平成 30 年度

修士学位論文

プログラミング学習ゲーム「アルゴロジック」に関数を導入する拡張の検討

A Study on Extending Game-based Programming Education Tool 'Algologic' with Functions

1215078　　粕谷 彪人

指導教員　　鵜川 始陽

2019 年 2 月 28 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報学コース

要 旨

プログラミング学習ゲーム「アルゴロジック」に関数を導入する拡張の検討

粕谷 彪人

近年プログラミング教育への関心が高まっており，2020 年には初等教育ではプログラミング教育が導入される．初等教育でのプログラミング教育の目的の一つに「プログラミング的思考を養う」がある．しかし我々は，プログラミング的思考のみではなく Wing が提唱する Computational Thinking も学習するのが良いと考える．プログラミング的思考を学習するために，ビジュアルプログラミング言語が開発されている．その一つにアルゴロジックがある．アルゴロジックは，プログラミングの制御構造である順次実行，ループ，条件分岐をゲーム感覚で体験することができる．しかし，アルゴロジックにはモジュール化の機能は存在しない．

そこで，本研究ではアルゴロジックにモジュール化の機能として関数を追加することを考えた．関数を追加したアルゴロジックでは，関数呼び出しの様子を表示する必要がある．本研究では，コールスタックを表示することで関数呼び出しの様子を表した．関数には，プログラムの列を引数として，一つだけ渡せるようにした．この引数の実行も分かりやすく表示する必要がある．そこで，引数の実行時の表示方法を，3 種類実装した．一つ目の表示方法は，呼び出した側の関数の定義中にある命令ブロックをハイライトする表示方法である．二つ目の表示方法は，関数呼び出しの際に仮引数を実引数に置き換える表示方法である．三つ目の表示方法は，引数ブロックを実行時に実引数のブロックの列をコールスタックに追加し表示する方法である．この 3 種類の表示方法を実装したアルゴロジックをユーザに使ってもらいどの表示が一番分かりやすかったか調査した．その結果，一つ目の表示方法が一番分か

りやすいことが分かった.

キーワード　ビジュアルプログラミング言語，プログラミング教育，モジュール化

Abstract

A Study on Extending Game-based Programming Education Tool 'Algologic' with Functions

Ayato Kasutani

In recent years, interest in programming education is increasing, and programming education will be introduced in elementary education in 2020. One of the goals of programming education in elementary education is 'to develop programming thinking'. However, we think that it is better to learn not only programming thinking but also computational thinking proposed by Wing. In order to learn programming thinking, visual programming languages are being developed. Algologic is one of such languages. With Algologic, users can experience control structures of programming, such as sequential executions, iterations and conditional branches, in a game. However, there is no modularization facilities in Algologic.

In this study, we discuss on adding procedures as a modularization facility to Algologic. The Algologic with procedures have to visualize invocations of procedures. We visualize procedure calls by showing the call stack. We designed procedures so that any procedure receives a single parameter. Execution of this parameter also have to be visualized. To seek the best way of visualization for understanding, we implemented three ways of visualization of parameter execution. The first a way is to highlight the instruction block in the definition of the caller procedure. The second a way is to replace the parameter with the argument when a procedure is called. The third a way is to push a sequence of the argument blocks to the call stack when executing a parameter

block. We surveyed the best way of visualization for understanding from users who used Algologic with all the three ways. As a result, we found that the first a way of visualization was the best.

