**２０２１　年度　　卒業論文**

卒業論文題目

|  |
| --- |
| 日本語環境ブロックプログラミングと連携した  ソースコードの穴埋め選択問題生成システム |

　指導教員　　　　　　鷹野　孝典　教授

神奈川工科大学　情報工学科

　　　　　学籍番号　　　　1821121

　　　　　学生氏名　　　　島岡　慎也

　提出日 2021年　12月23日　指導教員　　印

　受理日 2021年　12月23日　情報工学科長　　 印

# **論文要旨**

プログラミング初級者を対象として論理的思考を鍛えるために，ブロックプログラミングと呼ばれる，プログラミングの導入に利用されるシステムが存在する．これによりブロックを組み立てることで積み木の様にプログラミングを行える．一方，実際に運用されるシステム開発などでは，プログラミング言語を用いたコーディングが必要とされる．プログラミングの初級教育から中高等教育にかけて，論理的思考力からコーディング力の養成に円滑に移行できるような教育支援も考えていく必要がある．本研究では，ブロックプログラミングからコーディング用の練習問題を自動生成することに着目し，ブロックプログラミングと連携したソースコードの穴埋め問題生成システムを提案する．提案システムでは，Blocklyによって利用できる予約語と不等号や演算式記号をカテゴリーグループとして分けている．そして，ソースコード単語ごとにふりわけて，一つずつカテゴリーグループ内の単語と比較し，一致した単語と箇所を保存する．その中から乱数的にいくつかの箇所をピックアップして，穴埋め問題の生成をおこなっている．提案システムの問題自動生成機能によって生成される穴埋め選択問題が，ブロックプログラミングからコーディング学習のために適切に問題を生成可能であるかを難易度や出題基準の観点から評価することで提案システムの実現可能性を検証する．

# 目次

[**1.** **まえがき** 1](#_Toc90999288)

[1.1 はじめに 1](#_Toc90999289)

[1.2 本論文の構成 1](#_Toc90999290)

[2. 関連研究 2](#_Toc90999291)

[2.1 論理的思考に関する研究 2](#_Toc90999293)

[2.2 アルゴリズム的思考に関する研究 2](#_Toc90999294)

[2.3 問題自動生成システムに関する研究 2](#_Toc90999295)

[**3.** **提案システム** 4](#_Toc90999296)

[3.1 提案方式の概要 4](#_Toc90999298)

[3.2 提案方式の構成 4](#_Toc90999299)

[3.2.1. 問題ファイル 4](#_Toc90999305)

[3.2.2. 問題のXMLコードの用意 4](#_Toc90999306)

[3.2.3. 問題を自動生成する 5](#_Toc90999316)

[3.2.4. 選択問題の選択肢の決定 5](#_Toc90999317)

[3.2.5. 解答の正誤を採点 5](#_Toc90999318)

[3.2.6. 誤った解答の解説の表示 5](#_Toc90999319)

[3.3 問題の作成方法（教師視点） 6](#_Toc90999320)

[3.4 学習方法（学習者視点） 6](#_Toc90999321)

[**4.** **実装システム** 7](#_Toc90999322)

[4.1 実装システムの概要 7](#_Toc90999324)

[4.2 実装システムの実行環境構築 7](#_Toc90999325)

[4.2.1. 本研究のプログラムのダウンロード 7](#_Toc90999332)

[4.2.2. Node.js 8](#_Toc90999333)

[4.2.3. サーバの実行コマンド 8](#_Toc90999334)

[4.3 実装システムの機能説明 8](#_Toc90999335)

[4.3.1. Visual Programming Language 8](#_Toc90999338)

[4.3.2. コードジェネレート機能 9](#_Toc90999339)

[4.3.3. インデント 9](#_Toc90999340)

[4.3.4. JavaScriptコードの実行 10](#_Toc90999341)

[4.4 実装システムの選択式問題の実装 12](#_Toc90999342)

[4.4.1. XMLコードの生成とJavaScriptコードへの変換 12](#_Toc90999344)

[4.4.2. 穴埋め問題の生成 12](#_Toc90999346)

[4.5 実装システムの選択式問題の実装 13](#_Toc90999347)

[4.5.1. 穴埋め問題の解答の評価 13](#_Toc90999350)

[4.5.2. 実行結果と解答例との比較 14](#_Toc90999351)

[4.5.3. 誤答の場合の解説の表示 15](#_Toc90999352)

[4.6 Blocklyの変更点 16](#_Toc90999353)

[4.6.1. text\_printを変更 16](#_Toc90999355)

[4.6.2. for文を変更 16](#_Toc90999356)

[**5.** **実験** 18](#_Toc90999357)

[5.1 実験目的 18](#_Toc90999359)

[5.2 実験環境 18](#_Toc90999360)

[5.3 実験 18](#_Toc90999361)

[5.4 実験結果 18](#_Toc90999362)

[5.5 実験の考察 19](#_Toc90999363)

[6. むすび 21](#_Toc90999364)

[**謝辞**](#_Toc90999365)

[**参考文献**](#_Toc90999366)

[付録](#_Toc90999367)

# **まえがき**

## はじめに

2020年以降小学校，中学校，高等学校にてプログラミング教育必修の全面実施することが文部科学省により決定された1)．文部科学省の発表によると，プログラミング教育を必修とする目的は，現代社会で普遍的に求められる力としての論理的思考などを育むことである2)．

プログラミング初級者を対象として論理的思考を鍛えるために，ブロックプログラミングと呼ばれる，プログラミングの導入に利用されるシステムが存在する．これによりブロックを組み立てることで積み木の様にプログラミングを行える．一方，実際に運用されるシステム開発などでは，プログラミング言語を用いたコーディングが必要とされる．プログラミングの初級教育から中高等教育にかけて，論理的思考力からコーディング力の養成に円滑に移行できるような教育支援も考えていく必要がある．

そのようなプログラミング教育支援を行うために，ブロックプログラミングによる論理的思考力の訓練とともに，生成したブロックに対応付くソースコードを穴埋め問題として提供することでコーディング力を養成する教育方法が検討できる．また，このようにソースコードを穴埋め問題として出題する場合，問題の難易度に応じて穴埋め問題を自動で生成し，穴埋め箇所を選択問題として出題することで変化に富んだ問題を生成できると考えられる．しかし，ソースコードの穴埋め選択問題を自動で生成する技術はこれまでも研究がされているものの，今なお重要な課題となっている3),4)．

本研究では，このような課題を解決するために，ブロックプログラミングからコーディング用の練習問題を自動生成することに着目し，ブロックプログラミングと連携したソースコードの穴埋め問題生成システムを提案する．

提案システムでは，Blocklyによって利用できる予約語と不等号や演算式記号をカテゴリーグループとして分けている．そして，ソースコード単語ごとにふりわけて，一つずつカテゴリーグループ内の単語と比較し，一致した単語と箇所を保存する．その中から乱数的にいくつかの箇所をピックアップして，穴埋め問題の生成をおこなっている．

