト

プログラム文法	再現可能	再現不可能
変数	代入，参照	宣言
データ格納	配列	オブジェクト
算術演算子	+,-,*,/,%,**	++,--
比較演算子	==,!=,<,<=,>,>=	===,!==
論理演算子	&&,||	
条件分岐	If 文	switch 文
ループ処理	For 文，While 文	For in 文，For of 文，Label 文
関数	関数宣言，引数，定義，呼び出し	関数式，デフォルト引数

第 4 章　開発システム

本章では，実際に開発したビジュアル型とテキスト型プログラムの相互変換システムについて述べる．

4.1　開発システムの全体像

開発したプログラム相互変換システムの全体図を図 4.1 に示す．

インストールなどの手間省くためにブラウザ上で動作するシステムを構築した．ブラウザ側での動作(実行，変換)を HTML,CSS,JavaScript で実装し，プログラムのファイルやログを保存/読込するためにサーバ側では PHP を用いて実装した．

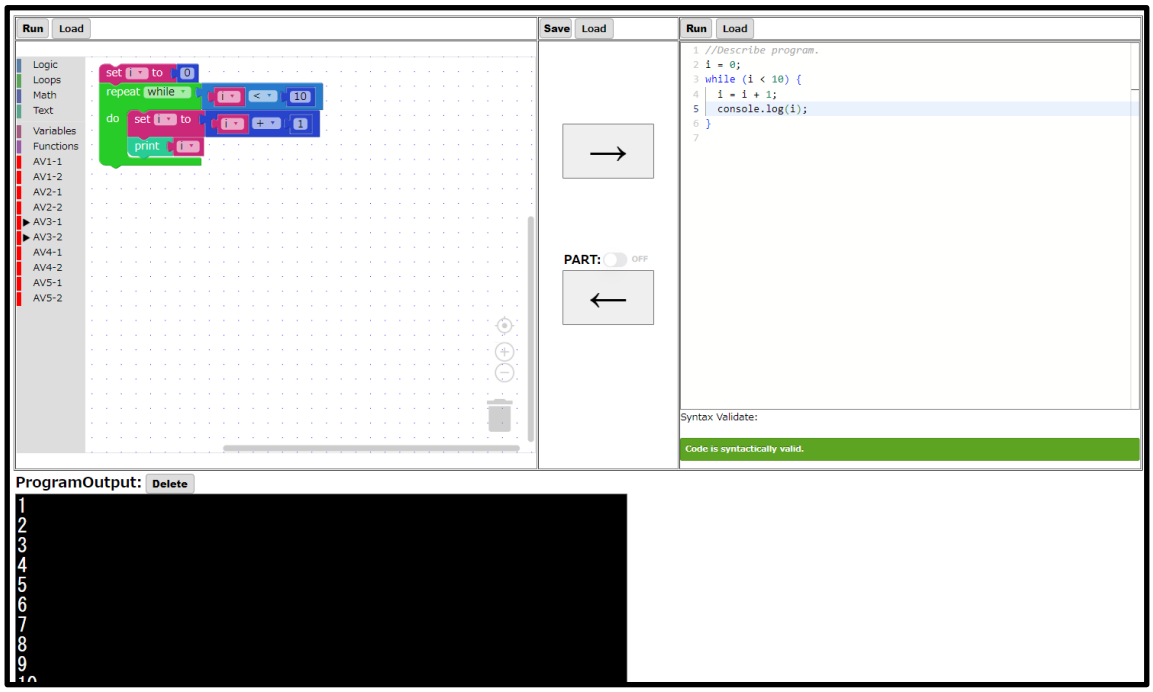

図 4.1：開発したシステムの全体図

図 4.1 のシステムは大きく分けて 4 つの部分からなっており，左上がビジュアル型操作部，右上がテキスト型操作部，真ん中がプログラム操作部，左下が実行結果部となっている．それぞれの部分の役割を以降で説明する．また，システム使用中に画面上部に表示される確認ダイアログに関しても以降で説明する．

4.2　ビジュアル型操作部

本節では，ビジュアル型プログラムのコーディング，実行，読込を行うビジュアル型操作部(図 4.2)について述べる．

ビジュアル型操作部では，構成要素として Run ボタン，Load ボタン，ツールバー，ビジュアル型プログラム記述部，ビジュアル型画面操作部，ゴミ箱がある．それぞれの構成要素の役割を以下に示す．

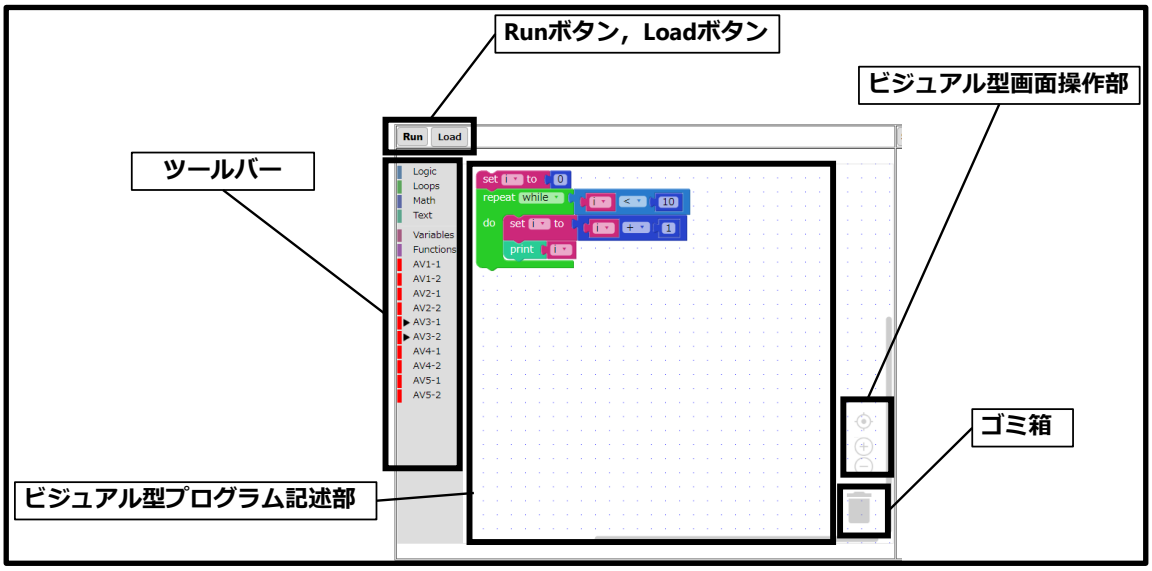

図 4.2：ビジュアル型操作部

- Run ボタン:
  ビジュアル型プログラミング記述部内のプログラムを実行する．
- Load ボタン:
  ビジュアル型プログラム(.xml)を読み込んでビジュアル型記述部に表示する．
- ツールバー:
  カテゴリー別にビジュアル型プログラムが内包されている．
  ビジュアル型プログラムをビジュアル型プログラム記述部からツールバーにドラック＆ドロップすることで削除できる．
- ビジュアル型プログラム記述部:
  ツールバーに内包されているビジュアル型プログラムをビジュアル型プログラム記述部にドラック＆ドロップし，ビジュアル型プログラム同士を組み

合わせてプログラミングを行う.