key words visual programming language, programming education, modularization

目次

第 1 章　はじめに　1

第 2 章　関連研究　5

2.1　Scratch 5

2.2　まねっこダンス 5

2.3　Google Blockly 6

第 3 章　アルゴロジック　8

3.1　概要 8

3.2　文法と意味 10

3.2.1　文法 10

3.2.2　評価規則 11

第 4 章　関数を持つアルゴロジック　14

4.1　引数の導入 14

4.2　再帰呼び出し 14

4.3　文法 15

4.4　意味 16

第 5 章　画面表示　17

5.1　概要 17

5.2　プログラム編集中の画面 19

5.3　プログラム実行中の画面 21

5.4　プログラム実行中の引数ブロックの表示 21

5.4.1　戻り表示 22

5.4.2 置換表示 23
5.4.3 スタック表示 24

第 6 章 引数の表示方法の比較 26

6.1 引数の実行時の表示方法の比較 26
6.2 考察 27

第 7 章 おわりに 31

謝辞 32

参考文献 33

図目次

1.1 アルゴロジックの画面
（JEITA アルゴリズム体験ゲームアルゴロジック https://home.jeita.or.jp/is/highschool/algo/prm/index.html の web サイトより） . . . 2
1.2 図 1.1 のステージの解答例 . . . 3
1.3 図 1.2 の実行時の例
（JEITA アルゴリズム体験ゲームアルゴロジック https://home.jeita.or.jp/is/highschool/algo/prm/index.html の web サイトより） . . . 4

2.1 まねっこダンスの課題（文献 [6] より） . . . 6

3.1 アルゴロジックの文法 . . . 11
3.2 *eval* の入出力 . . . 11
3.3 キャラクターの向きを $(1, 0)$ で表した時の実際の向き . . . 12
3.4 アルゴロジックの意味 . . . 13

4.1 関数を扱えるように拡張したアルゴロジックの文法 . . . 15
4.2 関数呼び出しの評価規則 . . . 16

5.1 プログラム編集中の画面 . . . 18
5.2 プログラム実行中の画面 . . . 18
5.3 エディタ . . . 19
5.4 ブロックの挿入の例 . . . 21
5.5 関数が呼びされた時の例 . . . 22
5.6 引数を持つ関数呼び出しのプログラム例 . . . 23
5.7 戻り表示での実行例 . . . 23

5.8 置換表示での実行例 24
5.9 スタック表示での実行例 24

6.1 一つ目のステージ 27
6.2 二つ目のステージ 27
6.3 三つ目のステージ 28
6.4 アンケートの結果 28
6.5 戻り表現のハイライトが遠くに移動する様子 30

表目次

3.1 命令ブロックの種類 9

5.1 命令ブロックの種類 20

6.1 アンケートでもらったコメント 29

第 1 章

はじめに

近年プログラミング教育への関心が高まっており，2020 年には初等教育でプログラミング教育が導入される．初等教育でのプログラミング教育の目的の一つに「プログラミング的思考を養う」が挙げられている [1]．プログラミング的思考とは,「自分が意図する一連の活動を実現するために，どのような動きの組み合わせが必要であり，一つ一つの動きに対応した記号を，どのように組み合わせたらいいのか，記号の組み合わせをどのように改善していけば，より意図した活動に近づくのか，と言ったことを論理的に考えていく力」と説明されており，プログラミング的思考を養うことで論理的，創造的に思考し課題を発見し解決する力が身に付くと考えられている．しかし我々は，プログラミング的思考のみではなく Wing が提唱する Computational Thinking（以下，CT）[3] も学習するのが良いと考える．CT とは，大きな問題を細分化や一般化して問題を解決するための手順を明確にする，全ての人にとっての基本的なスキルである [3]．諸外国では，CT を取得することを目標に学習されている．また，パパート [4] も構成要素をモジュール化し独立させる力を養うことの重要性を主張している．

プログラミング的思考を学習するために，初学者向けのプログラミング言語であるビジュアルプログラミング言語が開発されている [2][5][6][8]．ビジュアルプログラミング言語とは，一般的なプログラミング言語のようにテキストでプログラムを記述するのではなく，画面に表示されているブロックなどを用いてプログラムを記述する言語のことである．

ビジュアルプログラミング言語の一つにアルゴロジック [2] がある．アルゴロジックとは，ブロックでプログラムを作成し，プログラムの制御構造である順次実行，ループ，条件分岐をゲーム感覚で体験することができるプログラミング学習ゲームである．図 1.1 のように左