提案システムを用いて，教師は問題文とブロックプログラミングによって生成するXMLデータなどの問題ファイルを提案システムに登録することで，学習者にブロックプログラミング問題とそれに対応するソースコードの穴埋め選択問題を出題することができる．また，学習者は，提案システム上で出題されたブロックプログラミング問題を解くことで論理的思考力を身に着け，さらにソースコードの穴埋め選択問題を解くことでコーディング力を養成することができる．

提案システムの対象言語はJavaScriptであるが，教師は問題ファイルにてJavaScriptのコードではなく，XMLコードで管理する，これはBlocklyを用いることでブロックプログラミングの完成系を用意することができるためである．これにより，提示された問題に対してブロックがわかりやすくなる．また，BlocklyによってXMLコードからJavaScriptコードに変換することが可能であり，XMLコードのみを管理すれば，ソースコードの問題生成とブロックプログラミングによる補助を同時に行うことができる．

提案システムでは，学習者が選んだ選択肢によって完成されるJavaScriptコードを実行することができる．これにより，その選択肢を選んだ場合のプログラムの動作から，「実行結果が正しい場合」，「問題文に沿わない場合」，「エラーが発生して動作しない場合」など，実際に体験をすることができる．穴埋め問題を採点し，正解・不正解だけでなく実際に動作を確認できることで，学習者のプログラミング言語への学習効率の向上を図る．

実験では，提案システムの問題自動生成機能によって生成される穴埋め選択問題が，ブロックプログラミングからコーディング学習のために適切に問題を生成可能であるかを難易度や出題基準の観点から評価することで提案システムの実現可能性を検証する．

## 本論文の構成

　本論文は全６章で構成されている．本章では研究の背景や動機，目的について述べる．第２章では関連研究について述べる．第３章では本研究の提案方式とその特徴について述べる．第４章では提案方式を構築した本システムの設計について述べる．第５章では実験について述べる．第６章では結論と今後の展望について述べる．

# 関連研究

　本章では，関連研究について述べる．なお，論理的思考に関する研究を2.1節，アルゴリズム的思考に関する研究を2.2節，問題自動生成システムに関する研究を2.3節に示す．



## 論理的思考に関する研究

　言語論理教育を行う必要性を主張してきた研究者である井上は “言語教育でも、例えば「論理的思考力を高める」というような表現がよく使われますが、それはいろいろな意味に使われています。しかしそれらを大きく分けると、次の三つに分類することができます。すなわち、論理的思考とは、

（１）形式論理学の諸規則にかなった推論のこと（狭義）

（２）筋道の通った思考、つまりある文章や話が論証の形式（前提一結論、また主張一理由という骨組み）を整えていること

（３）広く直観やイメージによる思考に対して分析、総合、比較、関係づけなどの「概念的」思考一般のこと〈広義）

以上のうち、（１）と（２）とは結びついており、分ける必要はないようですが、現実には、（１）についてはあまり自覚されず、（２）のように論証の型式にレイアウトされていればそれでよしとする（その段階でとどまる）ことが多いのです。〔中略〕論や主張の根拠となっていることがらについて、それがその主張の必然性を裏付けているかどうか、前提と結論の間に論理的必然性があるかどうかというところまではあまり論議されないようです。“3)と定義している．

　本研究では論理的思考とは先に定義されている中でも広義である（２），（３）の文脈でとらえることとする．

## アルゴリズム的思考に関する研究

　 2.1節の論理的思考ではより広義な意味をとったが，プログラム教育において狭義的に論理的思考について考えるとアルゴリズム的思考と呼ばれる，“実行したい処理をアルゴリズムの形に分解・整理する思考” 5)が考えられる．プログラミング教育の重要性が高まって生きているが，プログラミング教育における現場ではプログラムの構文など文法知識を有していても実際にプログラムを記述できない学習者が存在する．アルゴリズム的思考を養うことで，プログラムを学習する場合に，十分なスキルを身に着けることができる．

## 問題自動生成システムに関する研究

　文献６）において，初学者の場合には，キーワードやプログラム全体の構造を繰り返し学習により覚えることが肝要とし，ソースコードを基にして穴埋め式のドリルを生成している．このシステムではキーワードや識別子を抽出し，乱数によって選び出して空所にすることでドリルを生成している．

　文献７）において，教育現場において演習問題の不足に対して，演習問題の自動生成の研究の多くではXMLタグを付与するなど事前準備作業があり，作問者にかかる負担が大きいことから，過去問題をリソースとして問題自動生成システムについて作問負担と問題の分析結果について検証している．

　文献８）において，プログラミング入門者を対象とした教育において，アルゴリズムの理解や文法の理解が必要不可欠として，サンプルプログラムを読解・あるいは記述の仕方を真似することが初期の学習の基本となる．そこで，ソースプログラムから学習用の空欄補充問題を自動生成することについてその手法を提案している．

# **提案システム**

　本研究では，ブロックプログラミングからコーディング用の練習問題を自動生成することに着目し，ブロックプログラミングと連携したソースコードの穴埋め問題生成システムを構築する．なお，提案方式の概要を3.1節，提案方式の構成を3.2節，提案方式を利用した問題生成方法を3.3節，提案方式を利用した学習方法を3.4節に示す．



## 提案方式の概要

　本研究では，問題に利用するJavaScriptコードを生成するために，事前にBlocklyのブロックプログラミングによって生成できるXMLコードと問題文と実行結果の例を用意する必要がある．これを問題ファイルと呼ぶ．これらを用意した後に以下のステップによって用意される選択問題を繰り返し解くことで，学習効率の向上を図る．提案方式の概要図を図3.1に示す．



図 3.1　提案内容の概要図

## 提案方式の構成

　本研究では穴埋め問題の生成のために問題ファイルと呼ぶファイルを作成する．本節では，問題ファイルの作成，問題ファイルから穴埋め問題の生成，解答，採点までの構成を説明する．



### 問題ファイル

　問題ファイルとは穴埋め問題の生成に利用するファイルのことで，「プログラミング言語のコード生成のためにタグ付けされたXMLデータ」，「問題文」，「問題の実行結果」，「評価のためのタグ」の5つのデータが入力されたファイルのことをいう．

### 問題のXMLコードの用意

　問題ファイルに利用する「プログラミング言語のコード生成のためにタグ付けされたXMLデータ」の作成は，本研究ではブロックプログラミングによって支援される．問題文に沿ってブロックプログラミングを書くことにより，自動でXMLへと変換されたデータを問題ファイルに用いる．



### 問題を自動生成する

本システムでは，問題の例となる完成されたコードから，穴埋め問題を自動で生成している．問題の生成数はＮ個とする．またワードがマッチした数をＭとする．手順として以下のステップによって生成される．