- ビジュアル型画面操作部:
  ビジュアル型プログラム記述部の画面の大きさを調整する.
  ⦿：プログラム中央に画面移動
  ＋：画面縮小
  －：画面拡大
- ゴミ箱:
  いらなくなったビジュアル型プログラムをここにドラック＆ドロップすることで削除することができる.

4.3　テキスト型操作部

本節では，テキスト型プログラムのコーディング，実行，読込を行うテキスト型操作部（図 4.3）について述べる.

テキスト型操作部では，構成要素として Run ボタン，Load ボタン，テキスト型プログラム記述部，構文検証部がある．それぞれの構成要素の役割を以下に示す.

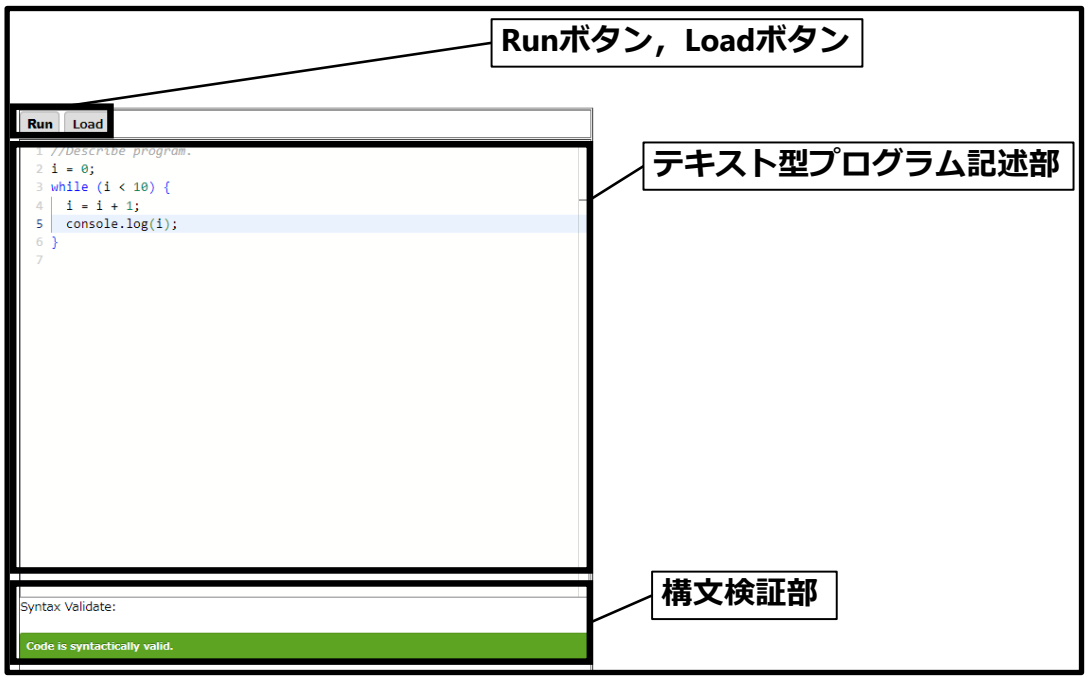

図 4.3：テキスト型操作部

- Run ボタン:
  テキスト型プログラミング記述部内のプログラムを実行する.
- Load ボタン:
  テキスト型プログラム(.js)を読み込んでテキスト型記述部に表示する.
- テキスト型プログラム記述部:
  ここにテキスト型プログラムを入力して, テキスト型プログラミングを行う. 改行すると，左に行数が表示される.
  テキスト型プログラム記述部内のプログラムに構文エラーがあると, エラー箇所を赤い波線でマークアップされる.
- 構文検証部:
  テキスト型プログラム記述部に入力されたプログラムが構文エラーを含んでいるか確認する.
  エラーがない場合，緑のテキストボックスが表示される.
  エラーがある場合, 赤のテキストボックスが表示され, テキストボックス内にエラー内容が表示される.

4.4 プログラム操作部

本節では，プログラムの変換や保存や読込を行うプログラム操作部（図 4.4）について述べる.

プログラム操作部では，構成要素として Save ボタン，Load ボタン，ビジュアル型からテキスト型への変換ボタン，テキスト型からビジュアル型への変換ボタンがある．それぞれの構成要素の役割を以下に示す.