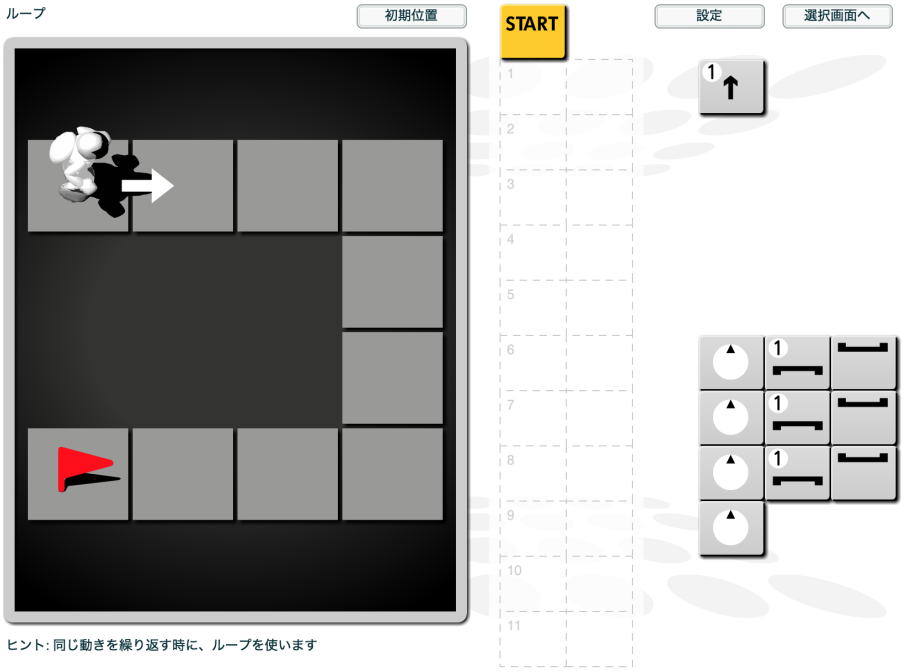

図 1.1　アルゴロジックの画面
（JEITA アルゴリズム体験ゲームアルゴロジック `https://home.jeita.or.jp/is/highschool/algo/prm/index.html` の web サイトより）

に課題（以下，ステージ）を表示し，ステージの中にロボットと旗を表示する．このゲームは，ロボットを制御するブロックとプログラムの制御構造であるループと条件分岐のブロックでプログラムを作成し全ての旗を回収するゲームである．プログラムの部品であるブロックはマウスで移動させることができ，ユーザがキーボードを触る必要はない．このゲームは提示されたブロックしか使うことができないため図 1.1 のようにループのブロックを使わないとクリアすることができないステージもある．図 1.1 をクリアするためのプログラムを図 1.2 に示す．このプログラムは，まずロボットを 3 マス分前進させる．その後，右に 90 度回転する．この二つの実行を 3 回繰り返すことで旗を回収しステージをクリアすることができる．アルゴロジックでは，プログラムの実行時に実行中のブロックを図 1.3 のようにハイライトしている．ハイライトしているため，ロボットの動きとプログラムの対応が取りやすく，間違えた場合どこで間違えたかがわかりやすくなっている．アルゴロジックでは，このようにプログラミングを体験しながらプログラミング的思考を養うことができる．

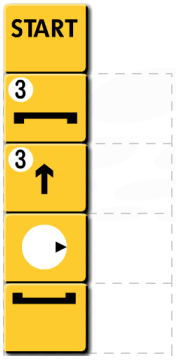

図 1.2　図 1.1 のステージの解答例

しかし，アルゴロジックにはモジュール機能は存在しない．そこで，アルゴロジックにモジュール機能を追加することを考えた．アルゴロジックにモジュール機能を追加する場合，モジュールはブロックの列とし，一般のプログラム言語の関数のように扱えるようにするのが自然である．本研究では，ブロックの列を関数として扱う機能をアルゴロジックに導入する際に，どのような仕様にすれば良いかを検討する．特に，関数の引数を導入するかどうかや，引数として何を渡せるようにするか，関数の実行をどう表示するかを検討する．また，関数を追加したアルゴロジックはどのような意味論を持つプログラミング言語なのか考察する．