Step-1:コードを単語で区切り，既定のワードとマッチするかを判断し，ヒットした場所とワードを保存するリストを作成する．

Step-2:問題の生成数の数Ｎとワードがヒットした数Ｍを比較する．

Step-3:Ｎ>Ｍだった場合はＭを保存して次のステップへ進む．Ｎ<=Ｍだった場合にはランダムでＮ個を抜き出し，次のステップへ進む．

Step-4:問題を生成することが決定したワードのヒットした場所をセレクトボックスに置換する．

Step-5:セレクトボックスの内容に選択式問題の解答を挿入し，オプションとして，正答の場合にはans，それ以外には数字を設定する．

### 選択問題の選択肢の決定

　問題の自動生成にて作成される選択肢の内容は同じカテゴリーからなる場合と，異なるカテゴリーを含む場合を設計した．同じカテゴリーとは，繰り返し命令カテゴリーでfor, while, doなど同じようなタイミングで利用されるキーワードのこととする．同じカテゴリーのみから問題を生成することでプログラム構文への理解が浅い学習者の学習支援を行う．異なるカテゴリーとは，本システムで利用可能な予約語と四則演算と不等号のすべてのことで，同一でないキーワードであればすべてが選択肢として利用される．異なるカテゴリーを含み選択問題が生成された場合，学習者はプログラムの前後をより理解して解答する必要がある．

### 解答の正誤を採点

　本項までに作成された選択問題にて問題の正誤を判定は以下のステップによって行う．

Step-1:問題数を判別する

Step-2:セレクトボックスの内容を読み取り，正解不正解を判断．

Step-3:正解だった場合正解数をカウントし，その問題番号を保存する．

Step-4:不正解だった場合は，その問題番号を保存する．

Step-5:正解数と正解した問題番号，不正解の問題番号を表示

### 誤った解答の解説の表示

問題の規定の正解である「コードの実行が可能であり，正しい実行結果が表示される選択肢」以外に「コードとして実行が出来，実行結果から問題の解答として正しい選択肢」と「コードとしては実行できる状態だが，問題の解答としては正しくない選択肢」，「エラーとなり実行が出来なくなる選択肢」が存在する．本研究のシステムでは，問題は１単語ごとに生成されることから，「同一の内容を異なる書き方をした結果解答として誤りとなる」ことは想定しない．また，選択肢問題のため「スペルミスによる誤り」は存在しない．つまり，解答の誤りは予約語の間違いか条件式の間違いとなる．

そのため，どのように誤ったのかを選択から判断し，選んだ選択肢はどういった場合に利用するか解説を表示することで，学習効率の向上につなげる．以下のステップで実行する．

Step-1:不正解だった問題を確認する．

Step-2:選ばれた選択肢を確認する．

Step-3:その選択肢がどのような場合に利用されるものか解説を表示する．

## 問題の作成方法（教師視点）

　学習教材となる，問題を生成する際に元となる「問題ファイル」を作成し，学習者に提示することで教師はソースコードの学習支援を行う．問題ファイルの作成方法については以下のステップで行う．

Step-1:提示する問題文とその解答を考える．

Step-2:提示する問題に対応するブロックプログラミングを完成させる．

Step-3:ブロックプログラミングよりXMLコードを作成する．

Step-4:XMLコード，問題文，解答（実行結果）を問題ファイルとして記述する．

## 学習方法（学習者視点）

　学習者は，教師により用意された問題ファイルを利用し，以下のステップによって学習を行う．

Step-1:教師より提示された問題ファイルを提案システムにて開く

Step-2:システムによって表示される問題文を確認する

Step-3:自動生成された問題を確認する

Step-4:穴埋め選択問題を解答する

Step-5:自動採点を確認する

Step-6:採点にて間違いがある場合Step-7へ，問題がない場合Step-8へ

Step-7:同じ問題ファイルから新たに問題を自動生成し，Step-3に進む

Step-8:異なる問題ファイルを選び，Step-2に進む

# **実装システム**

本章では提案方式を構築した本システムの設計について述べる．なお，実装システムの概要を4.1節，実装システムの実行環境を4.2節，実装システムの機能の説明を4.3節，穴埋め問題の生成方法を4.4節，問題の採点方法を4.5節，Blocklyの公式との変更点を4.6節に示す．



## 実装システムの概要

　実装システムの構築環境を表4.1に示す．また，本章ではプログラムのコード片をいくつか示している．それらの簡易な説明を表に示す．

表 4.1 実装システムの構築環境

|  |  |  |
| --- | --- | --- |
| 利用用途 | 名称 | バージョン・型番 |
| OS | Windows 10 | 19042.1348 |
| 開発言語 | JavaScript |  |
| ブラウザ | Google chrome | 96.0.4664.110 |
| パッケージ管理 | npm | 6.14.13 |
| JavaScript実行環境 | Node.js | 16.13.1 |
| VPL | Blockly | 5.20210325.0 |

表 4.2 本章のコードの説明

|  |  |  |
| --- | --- | --- |
| 表番号 | 表題 | 説明 |
| 10 | コードジェネレート機能 | コードジェネレート機能 |
| 11 | ブロックプログラミングからコードへのリアルタイム生成 | コードへのリアルタイム生成 |
| 12 | indent() | インデント |
| 13 | jsrunout() | コードの実行 |
| 18 | answer2() | 解答の採点 |
| 19 | check2() | 解答の比較 |
| 20 | numq() | 解答の比較 |
| 21 | gotou() | 解答の解説 |

## 実装システムの実行環境構築

　4.2節では，実装システムの実行環境を構築するための方法を示す．



### 本研究のプログラムのダウンロード

本研究で利用するプログラムはすべてGitHubにアップロードされている．以下のGitHubのリンクよりprojectの内容をすべてダウンロードし，次項にて示すNode.jsをインストールすることで本研究のシステムを利用することが可能である．

表 4.3 GitHubのリンク

|  |
| --- |
| https://github.com/kait-takanolab/1821121-shimaoka-thesis/tree/main/project |

### Node.js

　Node.jsとはサーバ上で動作するJavaScript環境．ほとんどのJavaScriptと異なり，ウェブブラウザ上でなく，サーバサイドでの動作となる．本研究ではサーバ上での実行と，npmの利用のためにNode.jsを利用する．npmとはパッケージ管理システムの一種で，Node package managerの意である．Node.jsを用いたシステムの管理のために作成され，Node.jsのツールやパッケージのインストールや管理はもちろん，それらパッケージを利用することも可能である．また，Node.jsのWebアプリケーションフレームワークであるexpress.jsを用いて環境を構築した．

推奨版のダウンロード・インストールをする．下記の表4.4で示すリンクにてダウンロードした後，特別な理由がなければNextと続けることでインストールが可能である．

表 4.4　node.jsのダウンロードリンク

|  |
| --- |
| https://nodejs.org/ja/ |

npmを用いたパッケージのインストールにはnpm install “パッケージ名称”というコマンドでインストールが可能である．上記のgithubリンクより本研究のシステムを全てダウンロードしている場合には必要なパッケージは同時にダウンロードされるが，新規に環境構築時にはパッケージのインストールが必要である．以下の表4.5に示す．