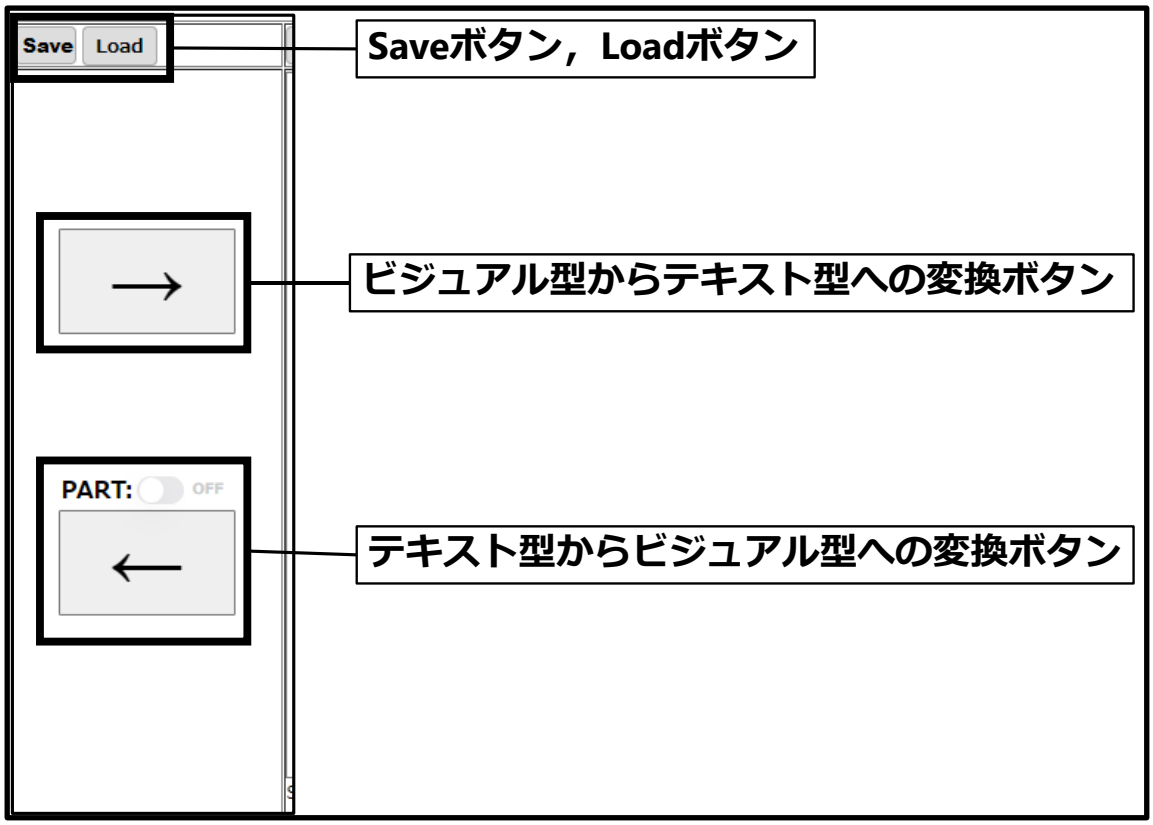

図 4.4：プログラム操作部

- **Save ボタン:**
  ビジュアル型とテキスト型プログラムを名前付けて保存する.
- **Load ボタン:**
  ビジュアル型とテキスト型プログラムを読み込んで表示する.
- **ビジュアル型からテキスト型への変換ボタン:**
  ビジュアル型記述部に入力されているビジュアル型プログラムをテキスト型プログラムに変換し，変換結果をテキスト型記述部に表示する.
- **テキスト型からビジュアル型への変換ボタン:**
  テキスト型記述部に入力されているテキスト型プログラムをビジュアル型プログラムに変換し，変換結果をビジュアル型記述部に表示する.
  テキスト型プログラムがエラーを含んでいる場合は，テキスト型からビジュアル型プログラムへの変換はできない.
  PART ボタンを ON にすると，テキスト型からビジュアル型プログラムへの部分変換が可能になる.
  テキスト型記述部内のマウスで範囲選択されているテキスト型プログラム

をビジュアル型プログラムに変換し，変換結果をビジュアル型記述部に表示する．

4.5　実行結果部

本節では，プログラムの実行結果やエラーを表示する実行結果部（図 4.5）について述べる．

実行結果部では，構成要素として Delete ボタン，実行結果表示部がある．それぞれの構成要素の役割を以下に示す．

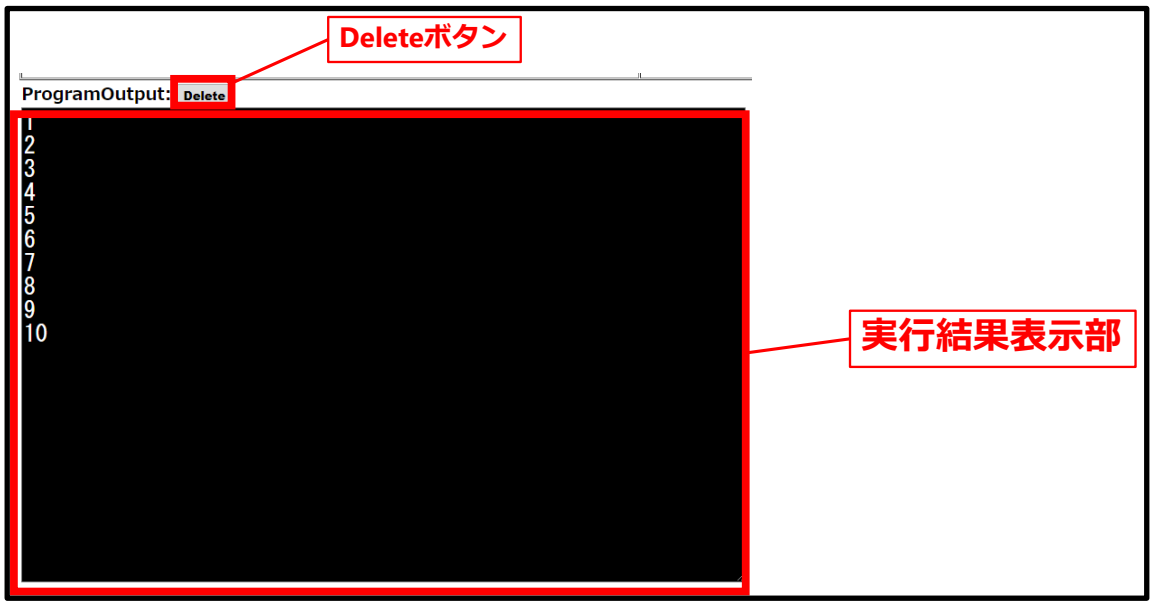

図 4.5：実行結果部

- **Delete ボタン:**
  実行結果表示部の内容を削除する．
- **実行結果表示部:**
  ここに実行結果が表示される．
  読み取り専用のため，内容を消したい場合は上の Delete ボタンを押す．
  実行/変換エラーを発生したときにエラー内容を表示する．

4.6　確認ダイアログ

本節では，システム使用中に上部に適宜表示される確認ダイアログ（図 4.6）について述べる．

確認ダイアログは，何らかのキーボード入力を必要とした場合と実行/変換エラーが発生したときにシステム上部に表示される．表示されている間は，システムが一時停止状態になり，「OK」または「キャンセル」をクリックすると，システムの動作が再開される．表示されるダイアログの役割を以下に示す．

図 4.6：確認ダイアログ

- **Save ダイアログ:**
  Save ボタンをクリックしたときに，表示されるダイアログ．
  保存したいファイルの名前をダイアログ内の入力し，OK をクリックすると，保存確認ダイアログが表示され，保存が完了したことを確認できる．
- **Variable ダイアログ:**
  ビジュアル型プログラミングにおいて変数の名前を入力するときに表示されるダイアログ．
  作成したい変数の名前をダイアログ内の入力し，OK をクリックすると，入力した名前の変数のビジュアル型プログラムがツールバーの Variables に追加される．
- **エラーダイアログ:**
  プログラムの実行/変換を行った際にエラーが発生すると，エラーメッセージを含んだダイアログが表示される．
  実行エラーの場合，「実行エラー：(エラーメッセージ)」．
  変換エラーの場合，「変換エラー：(エラーメッセージ)」．