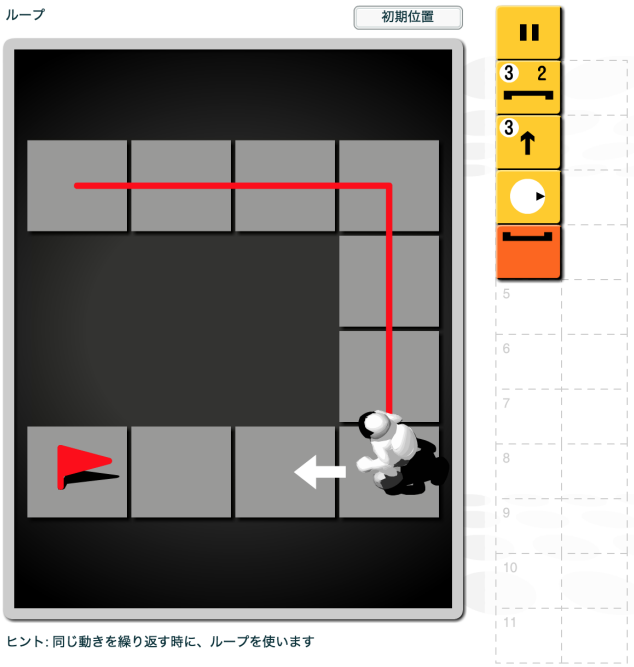

図 1.3　図 1.2 の実行時の例

（JEITA アルゴリズム体験ゲームアルゴロジック `https://home.jeita.or.jp/is/highschool/algo/prm/index.html` の web サイトより）

第 2 章

関連研究

2.1 Scratch

Scratch は，プログラミングの基本を学習することができるプログラミング学習ツールである [5]．Scratch は，C 言語や Java のようにテキストでプログラムを記述するのではなく，画面に表示されているブロックなどを利用してプログラムを記述するビジュアルプログラミング言語である．

Scratch では，画面にキャラクターなどを表示し，キャラクターを動かしたり音を出したりと自由に作品を作ることができ，学習者の創造性を養うことができる．また作成した作品は，オンラインで共有し，他人の作品にコメントしたり逆にコメントをもらったりできる．このように，お互いに交流することで協力して作業する能力も養うことができる．

Scratch には，関数が実装されており，引数も渡すことができる．引数には，数値，テキスト，真偽値を渡すことができる．しかし，Scratch はプログラムの命令がハイライトされないため，プログラムの命令と実行中の動作の対応が取りにくく，意図した動作ではなかった場合に修正が難しい．

2.2 まねっこダンス

まねっこダンスは，絵文字と日本語で記述できるプログラミング言語の一つであり，画面に表示されたキャラクターの踊りの真似をするプログラムを記述することでプログラムを学習することができる [6]．まねっこダンスは，PC で動作するのではなく若年層に馴染みのあ

図 2.1　まねっこダンスの課題（文献 [6] より）

るスマートフォンで動作する．

まねっこダンスでは，画面に表示されたキャラクターの踊りの真似をするという課題が出題される．ユーザはキャラクターの動きを指示する命令と，ループと分岐の制御構造を使ってプログラムを作成する．図 2.1 のように，画面に表示されたキャラクターの踊りと記述したプログラムの踊りが完全に一致した場合に正解となる．このように，課題を出題することで学習者が取り組むべき課題を明確化している．

作成したプログラムを実行する際，実行中の命令がハイライトされる．そのため，実行中の命令とキャラクターの踊りの対応が取りやすく，もしプログラムが不正解だったとしても容易に間違いの場所を見つけるけることができる．

まねっこダンスには，プログラムをモジュール化する機能は用意されていない．

2.3　Google Blockly

Google Blockly は，ビジュアルプログラミングエディタを開発するための JavaScript ライブラリである [7]．このライブラリを用いて開発されたゲーム Blockly Games が Web 上に公開されている [8]．Blockly Games は，プログラミング経験のない子供を対象にしており，ゲームを通してテキストでプログラミングをするための準備を目標としている．

Blockly Games には 8 種類のゲームがあり，各ゲームで課題が出題される．学習者はその課題をクリアするためのプログラムを記述する．ゲームには段階があり，最初はブロック

を用いたプログラミングを行い最終的にはテキストでプログラムを記述するゲームもある．そのため，学習者はテキストでプログラムを記述するためのハードルは下がると考えられる．また，ゲームの種類も多く子供が飽きにくいと考えられる．

Blockly Games は，ゲームによっては関数を定義して用いることができる．引数には，数値，真偽値を渡すことができる．しかし，実行中のブロックがハイライトされず，課題をクリアできなかった場合，プログラムの修正が難しい．