表 4.5 npmによるパッケージのインストールコマンド

|  |
| --- |
| npm install “package” |

### サーバの実行コマンド

　4.2.1項，4.2.2項にて本研究のシステムとNode.jsのインストールが完了した後に，projectの位置で下記のコマンドを実行することでサーバが起動される．その後Webブラウザにてlocalhost:3000にアクセスすることで本システムのindex.htmlにアクセスすることができる．以下の表4.6に示す．

表 4.6　サーバの実行コマンド

|  |
| --- |
| npm start |

## 実装システムの機能説明

　本節では，実装システムに利用する機能の説明と，コードを示す．



### Visual Programming Language

ビジュアルプログラミング言語（VPL）とは，プログラムを視覚的なオブジェクトの組み合わせによって作成するプログラミング言語である．ブロックプログラミング，グラフィカルプログラミング言語とも言う．VPLとは総称であり，どのように視覚的に示すかにはいろんな種類があるが，本研究ではBlocklyを利用する．本研究では一部改変したデータを用いるため，改変後のデータを本研究のproject内にBlocklyのファイルも同封されているが，Blocklyの公式の状態のデータを扱うためには，「google/Blockly」のgithubにてファイルをインストールした後，npmによるインストールによって可能となる．以下の表4.7，4.8に示す．

表 4.7　 google/blockly githubリンク

|  |
| --- |
| https://github.com/google/blockly |

表 4.8　 blockly install

|  |
| --- |
| npm install blockly |

### コードジェネレート機能

コードジェネレート機能とは，Blocklyによって提供されるモジュールの一つで，組み合わせたブロックからコードを生成する機能を提供している．Blockly公式より提供されるコードジェネレート機能はJavaScript，Python，PHP，Lua，Dartの５つのプログラミング言語に対応したものが存在し，関連研究にはC言語など他の言語に対応させたものも存在する．本研究ではJavaScriptに関連した選択式問題の生成・評価をしている．

コードジェネレート機能を利用するためには，Blocklyを用いるためのスクリプトをコード内に含める必要がある，以下の表4.9に示す．

表 4.9 コードジェネレート機能利用準備

|  |
| --- |
| <script src="blockly\_compressed.js"></script>  <script src="javascript\_compressed.js"></script> |

コードを生成する際には下記のコードを呼び出すことで可能となる．以下の表4.10に示す．

表 4.10 コードジェネレート機能

|  |
| --- |
| var code = Blockly.JavaScript.workspaceToCode(workspace); |

リアルタイムでコードを表示する際には下記の様に組み合わせることで可能である．以下の表4.11に示す．

表 4.11 ブロックプログラミングからコードへのリアルタイム生成

|  |
| --- |
| function myUpdateFunction(event) {  var code = Blockly.JavaScript.workspaceToCode(workspace);  document.getElementById('textarea').value = code;  }  workspace.addChangeListener(myUpdateFunction); |

### インデント

　通常プログラミングの際には，インデントと呼ばれるコードの字下げを行う．これは主にコードの視認性を向上する効果が期待されており，どの行が同じブロックに含まれているかが確認しやすくなることで学習支援に役立つと考えられる．Blocklyによるコードジェネレート機能によって生成されたコードをHTMLで表示する際に，インデントがない状態での表示となってしまうため，インデントを付与する関数を作成している．以下の表4.12に示す．

表 4.12　indent()

|  |
| --- |
| function indent(code) {  //インデントに利用する変数  var count = "0";  var ind = "&nbsp;&nbsp;&nbsp;&nbsp;";  var inc = '{';  var dec = '}';  var end = '<br>';  //文字列を読み込み、改行コードで配列に保存  var text = code;  var text2 = text.split(/\n|<br>/);    //行数ループ  for (let i = 0; i < text2.length; i++) {  //コンマを一度置換しておき，あとでもう一度変換する．  var result = text2[i].indexOf(',');  do{  text2[i] = text2[i].replace(",", "!exchange")  result = text2[i].indexOf(',', result)  }while(result != -1)  //インデントの数を減少  if(text2[i].includes(dec)){  count--;  // console.log(count);  }    //個数分インデント  if (count > "0") {  for(let j = 0; j < count; j++){  text2[i] = ind + text2[i];  }  }    //インデントの数を増加  if (text2[i].includes(inc)) {  count++;  // console.log(count);  }    if(i != text2.length -1){  text2[i] = text2[i] + end;  }  }  code = text2.join();  code = code.replace(/,/g, "");  code = code.replace(/!exchange/g, ",");  return code;  } |

### JavaScriptコードの実行

　4.3.2項にて作成されたコードは無限ループが発生していないかを判断した後にコードを実行する関数を呼び出す．Blockly自体に無限ループを判断する機能はないため，実行時のループ回数が規定回数を超えた時点で無限ループと判断し，Infinite loopと表示する．また，実行時にエラーが発生する場合にはエラーを表示する．実際の実装を以下の表4.13に示す．

表 4.13 jsrunout()

|  |
| --- |
| function jsrunout() {  // Generate JavaScript code and run it.  window.LoopTrap = 10000;  Blockly.JavaScript.INFINITE\_LOOP\_TRAP =  'if (--window.LoopTrap == 0) throw "Infinite loop.";\n';  var code = document.getElementById("output").innerText;  Blockly.JavaScript.INFINITE\_LOOP\_TRAP = null;  try {  execution(code);  } catch (e) {  document.getElementById('kaitou').textContent = e;  }  } |

test.htmlでは選択式問題からユーザが選んだ選択肢によって完成したJavaScriptコードを実行することが可能である．これはセレクトボックスによって作成された穴埋め問題の選ばれた解答を置換し，実際のコードとなる文字列を作成した後に実行している．以下の表4.14に示す．

表 4.14 jsruntst()

|  |
| --- |
| function jsrunout() {  // Generate JavaScript code and run it.  window.LoopTrap = 10000;  Blockly.JavaScript.INFINITE\_LOOP\_TRAP =  'if (--window.LoopTrap == 0) throw "Infinite loop.";\n';  var code = document.getElementById("output").innerText;  Blockly.JavaScript.INFINITE\_LOOP\_TRAP = null;  try {  execution(code);  } catch (e) {  document.getElementById('kaitou').textContent = e;  }  } |

作成されたJavaScriptコードを実行し結果を確認するために，本システムではFunction関数を用いる．実際には下記の様に実装している．JavaScriptコードを引数とする．以下の表4.15に示す．

表 4.15　execution()

|  |
| --- |
| function execution(code){  kaitouran();  Function(code)();  } |

実行結果を表示する欄を一度初期化するためにkaitouran()という関数を呼び出している．以下の表4.16に示す．

表 4.16　kaitouran(),

|  |
| --- |
| function kaitouran(){  document.getElementById("kaitou").textContent = "";  } |