第 5 章　評価実験

本章では，開発した相互変換システムを評価するために行った実験とその結果について述べる．

5.1　実験の目的

本実験では，被験者に開発したビジュアル型とテキスト型プログラム相互変換プログラムシステムを利用してプログラミングを行ってもらい．テキスト型プログラミング(JavaScript)がどれくらい習得できたかを判断する．プログラミングを行ってもらう際に相互変換を利用した場合と利用しなかった場合の習得度の比較を行い，システムの有用性を示す．

5.2　実験の方法

5.2.1　実験条件

本実験の被験者の対象は大学院に所属していて，情報系を専門としていない学生を対象とした．本実験の条件は，日本語が読めることとパソコンの基本的な操作ができることとした．理由としては，実験時に使用する資料が日本語で記載されているため，システムを使用するためにマウス操作やキーボード入力といった操作が必要だからである．また，実験において学習の順序に依存にする効果を減少させるために被験者毎に変換あり/なしの学習順番を入れ替え，学習に使用した問題２個（A,B）に関しても順番を入れ替えた．被験者４人毎に順序が１セットとなり，これらを繰り返して実験を行った．順序の１セットを表 5.1 に示す．

表 5.1：実験の順序

Group ID	学習 1		学習 2	
01	A	変換なし	B	変換あり
02	B	変換なし	A	変換あり
03	A	変換あり	B	変換なし
04	B	変換あり	A	変換なし

5.2.2　実験手順

実験を以下の(0)～(9)の手順で，計 120 分の時間で行った．

0) 実験の説明，同意書への記入（所要時間: 15 分）

実験の目的について被験者に詳細な説明を行う．

被験者は実験に参加する意思を示すため，同意書に署名をしてもらう．

1) 事前アンケートの実施（所要時間: 5 分）

ビジュアル型とテキスト型プログラミング学習経験について回答してもらう．

2) プレテスト（所要時間: 5 分）

テキスト型プログラミング(JavaScript)の習得度を測るテストを行う．

3) ビジュアル型プログラミングの練習（所要時間: 5 分）

Google Blockly の Blockly Game[21]を使用して，ビジュアル型プログラミングの練習を行う．

4) 学習 1：変換システムを使用しないプログラミング学習（所要時間: 30 分）

ビジュアル型プログラムからテキスト型プログラムまたはテキスト型プログラムからビジュアル型プログラムへの変換を行うシステムを使用せずにプログラミング学習を行ってもらう．

5) 中間テスト（所要時間: 5 分）

テキスト型プログラミング(JavaScript)の習得度を測るテストを行う.

6) 休憩（所要時間: 10 分）

7) 学習 2：変換システムを使用したプログラミング学習（所要時間: 30 分）

ビジュアル型プログラムからテキスト型プログラムまたはテキスト型プログラムからビジュアル型プログラムへの変換を行うシステムを使用してプログラミング学習を行ってもらう.

8) ポストテスト（所要時間: 5 分）

テキスト型プログラミング(JavaScript)の習得度を測るテストを行う.

9) 事後アンケートの実施（所要時間: 10 分）

変換システムの精度や有意性そしてユーザビリティ, 両型のプログラミングの習得度，自由記述を回答してもらう.

5.2.3 学習範囲

本実験で被験者がプログラミングを通して学習する学習対象は変数，算術演算，If 文(条件分岐文)，While 文(繰り返し文)，関数の五つとした．また，If 文や While 文の条件式には論理演算子(&&,||)を対象とせず，データ格納についても配列を使用しなかった. 5 つのそれぞれのプログラミング学習対象で学習できる学習項目を表 5.2 に示す.

表 5.2：プログラミング学習対象と学習項目

学習対象	学習項目
変数	代入，参照
算術演算	和，差，積，商の算術
If 文	条件式，条件合致/不合致時の動作
While 文	条件式，繰り返しの動作
関数	引数，関数の動作，呼び出し

表 5.2 のプログラミング学習項目を定めた理由は, 比較を行いやすくためにビジュアル型(Google Blockly)とテキスト型(JavaScript)の両型で学習できるものにし，互換性があるものにする必要があったためである.

5.2.4　学習の流れ

実験の手順においてプログラミングの学習(5.2.2 の(4)と(7))を行う際に以下のｉ～ｖの流れを表 5.2 の学習対象毎に行った.

i. 事前学習

問題を解く前に学習に必要なプログラミングの知識が記載された資料を読んで，学習を行う.

ii. 選択肢付問題の解答

プログラムの選択肢が付いた問題を解答してもらう.

選択肢には間違った選択肢が含まれており，その中から正しいプログラムを指定数個選んで，プログラミングを行う.

iii. 選択肢付問題の解答ファイルの保存

選択肢付問題の解答ファイルをサーバに保存する.

次の問題に進んだ後に前の問題の解答ファイルを再保存はできない.

iv. 穴埋め問題の解答

プログラムに穴が開いた問題を解答してもらう.

穴が開いている箇所は主に学習項目での学習においてコーディングする箇所となっている.

v. 穴埋め問題の解答ファイルの保存

選択肢付問題の解答ファイルをサーバに保存する.

次の問題に進んだ後に前の問題の解答ファイルを再保存はできない.

5.2.5　学習問題

本節では，5.2.2 実験の手順(4),(7)において学習する際に使用する問題(以降学習問題と表記する)について述べる.

学習問題は選択肢付問題と穴埋め問題の二種類用意し，それぞれの問題はビジュアル型とテキスト型で記載方式が異なる．学習にはスライドを用いており，スライド内に選択肢付問題と穴埋め問題が記載され，スライド内に記載されている実行結果と同じ結果になるプログラムを作成する問題になっている.

