**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**

# 目次

[**1.** **まえがき** 1](#_Toc89803156)

[1.1 はじめに 1](#_Toc89803157)

[1.2 本論文の構成 1](#_Toc89803158)

[2. 関連研究 2](#_Toc89803159)

[2.1 2](#_Toc89803161)

[**3.** **提案システム** 3](#_Toc89803162)

[3.1 提案方式の概要 3](#_Toc89803164)

[3.2 提案方式の構成 3](#_Toc89803165)

[3.2.1. 問題を自動生成する 3](#_Toc89803180)

[3.2.2. 問題の正誤を評価 4](#_Toc89803181)

[3.2.3. 整合性の評価 4](#_Toc89803182)

[3.3 提案方式を構成する機能 **エラー! ブックマークが定義されていません。**](#_Toc89803183)

[**4.** **実装システム** 6](#_Toc89803184)

[4.1 実装システムの概要 6](#_Toc89803186)

[4.2 実装システムの実行環境構築 6](#_Toc89803187)

[4.2.1. ・本研究のプログラムのダウンロード 6](#_Toc89803194)

[4.2.2. node.js 6](#_Toc89803195)

[4.2.3. サーバの実行コマンド 7](#_Toc89803196)

[4.3 実装システムの機能説明 7](#_Toc89803197)

[4.2.4. Visual Program Language 7](#_Toc89803199)

[4.2.5. コードジェネレート機能 7](#_Toc89803200)

[4.2.6. インデント 8](#_Toc89803201)

[4.2.7. JavaScriptコードの実行 10](#_Toc89803202)

[4.4 実装システムの穴あき問題の実装 11](#_Toc89803203)

[4.2.8. 穴埋め問題の生成 11](#_Toc89803205)

[4.5 実装システムの穴あき問題の評価の実装 11](#_Toc89803206)

[4.2.9. 穴埋め問題の解答の評価 12](#_Toc89803208)

[4.6 Blocklyの変更点 13](#_Toc89803209)

[4.2.10. text\_printを変更 13](#_Toc89803210)

[4.2.11. for文を変更 14](#_Toc89803211)

[**5.** **実験** 16](#_Toc89803212)

[5.1 実験目的 16](#_Toc89803214)

[5.2 実験環境 16](#_Toc89803215)

[5.3 実験 **エラー! ブックマークが定義されていません。**](#_Toc89803216)

[5.4 実験の考察 16](#_Toc89803217)

[**6.** **むすび** 17](#_Toc89803218)

[**謝辞** 18](#_Toc89803219)

[**参考文献** 19](#_Toc89803220)

[その他 19](#_Toc89803221)

# **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．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．

ブロックプログラミングと呼ばれる，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．

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，本研究では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．

＜＞

・背景の充実

・研究の概要・価値の充実

・日本語環境のブロックプログラミングによる学習者の導入のしやすさ

・ブロックプログラミングと連携したソースコードの選択問題生成

・実験や評価などの充実

・主張点の充実　特徴

## 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及び実験システムの実装について述べる．第５章では実験を行う。実験の概要と，比較実験後の結果を述べる．第６章では結論と今後の展望について述べる．

# 関連研究

・論理的思考

・授業支援

・多選択肢

・アルゴリズム的思考法に関する学習

・VPLの編集履歴に基づく学習分析

・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



## 

# **提案システム**

　本研究では，

なお，提案方式の概要を3.1節，提案方式の構成を3.2節，提案方式を構成する機能を3.3節に示す．

＜箇条書き 文章にする＞

・日本語ブロックプログラミングによってプログラム初級者でもわかりやすいプログラミング

・ブロックプログラミングとコーディングの比較

・選択式の演習問題による教育支援

・演習問題の自動生成

実装にも同じような内容を充実させる、提案と実装で利用するキーワードの変更をする



## 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．

## 

Step-1: XML状態のコードからJavaScriptコードを生成する．

Step-2:難易度によって選択問題に利用されるワード，問題数を選択する．

Step-3:JavaScriptコードから選択問題を自動生成する．

Step-4:解答の正誤を判別する．

Step-5:誤答の解説を行う．

Step-6: Step-2に戻る．

## 提案方式の構成



### 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．

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穴埋め問題出題のための選択肢自動生成方式

### 選択問題の難易度による選択肢の決定

箇条書きから文章に変更する

・難易度は３つ　簡単普通難しい

・初期では簡単が選ばれる

・簡単　同じカテゴリーから選ばれる選択問題を３問以下

・普通　同じカテゴリーから選ばれる選択問題を６問以下

・難しい 異なるカテゴリーから選ばれる選択問題を３問

・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

・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．

### 解答の正誤を採点

　3.2.1項にて作成し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．

# **実装システム**

　本研究では，

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，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．

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 |  |
| 開発言語 | JavaScript |  |
| ブラウザ | Google chrome |  |
| パッケージ管理 | npm |  |
| JavaScript実行環境 | Node.js | 16.13.1 |
| VPL | Blockly |  |

## 実装システムの実行環境構築



### ・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 4.2 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4.3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 4.3　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。下記にインストールのコマンドを示す。

表 4.4 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．

表 4.5　サーバの実行コマンド

|  |
| --- |
| npm start |

## 実装システムの機能説明



### 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。

表 4.6　 google/blockly githubリンク

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

表 4.7　 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、下記に示す。

表 4.8

|  |
| --- |
| <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。

表 4.9

|  |
| --- |
| 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。

表 4.10

|  |
| --- |
| 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．

表 4.11　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、あとで直します。  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．実際の実装を下記に示す．

表 4.12 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．

表 4.13 jsruntst()

|  |
| --- |
|  |

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。

表 4.14　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。

表 4.15　kaitouran(),

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

## 実装システムの選択式問題の実装

### XMLコードの生成とJavaScriptコードへの変換

xml.html

・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

・BlocklyによってブロックプログラムからXMＬコードに変換

・ＸＭＬコードからJavaScript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．

実際のコードは以下の通りである．

表 4.16　 match()

|  |
| --- |
|  |

## 実装システムの選択式問題の実装



### 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．

表 4.17　 answer2()

|  |
| --- |
|  |

　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．下記のコードによって実装．

### 実行結果と解答例との比較

表 4.18　 check2()

|  |
| --- |
|  |

numq説明

表 4.19　 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";  }  } |

説明

twice()

|  |
| --- |
|  |

### 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．

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．

gotou()

|  |
| --- |
|  |

## 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 4.20 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 4.21 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 4.22 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**



## 実験目的

## 実験環境

　４章の実装環境と同様

## 実験準備

### 問題文と対応したブロックプログラミングによるXMLの生成

・問題文を用意する

・問題に合わせてブロックプログラミングをする

・BlocklyによるXMLコードへの変更

・本システムにあわせたタグ、解答例の入力

## 実験

## 実験の考察

# **むすび**

# **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**

# 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