## 実装システムの選択式問題の実装

　本節では，実装システムの穴埋め問題の実装の流れとコードを示す．



### 問題ファイルの生成

本研究において，問題ファイルと呼ぶ問題を生成するためのファイルを用意する必要がある．問題ファイルの内容は「XMLコード」，「問題文」，「実行結果の例」，「採点時に利用するタグ」の４つである．データの利用方法はそれぞれ「問題となるJavaScriptコードの生成」，「問題文の表示」，「実行結果の評価」.「採点」のために利用される．これら４つのデータを用意し，「@@」によって区切ることで問題ファイルとして利用できるファイルとなる．

XMLコードの用意は次項にて示す．問題文，実行結果の例は教師が事前に考える必要がある．ただし，実際には実行結果に関しては，XMLコードを用意した後にBlocklyによって生成したJavaScriptコードの実行結果を利用して問題ないと考えられる．採点時に利用するタグについては，問題にて想定される解答が複数示される場合に，「num」というタグを付与し，解答が一つになる場合にはタグを何も付与しないことで採点時に解答数を認識して自動で採点が可能となる．

### XMLコードの生成

　問題ファイルの中で，ソースコードに相当する箇所を用意する方法について示す．

　「xml.html」というファイルが「project/public/」に存在する．これを利用することでXMLコードを生成することができる．以下に示すステップでXMLコードを生成し，問題ファイルに保存する．

Step-1:Blocklyを用いて問題文にそったブロックプログラムを作成する

Step-2:BlocklyによってブロックプログラムからXMＬコードに変換



### 穴埋め問題の生成

本システムでは，問題の例となる完成されたJavaScriptコードから，穴埋め問題を自動で生成している．手順として以下のステップによって生成される．

Step-1:コードを単語で区切り，既定のワードとマッチするかを判断し，ヒットした場所とワードを保存するリストを作成する．

Step-2:問題の生成数の数とワードがヒットした数を比べ，問題数を決定する．

Step-3:既定の数よりも多くワードがヒットしていた場合，規定数分だけランダムで数字を作成し，以下のステップに進む．

Step-4:リストより，ヒットした場所をセレクトボックスに置換する．

Step-5:セレクトボックスの内容に選択式問題の解答を挿入し，オプションとして，正答の場合にはans，それ以外には数字を設定する．

問題の生成数は簡単，普通，難しいまで３種類で簡単，普通は同じカテゴリーのキーワードによるそれぞれ3，6個の問題を生成する．難しいでは異なるカテゴリーのキーワードによる３つの選択肢からなる３個の問題を生成する．通常の難易度は簡単で作成され，ブラウザに表示されるボタンにより難易度の変更・問題の再生成を行う．

同じカテゴリーグループは次の表4.17のとおりである．

表 4.17 カテゴリーグループ

|  |  |
| --- | --- |
| カテゴリー番号 | メンバー |
| グループ１ | '<'，'>'，'<='，'>='，'=='，'!=' |
| グループ２ | "+"，"-"，"\*"，"/"，"%" |
| グループ３ | 'for', 'while', 'do' |
| グループ４ | 'if', 'else', 'switch' |
| グループ５ | 'break'，'continue' |
| グループ６ | '?'，':' |
| グループ７ | 'sqrt'，'abs'，'log'，'exp'，'pow' |
| グループ８ | 'sin'，'cos'，'tan' |
| グループ９ | 'asin'，'acos'，'atan'，'atan2' |
| グループ１０ | 'PI'，'E' |
| グループ１１ | 'round'，'ceil'，'floor'，'random' |
| グループ１２ | 'length'，'indexOf'，'charAt'，'slice' |
| グループ１３ | 'toUpperCase'，'toLowerCase'，'textToTitleCase'] |
| グループ１４ | 'trim'，'replace'，'split' |
| グループ１５ | 'textCount'，'textReplace' |
| グループ１６ | 'alert'，'prompt'，'print' |
| グループ１７ | 'sort'，'revserse'，'length' |
| グループ１８ | 'funtion'，'return' |
| グループ１９ | '='，'=='，'+='，'-=' |

## 実装システムの選択式問題の実装

　本節では，穴埋め問題の採点に関連する実装について示す．



### 穴埋め問題の解答の評価

　穴埋め問題の解答から，解答者が何問正解したのかを判断する．正解数と，正解した問題番号を表示する．以下の表4.18に示す．

表 4.18　 answer2()

|  |
| --- |
| function answer2(){  //正解数をカウントする  var count = 0;  //ループ回数（問題数）をカウント  var loop = 0;  //正解した問題番号を保存する  var kotae =[];  //選ばれた解答を保存したい  var ans = [];  //セレクトボックスがいくつあるのかをカウント  //セレクトボックスがなかったらBreak  while(1){  var select = document.querySelector(`select.sele${loop}`);  if(select != null){  //セレクトボックスのValueを入手（正解ならans, それ以外だと数字が返る）  idx = select.selectedIndex;  if(select.options[idx].value == "ans"){  //何問正解しているかを合計  kotae[count] = loop+1 +"問目";  count++;  }else{  ans[loop] = [loop+1,select.options[idx].text];  }  }else{  break;  }  loop++;  } |

### 実行結果と解答例との比較

　実行結果から，問題の解答例と比較し，正誤を評価する．複数文字列を表示する問題の場合には，何個目の表示が間違っているのかも表示する．また，問題ファイルに実行結果の比較のためのタグを設定し，そのタグによって解答の個数などを確認し，評価している．表4.19に示すcheck2という関数により基本は実行結果を比較し，タグの内容次第でnumqという関数による評価を行う．numqという関数は解答が複数ある場合に実行される．表4.20に示す．

表 4.19　 check2()

|  |
| --- |
| function check2(){  var code1 = document.getElementById('kaitou').textContent;  var code2 = strcode[2];  var hyouka = document.getElementById("answer").textContent;  if(code1 == code2){  hyouka = "実行結果は正しい";  }else{  if(strcode[3]=="num"){  hyouka = numq(code1,code2);  }else{  var hyouka = "実行結果は間違い\n";  }  }  return hyouka;  } |

表 4.20　 numq()

|  |
| --- |
| function numq(code1,code2){  hyouka = "";  c1 = code1.split(' ');  c2 = code2.split(' ');  console.log(code1,code2,c1,c2);  if(c1.length != c2.length){  hyouka += "ループに間違いがあります、"  if(c1.length > c2.length){  hyouka += "回答が多いです\n"  }else{  hyouka += "回答が少ないです\n"  }  }  for(var i=0; i < c2.length; i++){  if(c1[i] != c2[i]){  hyouka += i+1 + "個目は誤り\n"  }  }  if(c2.length < c1.length){  hyouka += c2.length + "以降は誤り\n"  }  if(hyouka != ""){  return hyouka;  }else{  return "実行結果は正しい";  }  } |