ビジュアル型の選択肢付問題と穴埋め問題は，ビジュアル型環境のツールバーに内包されており，図 4.2 内の AV1-1～AV5-2 のフォルダーがそれに該当する．フォルダー名の最後が 1 なら選択肢付問題，2 なら穴埋め問題となってお

り，例として AV1-1 は変数の選択肢付問題，AV1-2 は変数の穴埋め問題となっている．ビジュアル型においての穴埋め問題は，足りないプログラムを補う問題となっているため，ツールバーにから適切なプログラムを選ぶことが期待される．学習問題の例として，ビジュアル型の変数の選択肢問題を図 5.1 に示し，穴埋め問題を図 5.2 に示す．

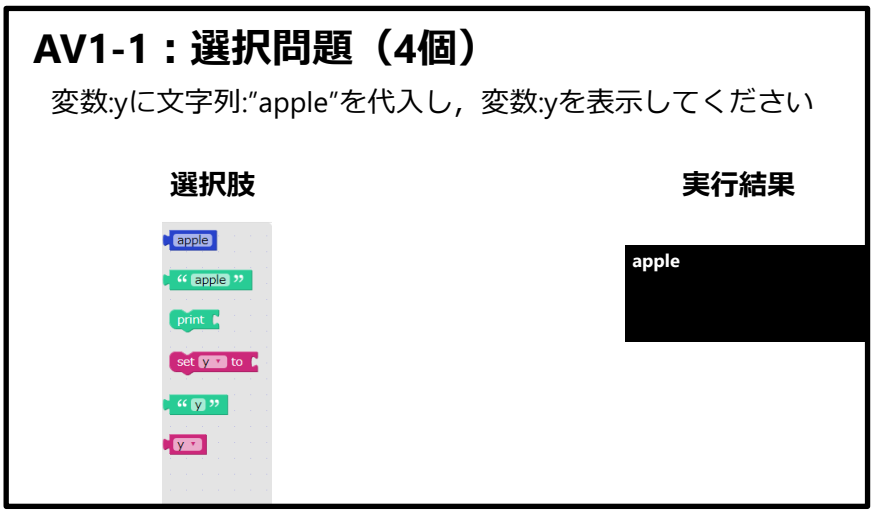

図 5.1：ビジュアル型の選択肢付問題(変数)

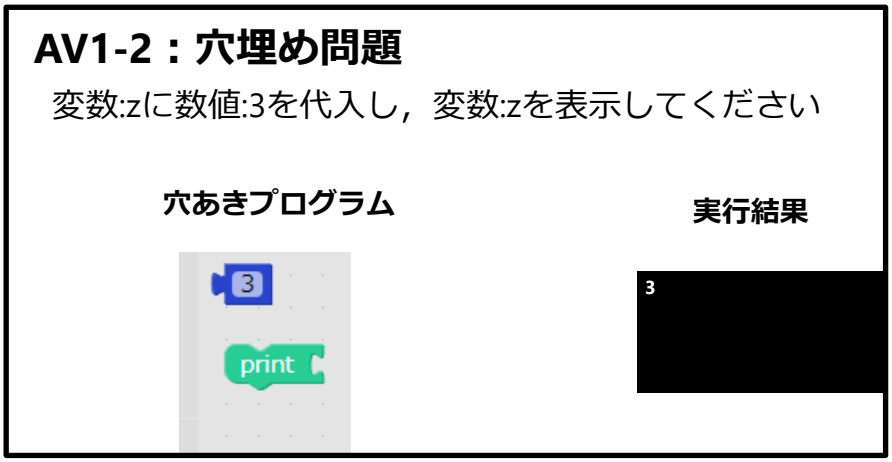

図 5.2：ビジュアル型の穴埋め問題(変数)

テキスト型の選択肢付問題と穴埋め問題は，問題のプログラムがスライド内に記載されており，そこからコピー＆ペーストすることによって解くことを想定している．テキスト型においての穴埋め問題は，穴埋め該当箇所に適切なプログラムをコーディングする問題となっている．学習問題の例として，テキスト型

の変数の選択肢問題を図 5.3 に示し，穴埋め問題を図 5.4 に示す．

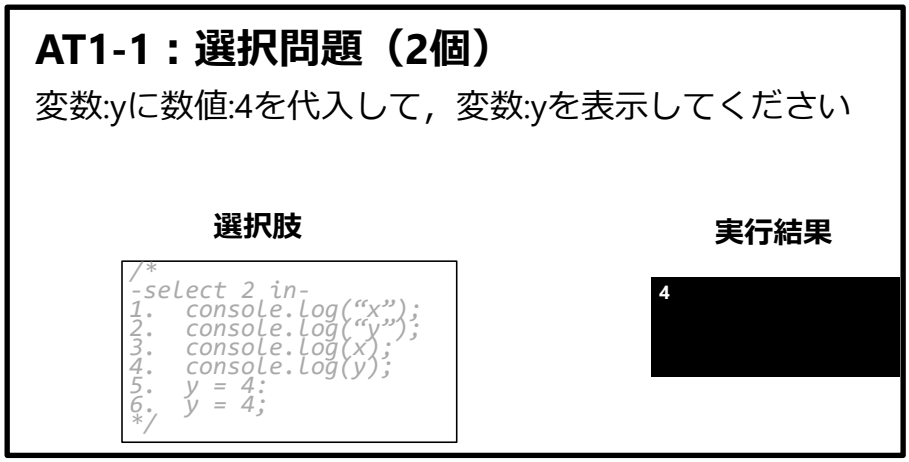

図 5.3：テキスト型の選択肢付問題(変数)

図 5.4：テキスト型の穴埋め問題(変数)

5.2.6　習得度把握テスト

本節では，5.2.2 実験の手順(2),(5),(8)において行う習得度把握テストについて述べる．

実験内で合計 3 回行うテストについては，表 5.2 の学習対象毎に一問用意し，5 点満点のテストとした．また，テキスト型プログラミングの習得を測るためのプログラミング言語を JavaScript とし，全問全て選択問題とした．

一回目で行うテストは，学習前における被験者のテキスト型プログラミング(JavaScript)の習得度を測るために行う．二回目，三回目で行うテストは，学習後における被験者のテキスト型プログラミング(JavaScript)の習得度を測るため

に行う．一回目，二回目，三回目で行うテストは，問題と選択肢は同じ内容のため，回数を追うごとに解答が変化することが期待される．

5.3　実験結果・評価

情報系を専門としていない大学院生 8 名を対象にプログラミング相互変換システムの有効性を示すための実験を行い以下のような結果を得た．

5.3.1　事前アンケート

事前アンケートでは，ビジュアル型とテキスト型プログラミングの経験について GoogleForm 上にて回答を求めた．事前アンケートの結果として，ビジュアル型とテキスト型の経験なし，ビジュアル型とテキスト型の経験あり，テキスト型のみ経験ありの 3 種類に分類することができ，結果を表 5.3 に示す．

表 5.3：事前アンケートの結果

プログラミング経験	人数
特になし	1
ビジュアル型とテキスト型	1
テキスト型のみ	6

表 5.3 の内訳として，6 名が経験していたテキスト型プログラミング言語は VisualBasic,Python,C,HTML,CSS,JavaScript,R となっており，経験期間は 2 カ月～5 年だった． 1 名が経験していたビジュアル型プログラミング言語は Scratch となっており，経験期間は 2 年だった．

5.3.2　習得度把握テスト

実験の被験者の 3 回の習得度把握テストの全体結果を表 5.4 に示す.