第 3 章

アルゴロジック

本研究では，アルゴロジックに関数を導入することを検討している．アルゴロジックにはいくつかのバージョンがあるが，ここでは条件分岐を含まないアルゴロジックを対象とする．そこで本章では，アルゴロジックの概要について説明し，アルゴロジックをプログラミング言語として見たときの文法と意味を定義する．

3.1 概要

アルゴロジックは，プログラムの基本をゲーム感覚で学習することができるプログラミング学習ゲームである．このゲームは，図 1.1 の左側にゲームの課題（以下，ステージ）が出題される．ステージには，ロボットと旗が表示されている．この旗を全て回収するようなロボットを制御するプログラムを作成することでステージをクリアできる．

プログラムは，図 1.1 の右側にある命令ブロックを使って作成する．命令ブロックの種類を表 3.1 に示す．表 3.1 の前進ブロックは，ロボットが向いている方向に左上の数値分移動する命令ブロックである．表 3.1 の左進ブロックは，ロボットが向いている方向に対して，左方向に左上の数値分横移動する命令ブロックである．表 3.1 の右進ブロックは，ロボットが向いている方向に対して，右方向に左上の数値分横移動する命令ブロックである．表 3.1 の回転ブロックは，ロボットが現在向いている方向から 7 方向に向きを変える命令ブロックである．表 3.1 のループ開始ブロックと表 3.1 のループ終了ブロックは，これらの間の命令ブロックをループ開始ブロックの左上の数値の回数繰り返す命令ブロックである．

ユーザは命令ブロックを，マウスで移動させ，図 1.1 の中央のスペースでプログラムを作

ブロック	ブロック名
	前進ブロック
	左進ブロック
	右進ブロック
	回転ブロック
	ループ開始ブロック
	ループ終了ブロック

表 3.1　命令ブロックの種類

成する．例えば，図 1.1 の課題をクリアするプログラムは図 1.2 のようになる．図 1.2 のプログラムは以下の処理で構成されている．

処理 1　ロボットを 3 マス分前進させる

処理 2　右に 90 度回転する

処理 3　処理 1 に制御を移す

作成したプログラムは，プログラムを作成するためのスペースの上にあるスタートボタンを押すことで実行することができる．

プログラムが実行されると，プログラム通りにロボットが動作する．ロボットがマスから出ることなく全ての旗を回収することでステージクリアとなる．クリアできなかった場合，学習者はクリアを目指してプログラムを修正（以下，デバッグ）することができる．プログラムを実行すると，ロボットの動作に対応してプログラムがハイライトされる．ハイライトされることでロボットがマスから出た場合どのブロックが意図した動作をしていないかなど調べ，デバッグを容易にしている．

アルゴロジックでは，プログラムの制御構造である順次実行やループ，条件分岐を体験できる．アルゴロジックは，各ステージで使用できるブロックが決まっている．そうすること

で，学習者にループや条件分岐のブロックを使用しないといけない状況を作っている．また，ループのブロックなどを使わなくてもクリアできるステージもある．しかしアルゴロジックはクリアを 2 段階に分けている．1 段階目は，使えるブロックでとにかくクリアすることで達成される．2 段階目は，最小ブロック数で課題をクリアすることで達成される．ループなどを使わずクリアした場合，1 段階目のクリアとなるように作られている．2 段階目のクリアにこだわることでループなどの学習のモチベーションの維持にも繋がると考えられる．

このように順次実行やループ，分岐については学習できるが，プログラムをモジュール化する機能は実装されていないため学習できない．モジュール化を学習できるようにアルゴロジックに関数を定義する機能を追加する場合，特に実行中の引数の表示方法が問題になると考えられる．例えば引数付き関数を呼び出した場合，実行時にどのブロックをハイライトするとロボットの動きとの対応が取りやすいかなどが問題になる．

3.2　文法と意味

本節では，アルゴロジックをプログラミング言語として見たときの文法と意味を定義する．アルゴロジックでは，プログラムによってキャラクターを動作させる．本論文では，キャラクターの動作をアルファベットとする文字列を使い，プログラムの実行によって引き起こされるキャラクターの動作に対応する文字列でプログラムの意味を与える．