### 誤答の場合の解説の表示

問題の規定の正解である「コードの実行が可能であり，正しい実行結果が表示される選択肢」以外に「コードとして実行が出来，実行結果から問題の解答として正しい選択肢」と「コードとしては実行できる状態だが，問題の解答としては正しくない選択肢」，「エラーとなり実行が出来なくなる選択肢」が存在する．本研究のシステムでは，問題は１単語ごとに生成されることから，「同一の内容を異なる書き方をした結果解答として誤りとなる」ことは想定しない．また，選択肢問題のため「スペルミスによる誤り」は存在しない．これらのことから，解答の誤りは予約語の間違いか条件式の間違いとなる．どのように誤ったのかを選択された解答から判断し，選んだ解答はどういった場合に利用するか解説を表示することで，学習効率の向上につなげる．以下のステップで実行する．また以下の表4.21にコードの省略版を示す．

Step-1:不正解だった問題を確認する．

Step-2:選ばれた選択肢を確認する．

Step-3:その選択肢がどのような場合に利用されるものか解説を表示する．

表 4.21 gotou()の省略版

|  |
| --- |
| function gotou(kaisetu){  var hyouka ="";  //kaisetu[].lengthの回数ループする  if(kaisetu.length != 0){  hyouka += "間違いの解説：\n"  }  for(var i = 0; i < kaisetu.length; i++){  if(kaisetu[i] != null){  hyouka += kaisetu[i][0];  switch(kaisetu[i][1]){  case "<":  hyouka += "問目：a < b の場合、bの値を含まないbより小さい値をaがとっている場合に使うよ！\n";  break;  default:  hyouka += "問目:は正解"  break;  }  }  }  return hyouka;  } |

## Blocklyの変更点

Blocklyを公式の状態のまま利用すると本システムでは扱いづらい部分が存在したため，一部を改変し利用している．変更箇所はblocks\_compressed.js , javascript\_compressed.js, ja,jsの３ファイルで，文字列出力とfor文の利用のために行った．



### text\_printを変更

　text\_printとはBlocklyによって文字列のアウトプットを扱うために利用されるもので，初期設定では，文字列をalert関数にて表示するものであった．これをテキストエリアに実行結果を表示するように変更することで，実行結果画面に残るため，学習者がみやすいと考えた．また，文字列がDOM上に残るために，解答例との比較の実装の容易さも含めこのような変更を行った．

　しかし，JavaScriptのコードとしてはalertなどが標準出力として多く利用され，テキストエリアのIDを指定しての内容の書き換えを行うコードよりも，alertという関数を覚えた方がいいという考えもアンケートでは集まった．これはJavaScriptというHTMLなどフロントエンドのブラウザで表示するものと同時に利用することが考えられるプログラミング言語だから起こる問題である．Pythonなどバックエンドでの動作における標準入出力の場合ではprintなどほぼ一意に決定する．

表 4.22 JavaScriptの標準出力を変更している箇所

|  |
| --- |
| Blockly.JavaScript['text\_print'] = function(block) {  // Print statement.  var msg = Blockly.JavaScript.valueToCode(block, 'TEXT',  Blockly.JavaScript.ORDER\_NONE) || '\'\'';  return "document.getElementById('kaitou').textContent += " + msg + " + \" \";\n";  }; |

### for文を変更

　Blocklyの実装では最初は”<=”の形で以上または以下という書き方でのfor文のみを考慮されていた．この書き方だけでもプログラムは作成できるが，本研究では学習支援を目的としているため，”<”という「より多い」や「未満」という形で扱えるように変更した．下記に変更を示す．また，Blocklyの表示のためにメッセージを管理するファイルも同様に同期した．

表 4.23 Blocklyのforループの表示と実行の変更

|  |
| --- |
| {type:"controls\_for1",message0:"%{BKY\_CONTROLS\_FOR\_TITLE1}",args0:[{type:"field\_variable",name:"VAR",variable:null},{type:"input\_value",name:"FROM",check:"Number",align:"RIGHT"},{type:"input\_value",name:"TO",check:"Number",align:"RIGHT"},{type:"input\_value",name:"BY",  check:"Number",align:"RIGHT"}],message1:"%{BKY\_CONTROLS\_REPEAT\_INPUT\_DO} %1",args1:[{type:"input\_statement",name:"DO"}],inputsInline:!0,previousStatement:null,nextStatement:null,style:"loop\_blocks",helpUrl:"%{BKY\_CONTROLS\_FOR\_HELPURL}",extensions:["contextMenu\_newGetVariableBlock","controls\_for\_tooltip"]},  {type:"controls\_for2",message0:"%{BKY\_CONTROLS\_FOR\_TITLE2}",args0:[{type:"field\_variable",name:"VAR",variable:null},{type:"input\_value",name:"FROM",check:"Number",align:"RIGHT"},{type:"input\_value",name:"TO",check:"Number",align:"RIGHT"},{type:"input\_value",name:"BY",  check:"Number",align:"RIGHT"}],message1:"%{BKY\_CONTROLS\_REPEAT\_INPUT\_DO} %1",args1:[{type:"input\_statement",name:"DO"}],inputsInline:!0,previousStatement:null,nextStatement:null,style:"loop\_blocks",helpUrl:"%{BKY\_CONTROLS\_FOR\_HELPURL}",extensions:["contextMenu\_newGetVariableBlock","controls\_for\_tooltip"]}, |