表 5.4：習得度把握テストの全体結果

被験者 ID	一回目	二回目	三回目
01	2	2	3
02	2	3	3
03	2	3	4
04	3	5	5
05	3	4	4
06	2	4	4
07	0	1	2
08	2	3	4

表 5.4 の一回目，二回目，三回目の対応のある三種のデータにフリードマン検定を用いて，有意確率を求めた．計算結果を表 5.5 に示す.

表 5.5：フリードマン検定全体結果

自由度	検定統計量	有意確率
2	14	0.0009119

表 5.5 より表 5.4 のデータは 5%水準で有意差があることが判明したため，ウィルコクソン符号順位検定でそれぞれの組み合わせの有意確率を求めた後にボンフェローニ検定で有意確率を求めた. ボンフェローニ検定の結果を表 5.6 に示す.

表 5.6：ボンフェローニ検定全体結果

	一回目	三回目
三回目	0.034	-
二回目	0.053	0.216

表 5.6 より，一回目と三回目の組み合わせに 5%水準の有意差があること，他の二つの組み合わせは 5%水準の有意差がないことが判明した．このことにより，開発システムを利用してプログラミング学習を行った結果，テキスト型プログラムの習得の促進を行うことができたと考えられる．

本実験では，実験の流れの学習 1 と学習 2 でシステムの相互変換を利用する/しないを被験者毎に分けて学習を行った．相互変換利用した前後と利用しなかった前後での点数差を表 5.7 に示す．

表 5.7：習得度把握テストの点数差

被験者 ID	相互変換なし	相互変換あり
01	0	1
02	1	0
03	1	1
04	0	2
05	1	0
06	2	0
07	1	1
08	1	1

表 5.7 の相互変換なしと相互変換ありある 2 種の対応するデータにウィルコクソン符号順位検定を用いて，有意確率を求めた．計算結果を表 5.8 に示す．

表 5.8：ウィルコクソン符号順位検定の結果(相互変換なし/あり)

有意確率
0.89

表 5.8 より，今回の結果では相互変換なしと相互変換ありの条件における学習効果の差は観測されなかった．

5.3.3　事後アンケート

事後アンケートでは, システムを使用してプログラミング学習を終えた後に 7 つの質問を GoogleForm 上にて回答を求めた．以下に質問内容を示す.

質問 1：
テキスト型からビジュアル型へのプログラムの変換は上手くできていましたか？
質問 2：
ビジュアル型からテキスト型へのプログラムの変換は上手くできていましたか？
質問 3：
プログラムの学習支援(変換)システムは学習の役に立ちましたか？
質問 4：
学習支援システムは使いやすかったですか？
質問 5：
ビジュアル型プログラミングは習得できましたか？
質問 6：
テキスト型プログラミングは習得できましたか？
質問 7：
実験全体についてのフィードバックや感想がございましたらご記入ください.

質問 1〜6 に関しては， 7 段階(1:否定的〜7:肯定的)で回答した後にその理由を記述した．質問 7 に関しては記述だけで回答した．事後アンケートの質問における被験者の段階の回答結果を表 5.9 に示す．また，それぞれの質問の理由を表 5.10〜5.15 に示し，質問 7 で回答されたシステムのフィードバックを表 5.16 に示す.

表 5.9：事後アンケートの結果

被験者 ID	質問 1	質問 2	質問 3	質問 4	質問 5	質問 6
01	6	6	2	7	3	5
02	7	7	7	5	7	4
03	5	5	4	3	4	3
04	5	6	2	6	7	4
05	6	7	6	6	6	6
06	4	2	4	4	5	6
07	7	7	7	6	7	6
08	7	7	7	5	5	4

表 5.10：事後アンケート質問 1 の理由

理由	人数
想像通りに変換されていたため	3
変数の宣言文が変換されなかったため	2
変換システムが使いやすかったため	2

表 5.11：事後アンケート質問 2 の理由

理由	人数
想像通りに変換されていたため	3
変換時に不要な変数が増えたため	1
変換システムが使いやすかったため	2

表 5.12：事後アンケート質問 3 の理由

理由	人数
視覚的に分かりやすかったため	3
プログラムが合っているか確認できるため	1
直感的に理解しやすかったため	1
両型の表現が異なり，解答に混乱したため	1
変換するより自分で書いた方が速いため	1
精度がそんなに高くないため	1

表 5.13：事後アンケート質問 4 の理由

理由	人数
UI が簡潔で分かりやすかったため	3
両型を並べて学習することで理解が速くなったため	1
ビジュアル型プログラムの操作がしにくかったため	4

表 5.14：事後アンケート質問 5 の理由

理由	人数
視覚的/感覚的に理解しやすかったため	2
ビジュアル型プログラミングのやり方を覚えたため	3
自分が使用している言語の表現に近いため	1

表 5.15：事後アンケート質問 6 の理由

理由	人数
実験時間内に全て学習できなかったため	2
テキスト型プログラミングの方が得意なため	2
学習忘れがあったため	2

表 5.16：開発システムのフィードバック

フィードバック
テキスト型からビジュアル型への変換が良かった.
プログラム消去(リセット)機能を追加して欲しい.
エラーをもっと視覚的に強調した方が良い.
テキスト型とビジュアル型の色を統一した方が良い.
保存/読込機能は中規模プログラミングのチャレンジに役立つ.

表 5.10 と 5.11 のプログラムの変数文の変換に関しての否定的な意見や表 5.12 の「変換の精度がそんなに高くなかったため」の意見が発生した原因は，システム仕様の説明不足であり，変換可能なプログラムと不可能なプログラムを十分に説明できていなかったことが考えられる． 表 5.13 の「ビジュアル型プログラムの操作がしにくかったため」についても同様で，ブロックの複数複製方法について説明できなかったためと考えられる.

表 5.12 の「変換するより自分で書いた方が速いため」については，今回の実験では変数，算術演算，If 文，while 文，関数といった比較的小規模のプログラミング学習を行ったため，変換を行うより自身でコーディングした方が速くなったと考えられる.

表 5.12 の「両型の表現が異なり，解答に混乱したため」を解決するためには，表 5.16 の「テキスト型とビジュアル型の色を統一した方が良い.」という意見を取り入れて，ビジュアル型とテキスト型プログラムの表現の差を小さくすればよいと考えられる.