3.2.1　文法

アルゴロジックの文法を図 3.1 に示す．プログラムは ⟨ 命令 ⟩ の列である．⟨ 命令 ⟩ には，ループ（[$_n$ ⟨ 列 ⟩]）とキャラクターの動作を表す命令（F，L，R）がある．ループの n は，⟨ 列 ⟩ を繰り返し実行する回数を表す整数である．キャラクターの動作の命令は 3 種類あり，F は前進，L はキャラクターの向きを左に 90 度回転，R はキャラクターの向きを右に 90 度回転する命令である．

$$
\begin{aligned}
\langle\ \text{列}\ \rangle ::= &\langle\ \text{命令}\ \rangle\langle\ \text{列}\ \rangle \\
&\mid\ \varepsilon \\
\langle\ \text{命令}\ \rangle ::= &\texttt{[}_n\langle\ \text{列}\ \rangle\texttt{]} \\
&\mid\ \texttt{F} \mid \texttt{L} \mid \texttt{R}
\end{aligned}
$$

図 3.1　アルゴロジックの文法

$$
\begin{aligned}
p \in P &::= \text{プログラム} \\
d \in D = \{(1,0),\ (0,1),\ (-1,0),\ (0,-1)\} &::= \text{キャラクターの向き} \\
s \in D^* = S &::= \text{出力文字列}
\end{aligned}
$$

図 3.2　*eval* の入出力

3.2.2　評価規則

3.2.1 節で定義した文法に沿ったプログラムの意味を定義する．プログラム p に意味を与える関数を *eval* で表す．*eval* の入力と出力は，

$$eval : P \times D \to S \times D$$

とする．ここで入力の P はプログラム，D は現在のキャラクターの向きである．出力の S はキャラクターの動作を表す出力文字列，D は実行後のキャラクターの向きである．上記 P，D，S の定義を図 3.2 に示す．図 3.2 の p は，プログラムを表す．図 3.2 の d は，キャラクターの向きを x 軸方向と y 軸方向の組みで表している．例えば，キャラクターの向きが $(1,0)$ だった場合，キャラクターは図 3.3 のように x 軸の正の方向を向いている．キャラクターの向きが $(-1,0)$ だった場合，キャラクターは x 軸の負の方向を向くことになり図 3.3 とは逆の方向を向いていることを表す．

アルゴロジックのプログラムの意味を図 3.4 に示し，それぞれ説明する．

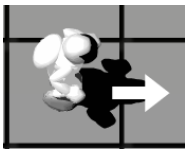

図 3.3　キャラクターの向きを $(1,0)$ で表した時の実際の向き

$eval(\mathtt{F}, d)$ の入力 F は，キャラクターを前進させる命令である．そのため，もう一つの入力である現在のキャラクターの向き d を出力文字列として返す．この命令では，キャラクターの向きは変わらないので $eval$ 関数の二つ目の出力 D は入力の d と同じである．よって，$eval(\mathtt{F}, d) = (d, d)$ となる．

$eval(\mathtt{L}, d)$ の入力 L は，キャラクターの向きを左向きに 90 度回転させる命令である．そのため，出力文字列はなく，$eval(\mathtt{L}, d) = (\varepsilon, (-y, x))$ となる．

$eval(\mathtt{R}, d)$ の入力 R は，キャラクターの向きを右向きに 90 度回転させる命令である．そのため，L 同様，出力文字列はなく，$eval(\mathtt{R}, d) = (\varepsilon, (y, -x))$ となる．

$eval(c \cdot p, d)$ の入力は，図 3.1 の ⟨ 列 ⟩ である．ここで c は，⟨ 命令 ⟩ であり，p は以降のプログラムを表す ⟨ 列 ⟩ である．$eval(c \cdot p, d)$ は，$eval(c, d) = (s_c, d')$ と $eval(p, d') = (s_p, d'')$ が成り立つ時，$eval(c \cdot p, d) = (s_c \cdot s_p, d'')$ となる．

$eval(\varepsilon, d)$ の入力は空のプログラムである．そのため，処理は行われず，$eval(\varepsilon, d) = eval(\varepsilon, d)$ となる．