表 4.24 Javascriptのコードを生成する場合の上記の変更の同期

|  |
| --- |
| Blockly.JavaScript.controls\_for1=function(a){var b=Blockly.JavaScript.variableDB\_.getName(a.getFieldValue("VAR"),Blockly.VARIABLE\_CATEGORY\_NAME),c=Blockly.JavaScript.valueToCode(a,"FROM",Blockly.JavaScript.ORDER\_ASSIGNMENT)||"0",d=Blockly.JavaScript.valueToCode(a,"TO",Blockly.JavaScript.ORDER\_ASSIGNMENT)||"0",e=Blockly.JavaScript.valueToCode(a,"BY",Blockly.JavaScript.ORDER\_ASSIGNMENT)||"1",f=Blockly.JavaScript.statementToCode(a,"DO");f=Blockly.JavaScript.addLoopTrap(f,a);if(Blockly.isNumber(c)&&Blockly.isNumber(d)&&  Blockly.isNumber(e)){var g=Number(c)<=Number(d);a="for ("+b+" = "+c+"; "+b+(g?" <= ":" >= ")+d+"; "+b;b=Math.abs(Number(e));a=(1==b?a+(g?"++":"--"):a+((g?" += ":" -= ")+b))+(") {\n"+f+"}\n")}else a="",g=c,c.match(/^\w+$/)||Blockly.isNumber(c)||(g=Blockly.JavaScript.variableDB\_.getDistinctName(b+"\_start",Blockly.VARIABLE\_CATEGORY\_NAME),a+="var "+g+" = "+c+";\n"),c=d,d.match(/^\w+$/)||Blockly.isNumber(d)||(c=Blockly.JavaScript.variableDB\_.getDistinctName(b+"\_end",Blockly.VARIABLE\_CATEGORY\_NAME),a+=  "var "+c+" = "+d+";\n"),d=Blockly.JavaScript.variableDB\_.getDistinctName(b+"\_inc",Blockly.VARIABLE\_CATEGORY\_NAME),a+="var "+d+" = ",a=Blockly.isNumber(e)?a+(Math.abs(e)+";\n"):a+("Math.abs("+e+");\n"),a=a+("if ("+g+" > "+c+") {\n")+(Blockly.JavaScript.INDENT+d+" = -"+d+";\n"),a+="}\n",a+="for ("+b+" = "+g+"; "+d+" >= 0 ? "+b+" <= "+c+" : "+b+" >= "+c+"; "+b+" += "+d+") {\n"+f+"}\n";return a};  Blockly.JavaScript.controls\_for2=function(a){var b=Blockly.JavaScript.variableDB\_.getName(a.getFieldValue("VAR"),Blockly.VARIABLE\_CATEGORY\_NAME),c=Blockly.JavaScript.valueToCode(a,"FROM",Blockly.JavaScript.ORDER\_ASSIGNMENT)||"0",d=Blockly.JavaScript.valueToCode(a,"TO",Blockly.JavaScript.ORDER\_ASSIGNMENT)||"0",e=Blockly.JavaScript.valueToCode(a,"BY",Blockly.JavaScript.ORDER\_ASSIGNMENT)||"1",f=Blockly.JavaScript.statementToCode(a,"DO");f=Blockly.JavaScript.addLoopTrap(f,a);if(Blockly.isNumber(c)&&Blockly.isNumber(d)&&  Blockly.isNumber(e)){var g=Number(c)<=Number(d);a="for ("+b+" = "+c+"; "+b+(g?" < ":" > ")+d+"; "+b;b=Math.abs(Number(e));a=(1==b?a+(g?"++":"--"):a+((g?" += ":" -= ")+b))+(") {\n"+f+"}\n")}else a="",g=c,c.match(/^\w+$/)||Blockly.isNumber(c)||(g=Blockly.JavaScript.variableDB\_.getDistinctName(b+"\_start",Blockly.VARIABLE\_CATEGORY\_NAME),a+="var "+g+" = "+c+";\n"),c=d,d.match(/^\w+$/)||Blockly.isNumber(d)||(c=Blockly.JavaScript.variableDB\_.getDistinctName(b+"\_end",Blockly.VARIABLE\_CATEGORY\_NAME),a+=  "var "+c+" = "+d+";\n"),d=Blockly.JavaScript.variableDB\_.getDistinctName(b+"\_inc",Blockly.VARIABLE\_CATEGORY\_NAME),a+="var "+d+" = ",a=Blockly.isNumber(e)?a+(Math.abs(e)+";\n"):a+("Math.abs("+e+");\n"),a=a+("if ("+g+" > "+c+") {\n")+(Blockly.JavaScript.INDENT+d+" = -"+d+";\n"),a+="}\n",a+="for ("+b+" = "+g+"; "+d+" >= 0 ? "+b+" < "+c+" : "+b+" > "+c+"; "+b+" += "+d+") {\n"+f+"}\n";return a}; |

# **実験**

　本章では，提案手法を適用した実験システムを用いて，提案方式を検証する実験について述べる．



## 実験目的

本システムの問題自動生成機能によって生成される穴埋め選択問題は学習のために適切に問題を生成することができているのか，複数の問題ファイルと難易度における実際の出題内容をいくつかの判断基準ごとに評価する．

## 実験環境

　本実験では，自動問題生成のために利用される問題ファイルを複数用意した．それぞれの問題ファイルを用いて問題生成を行い，難易度ごとの選択肢を蒐集した．問題は難易度毎に「簡単」，「難しい」では３問ずつ，「普通」では６問生成され，１つの問題ファイルにつき問題を12個の問題が生成される．また，問題ファイルは「project/public/question/」に用意した，4~8までの５つの問題ファイルを用いて実験を行った．実際の問題数は以下の表5.1に示す．

表 5.1 蒐集した問題数と選択肢

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 問題ファイル数 | 生成された総問題数 | 難易度：簡単の問題数 | 難易度：普通の問題数 | 難易度：難しいの問題数 |
| 5 | 60 | 15 | 30 | 15 |

## 実験

問題ファイルより生成される穴埋め選択問題の内容から，以下の表5.2に示す判断基準を満たす割合を算出した．計算は全問題を通しての割合の他に，難易度ごとの比較も行った．

表 5.2 実験の評価基準

|  |  |  |
| --- | --- | --- |
|  | 基準 | 評価 |
| １ | 正解となる解答が選択肢に入っている． | Yes / No |
| ２ | 設問内で選択肢が複数かぶっていない． | Yes / No |
| ３ | ブロックプログラミングより解答が推測できる． | Yes / No |
| ４ | 出題内容の種類 | （予約語，不等号，四則演算，  特殊記号） |
| ５ | 選択肢から正解が推測できる場合． | Yes / No |

## 実験結果

　実験結果は判断基準Ｘを満たすデータ数を問題数Ｎで割った割合を算出した．以下に全問題を対象とした場合と難易度毎の４種類を示す．

表 5.3 全問題の結果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 全問題 | | | | | | 問題数 | 60 |
| 基準１ | 基準２ | 基準３ | 基準４ | | | | 基準５ |
| 予約語 | 不等号 | 演算式 | 特殊記号 |
| 100% | 100% | 75% | 10% | 48% | 35% | 7% | 12% |

表 5.4 難易度：簡単の結果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 難易度：簡単 | | | | | | 問題数 | 15 |
| 基準１ | 基準２ | 基準３ | 基準４ | | | | 基準５ |
| 予約語 | 不等号 | 演算式 | 特殊記号 |
| 100% | 100% | 73% | 7% | 53% | 40% | 0% | 0% |

表 5.5 難易度：普通の結果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 難易度：普通 | | | | | | 問題数 | 30 |
| 基準１ | 基準２ | 基準３ | 基準４ | | | | 基準５ |
| 予約語 | 不等号 | 演算式 | 特殊記号 |
| 100% | 100% | 73% | 10% | 43% | 37% | 10% | 0% |

表 5.6 難易度：難しいの結果

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 難易度：難しい | | | | | | 問題数 | 15 |
| 基準１ | 基準２ | 基準３ | 基準４ | | | | 基準５ |
| 予約語 | 不等号 | 演算式 | 特殊記号 |
| 100% | 100% | 80% | 13% | 53% | 27% | 7% | 47% |

## 実験の考察

評価基準１，評価基準２の結果から，問題の自動生成は実装通りに動くことが確認できた．

評価基準３より，難易度が簡単または普通の場合に推測ができる箇所はブロックプログラミングにおいて「もし」とかかれている場合の「if」や日本語環境で「～まで」のように表示されている場合などがあった．推測ができない場合は，コーディングにおける構文に用いる「：」や，代入を表す「＝」と比較を表す「＝＝」など，ブロックプログラミングでは示されない箇所があった．日本語環境ブロックプログラミングとの連携による，コーディングへの架け橋としてブロックプログラミングから推測できる箇所とできない箇所について，できない，または難しいと判断される問題については解説をつけることで対応した．例えば「if」が正解の場合に，ブロックには「もし」と表示されている．こういった際に，コーディングでは「if」という予約語が対応することを文章として表示することで対応した．変数と数字の間を穴埋め問題として難易度が難しいで出題した場合に，選択肢として等号が一つしかなく，予約語が多く選択肢に選ばれた場合に，解答として予約語は当てはまりづらいと判断ができるような場合があった．