表 5.15 の「学習忘れがあったため」を解決するためには，表 5.16 の「エラーをもっと視覚的に強調した方が良い.」という意見を取り入れて，プログラミングのコーディングをより支援をする機能が必要だと考えられる.

5.4　エラー率

ビジュアル型及びテキスト型プログラミング学習の際に被験者が行ったプログラムの実行/変換の回数と発生したエラー数をまとめたものを表 5.17 に示す.

表 5.17：プログラムの実行/変換の合計動作数/エラー数

動作	回数	エラー数
ビジュアル型プログラムの実行	200	0
テキスト型プログラムの実行	215	17
ビジュアル型からテキスト型への変換	34	0
テキスト型からビジュアル型への変換	28	0

表 5.17 より，エラーが発生したのはテキスト型プログラムの実行だけであった．プログラミングのログから取得できたテキスト型プログラムの実行において発生したエラーを種類分けしたものを表 5.18 に示す.

表 5.18：テキスト型プログラムの実行エラーの種類と発生回数

エラー内容	発生回数
トークンエラー {:Invalid or unexpected token, Unexpected token '〜'}	4
定義エラー{:〜 is not defined}	10
識別子エラー{:Unexpected identifier '〜'}	1
関数呼び出しエラー{:missing) after argument list}	2

表 5.18 より，発生数は定義エラー>トークンエラー>関数呼び出しエラー>識別子エラーの順になっていることが分かった．定義エラーが一番多い理由は，最初から最後まで変数を使うためだと考えられる．その他のエラーに関しては，個人差によって発生の有無が変化していると判断した.

ビジュアル型プログラムの実行及びビジュアル型からテキスト型への変換においてエラーが起きないのは，ビジュアル型の環境に用いた Google Blockly[8]の仕様でエラーが起こらなかったためと考えられる.

テキスト型からビジュアル型への変換においてエラーが発生しなかった理由

は，事前にテキスト型プログラムにエラーが含まれていると変換できないことを伝えていたこととテキスト型環境の構文検証部にてエラーの確認がしやすかったためだと考えられる．そのほかには，学習に使用した問題の形式として選択肢付問題と穴埋め問題を用いたことによって記述量が少なくなったためとも考えられる．

5.5　変換精度の評価

参考書[22]内に記載されているテキスト型プログラムを用いて，テキスト型からビジュアル型への変換精度の評価を行った．評価に使ったテキスト型プログラムはビジュアル型と互換性のあるもので，学習対象の変数，算術演算，If 文，While 文，関数の 5 つからそれぞれ 2 つずつを用いて計 10 個の変換精度の実験を行った．

変換精度の計算は以下の計算式で行い，結果を表 5.19 に示す．

$$変換精度 = \frac{変換できたブロック数}{必要なブロック数}[\%]$$

表 5.19：変換精度の実験結果

学習項目	変換できた数	必要な数	変換率[%]
変数	4	4	100
	8	8	100
算術演算子	6	6	100
	8	8	100
If 文	12	12	100
	14	14	100
While 文	12	12	100
	12	12	100
関数	4	4	100
	9	10	90

表 5.19 より，関数以外の学習項目の変換精度は 100%だった．関数の変換率が 90%の理由は，関数の呼び出し命令文の AST を探索する際に引数のデータが取得できなかったためである．そのため，テキスト型の関数の呼び出し命令文をビジュアル型へ変換を行った際にビジュアル型の引数の表記が”undefined”(図 5.5)になってしまい，関数の呼び出しが正しく動作しなくなることが判明した．この問題を解決するためには，関数の定義文の AST を探索した際に引数の情報を関数毎に保存し，対応づける必要があると考えられる．

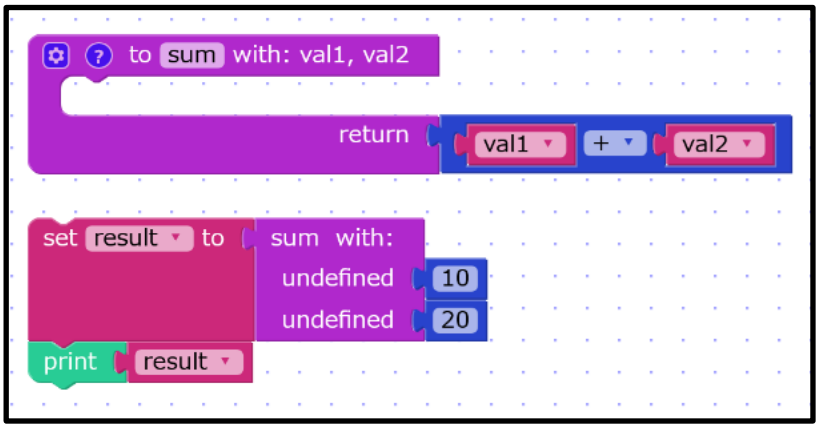

図 5.5：関数呼び出し命令文の変換失敗例

第 6 章　結論

本章では, 開発したプログラム相互変換システムについての評価をまとめ, 今後の課題について述べる.

6.1　まとめ

本研究では，ビジュアル型プログラミングを終えた学習者がテキスト型プログラミングに移行できないという課題に着目し，この課題を解決するためにビジュアル型とテキスト型プログラムの相互変換システムを開発し，テキスト型プログラミングの学習の促進を行った．この目的達成のために①ビジュアル型とテキスト型プログラミングが比較しやすい環境を構築し，②ビジュアル型とテキスト型プログラムの相互変換を実装し，③開発した学習支援システムの有用性を示す実験を行った.

①の目的達成のためにビジュアル型環境を Google Blockly[8]を用いて構築し, テキスト型環境を MonacoEditor[18]を用いて構築し，ブラウザ上で左右に両方の環境を展開することによって, 比較しやすい環境を開発した．また，テキスト型環境のエラー支援として構文検証部を導入することによって，テキスト型プログラミング時の文法エラーの発生を削減し，ビジュアル型プログラミングのとの差を少なくした.

②の目的達成のためにビジュアル型からテキスト型への変換は Google Blockly[8]の内部プログラムの変換機能を用いて行い，テキスト型からビジュアル型への変換はプログラムの AST を解析することにより実現した．開発システムの中央部にある 2 種類の変換ボタンを押すだけで変換できるようにすることにより，変換行為を簡単に行えるようにした.

③の目的達成ために開発したシステムを用いてプログラミング学習を行う実験を情報系を専門としていない大学院生 8 名を対象に行った. 実験では, プログラミング学習二回と習得度把握テスト三回, アンケート二回を行った. 学習対象を変数，算術演算，If 文，While 文，関数の 5 つとし，習得度把握テストでどれだけの学習対象を学習できたか判断した. その結果, 開発した学習支援システム自体はテキスト型プログラミング学習に有用性があると示すことができたが, 相互変換機能は有用性あることを示せなかった.

6.2　今後の課題

本研究で開発した学習支援システムにはいくつかの問題があるためその部分を改善する必要がある．

本研究で開発したシステムではビジュアル型とテキスト型プログラミングが比較しやすい環境を構築したが，より比較しやすい環境が構築できると考えられる．実験のアンケートのフィードバックで「テキスト型とビジュアル型の色を統一した方が良い．」という意見があり，より比較しやすい環境にするために文字やブロックの色を統一するべきである．また，プログラムの複製/消去法に関してもビジュアル型とテキスト型環境で差が生じてしまっている．テキスト型プログラムは範囲選択での複製/消去ができるが，ビジュアル型では組み合わさったプログラムでしか複製/消去ができないため，ビジュアル型にも複数の独立したプログラムを複製/消去できる機能が必要である．

ビジュアル型とテキスト型の変換に関しても改善が必要である．本研究ではテキスト型からのビジュアル型への変換はプログラムの AST を用いて行った．しかし，変換できるものは決められたビジュアル型プログラムだけになってしまっているため，自由に記述できるテキスト型プログラムの全てに対応することができない．この問題を解決するためには，AST からビジュアル型プログラムを自動で生成する機能を実装する必要がある．また，変換の精度が低くなってしまうと学習者が適切にプログラミング学習できないため，精度も重要視する必要がある．

対応しているプログラミング言語に関しても改善が必要である．本研究では習得するテキスト型プログラミング言語を JavaScript とした．しかし，他のプログラミング言語で本研究と同じように学習した場合にどのような学習結果が得られるかが分からないため，対応言語の増加も必要だと考えており，学習者が学習したいプログラミング言語を選ぶことができる環境にするべきである．

実験の対象にも改善が必要である．本研究では両型経験なし 1 名，両型経験あり 1 名，テキスト型のみ経験者 6 名を対象に実験を行ったため，対象が偏った実験になった．そのため，相互変換のビジュアル型からテキスト型への移行に関する有効性の評価を十分に行えなかった．そのことより，実験対象に両型経験なし，ビジュアル型のみ経験者の数を増やす必要がある．

対外発表

野口，太田，谷，長谷川，”ビジュアル型とテキスト型プログラムの相互変換を用いたプログラミング学習支援システム”：教育システム情報学会学生研究発表会北信越支部(2024 in press)

謝辞

本研究の遂行にあたり，研究テーマ決定から論文作成までにおいて多くの助言やアドバイスを頂きました長谷川忍教授及び研究室の皆様に心より感謝いたします．また，JAIST の充実した教育により電子系から情報系への分野変更ができたことも誠に感謝申し上げます．最後に実験の被験者として参加してくださった皆様に対しましても，感謝の念を示します．

参考文献

[1] 文部科学省.“高等学校学習指導要領 情報科関係資料”.https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/mext_01831.html(参照 2023-1-15)

[2] 文部科学省.”新学習指導要領に対応した令和 7 年度大学入学共通テストにおける「情報 I」について”. https://www.ipsj.or.jp/event/taikai/84/ipsj_web2022/html/event/pdf/maeda-20220305.pdf(参照 2023-1-15)

[3] Scratch Team Lifelong Kindergarten Group MIT Me-dia Lab. “Scratch - Imagine.Program.Share”. http://scratch.mit.edu/(参照 2024-01-15)

[4] Sony Marketing Inc. “MESH｜誰でも手軽にプログラミングができる｜つくって楽しい｜学んで楽しい”. https://meshprj.com/jp/(参照 2024-01-15)

[5] 合同会社デジタルポケット.“ビスケット viscuit｜ コンピュータは粘土だ!!”. https://www.viscuit.com/(参照 2024-01-15)

[6] devjobsscanner. “Top 8 Most Demanded Programming Languages in 2023”. https://www.devjobsscanner.com/blog/top-8-most-demanded-programming-languages/(参照 2024-01-26)

[7] 総務省.“プログラミング人材育成の在り方に関する調査研究報告書”. https://www.soumu.go.jp/main_content/000424363.pdf(参照 2024-01-15)

[8] Google. “Blockly ｜ Google for Developers”. https://developers.google.com/blockly?hl=ja(参照 2024-01-15)

[9] 山梨裕矢・佐々木晃 .“ビジュアルプログラミングとテキスト記述形式の連携による初学者向けプログラミング学習システムの提案 .” *プログラミング・シンポジウム予稿集* , 2018 , 第 59 回, 2018 , p.79-84 .

[10] 山梨裕矢 .“構文誤りを含むプログラムのブロック言語表現への変換.” *法政大学大学院紀要. 情報科学研究科編*, 14, 2019, p.1-6 .

[11] 末吉春一・佐藤喬 .”ビジュアルプログラミングを用いたテキストベースプログラミング学習支援システム.” *第 78 回全国大会講演論文集*, 2016, 1 , 2016, p.897-898

[12] 松澤芳明・酒井三四郎 .“ビジュアル型言語とテキスト記述型言語の併用によるプログラミング入門教育の試みと成果 .” *情報処理学会研究報告* , コンピュータと教育研究会報告 Vol.2013, No.2, 2013, p1-11.

[13] Microsoft. “Microsoft MakeCode”. https://www.microsoft.com/ja-jp/makecode(参照 2024-01-15)
[14] A. P. Black, K. B. Bruce, M. Homer, J. Noble, A. Ruskin, and R. Yannow, “Seeking Grace: a new object-oriented language for novices.” SIGCSE’13, 2013, p129-134
[15] N. J. Homer, M, “Combining tiled and textual views of code.” IEEE VISSOFT, 2014, p.1–10.
[16] Microsoft. “Monaco Editor”. https://microsoft.github.io/monaco-editor/ (参照 2024-01-15)
[17] “esprima”. https://github.com/jquery/esprima(参照 2024-01-15)
[18] “Esprima: Syntax Validator”. https://esprima.org/demo/validate.html (参照 2024-01-15)
[19] numb_86. “AST で JavaScript のコードを変換する - 30 歳からのプログラミング”. https://numb86-tech.hatenablog.com/entry/2020/09/11/104131 (参照 2024-01-15)
[20] “estraverse”. https://github.com/estools/estraverse(参照 2024-01-15)
[21] Google. “ブロックリー・ゲーム”. https://blockly.games/(参照 2024-01-15)
[22] CodeMafia 外村将大 . 独習 JavaScript 新版 . 翔泳社 , 2021.