評価基準４は，今回の実験ではすべての場合において8割以上が不等号あるいは演算式から問題が生成されていることがわかった．これは，今回実験で利用している問題ファイルでは短い表現によって生成されることと，計算問題を多く利用したことによる代入や計算の多さが一つ目の理由である．また，文字列の表示をするコードが存在する場合に，本システムでは２か所が問題生成される可能性があるが，それぞれ不等号と演算式であることが二つ目の理由である．これらの理由からこの二つの種類の解答が多く生成されたと考えられる．

評価基準５とは，予約語，等号・不等号，四則演算，特殊記号のすべてのキーワードから３つランダムで選出するため，正解となる選択肢が予約語「if」の場合に，他２つのキーワードが「＝」「＋」のような不等号や四則演算のみからなる場合が存在する．このような問題文や前後のコードと関連する種類の選択肢が一つしか存在しない場合に，推測によって正解が推測できるといえる．今回の実験では，表5.6より47％の確率で評価基準５を満たしていることから，問題を提示する場合に，教師は注意してこの難易度を利用する必要があるが，ブロックプログラミングと前後のコードを比較することで理解が進むという観点で見れば，本研究の目的である，論理的思考力からコーディング力の養成に円滑に移行できるような教育支援学習として扱うことも可能だと考えられる．

以上のことから，提案方式において，論理的思考力からコーディング力の養成に円滑に移行できるような教育支援として，穴埋め問題の自動生成のシステムの設計・開発ができた．

# むすび

　本研究では，ブロックプログラミングからコーディング用の練習問題を自動生成することに着目し，ブロックプログラミングと連携したソースコードの穴埋め問題生成システムを構築した．

本研究において，問題文とブロックプログラミングによって生成するXMLデータなどの問題ファイルを提示することで，教師は学習支援を行う．また，教師によって提示された問題ファイルより自動で生成された問題に解答し，採点をうけることで学習者は学習を行うことができる．

実験では，本研究の問題自動生成機能によって生成される穴埋め選択問題が，ブロックプログラミングからコーディング学習のために適切に問題を生成することができるのか，複数の問題ファイルと難易度において実際に生成された穴埋め問題内容をいくつかの判断基準ごとに評価した．５種類の評価基準において，穴埋め問題の自動生成によって選択肢が不備なく生成されていること，生成された内容が学習のために利用できる内容であることから，提案システムの実現可能性を示した．

　今後の展望として，二つの内容が挙げられる．1つ目は，JavaScriptのソースコードを対象として本研究は穴埋め問題の生成を行っているが，HTMLなどと連携してフロントエンドでの動作の補助に利用されることが多いJavaScriptのコードでは，文字列の入出力の方法が他のプログラミング言語と比較して多種類だといえる．特に文字列の出力については，５章で利用した問題ファイルにも対応するソースコードが存在したが，問題として生成することはできていないことである．２つ目は，実験において出題された内容において不等号と演算式が出題される割合が多く，予約語の割合がかなり少なかったことである．本研究では，キーワードとマッチした箇所から乱数的に問題を生成していることと，実験のために作成した問題ファイルの内容となるコード片が代入や計算を行うことでアルゴリズムの学習をする内容だったことが原因で出題される割合に差が出てしまったと考える．これらのことに対応する方法を検討する．

# 

# **謝辞**

　本研究を行うにあたり，多くの助言をくださった鷹野孝典教授に心から感謝いたします．

研究を一緒に進めながら様々な意見を出し合い，本研究の方向性の確立にご協力くださった鷹野研究室の４年生のメンバーに深く感謝いたします．

# **参考文献**

1. 文部科学省 初等中等教育局 情報教育・外国語教育課 情報教育振興室, 小学校プログラミング教育の趣旨と計画的な準備の必要性について，令和元年度 小学校プログラミング教育担当者等セミナー， https://www.mext.go.jp/content/20200210-mxt\_jogai01-100013292\_01.pdf，

(2021/12/20)．

1. 小学校段階におけるプログラミング教育の在り方について（議論の取りまとめ），https://www.mext.go.jp/b\_menu/shingi/chousa/shotou/122/attach/1372525.htm, 2021/12/20．
2. 井上尚美：言語論理教育入門―国語科における思考，pp.32-33，明治図書，（1989/7/1）．
3. 道田康司：論理的思考とは何か？，琉球大学教育学部紀要，no.63，pp.181 -193，(2003/9)．
4. 萱津理佳, 香山瑞穂, 國宗永佳, 永井孝, 不破泰：アルゴリズム的思考法に関する学習を取り入れたC言語によるプログラミング教育の実践と評価 : 2010年度から2013年度の4年間の実践より [研究ノート]，長野県短期大学紀要，Vol.69，pp.69-78，(2015/02)．
5. 内田保雄：初級プログラミングのための自動作問システム，情報処理学会研究報告コンピュータと教育（CE），Vol.2007，123(2007-CE-092)，pp.109-113，(2007/12/08).
6. 野上裕二，納富一宏：プログラミング学習支援における問題自動生成に関する基礎的検討, 情報処理学会 第16回情報科学技術フォーラム(FIT2017)講演論文集, 第3分冊, K-022, pp.465-466, (2017.09).
7. 福坂祥基，高木正則，山田敬三，佐々木淳：問題自動生成システムを利用した作問演習の実践と評価，JSiSE研究会研究報告 32(5)，pp.107-114，（2018-01）．

# 付録

問題ファイルを付録につける

# その他

* ソースコード等を記述するときは，表内に挿入し，行間を「固定値：12pt」に設定する．行間を制する者はwordを制する．
* 研究でやったことはすべて，基本的に「本研究では」と称する
* ファイル名は，「1821~-name-thasis.docx」で統一する．
* 目次では、謝辞や参考文献のページ番号は載せない
* 要旨には，背景，課題，提案，実験で確認したことを書く
* 図表番号とラベルの間は半角スペースを入れる
* 節と節の間は改行する．章と章は改ページする
* 本文は日本語：MS明朝　　英語：Cenutty
* ただし数式は「挿入→数式」から入力し，本文中で数式を用いる場合はフォントの形が似ているTimes New Romanを使う．
* 付録にあるプログラムの説明がしたいとき本文の方に，例えば，“（付録３参照）”，“……を付録３に示す．”等と参 照先を記しておく．
* 章の終わりに改ページを行う．
* 改ページはCtrl+Enter
* 今後の課題ではなく展望．修正点とか書かない