$eval(\mathtt{[}_n \ \ p \ \mathtt{]}, d)$ の入力は，繰り返し回数 n が 0 より大きいループ命令である．$eval(\mathtt{[}_n \ p \ \mathtt{]}, d)$ は，$eval(p, d) = (s_0, d')$ と $eval(\mathtt{[}_{n-1} \ p \ \mathtt{]}, d) = (s_1, d'')$ が成り立つ時，$eval(\mathtt{[}_n \ p \ \mathtt{]}, d) = (s_0 \cdot s_1, d'')$ となる．n が 0 のとき，入力である $\mathtt{[}_0 \ p \ \mathtt{]}$ は実行されないため，$eval(\mathtt{[}_0 \ p \ \mathtt{]}, d) = (\varepsilon, d)$ となる．

$$eval(\mathtt{F}, d) = (d, d)$$

$$eval(\mathtt{L}, (x, y)) = (\varepsilon, (-y, x))$$

$$eval(\mathtt{R}, (x, y)) = (\varepsilon, (y, -x))$$

$$\frac{eval(c, d) = (s_c, d') \qquad eval(p, d') = (s_p, d'')}{eval(c \cdot p, d) = (s_c \cdot s_p, d'')}$$

$$eval(\varepsilon, d) = (\varepsilon, d)$$

$$\frac{eval(p, d) = (s_0, d') \qquad eval(\mathtt{[}_{n-1} p \mathtt{]}, d')}{eval(\mathtt{[}_n\ p\ \mathtt{]}, d) = (s_0 \cdot s_1, d'')\ (\text{ただし } n > 0)}$$

$$eval(\mathtt{[}_0\ p\ \mathtt{]}, d) = (\varepsilon, d)$$

図 3.4　アルゴロジックの意味

第 4 章

関数を持つアルゴロジック

本章では，本研究で設計した関数を持つアルゴロジックのプログラミング言語としての仕様を示す．

4.1　引数の導入

本研究では，関数に引数を渡せるようにする．引数を渡せるようにすることで，関数を一般化しやすくなると考えた．しかし，二つ以上の引数を渡せるようにした場合，引数を二つ以上使うことができるステージを用意する必要がある．そうなると課題の難易度が上がり学習者の学習意欲を削ぐ可能性があるため，引数は一つだけ渡すことができるようにした．

次に，引数で渡すものについて検討する．引数で渡すものは，空列を含む命令ブロックの列にする．引数には，一つのブロックや数値を渡す仕様も考えられる．しかし，一つのブロックを渡すようにした場合，ループ開始のブロックを渡すことができ，実行時に構文エラーが発生する可能性がある．構文エラーは学習者がつまづく要因になると考え，構文エラーが発生しにくいプログラム列を渡すようにした．また，数値を導入すると，プログラムの実行中の状態を構成する要素が増える．そのため，数値は導入しないことにした．

4.2　再帰呼び出し

本教材では，条件分岐は実装していないため，関数の再帰呼び出しをした場合に終了することができない．そのため，関数の再帰呼び出しを禁止するか検討した．

しかし，本教材では関数の再帰呼び出しは禁止しないことにした．学習者が作った関数は，

$$
\begin{aligned}
\langle\ \text{列}\ \rangle ::= &\ \langle\ \text{命令}\ \rangle\langle\ \text{列}\ \rangle \\
&\mid\ \varepsilon \\
\langle\ \text{命令}\ \rangle ::= &\ \texttt{[}_n\langle\ \text{列}\ \rangle\texttt{]} \\
&\mid\ \texttt{F} \mid \texttt{L} \mid \texttt{R} \mid \texttt{P} \\
&\mid\ \langle\ \text{関数名}\ \rangle\ \texttt{[}\langle\ \text{列}\ \rangle\texttt{]} \\
\langle\ \text{関数名}\ \rangle ::= &\ \texttt{S} \mid \texttt{H} \mid \texttt{D} \mid \texttt{C}
\end{aligned}
$$

図 4.1　関数を扱えるように拡張したアルゴロジックの文法

他の関数と組み合わせることで機能する場合があると考えられる．また本教材では無限ループは実装されていない．そこで，無限ループの代わりに制御するキャラクターをゴールさせることで終了する再帰関数を作ることができるようにした．

4.3　文法

関数を扱えるように拡張したアルゴロジックの文法を図 4.1 に示す．

キャラクターを制御する命令はアルゴロジック同様，前進の F，キャラクターの向きを左に 90 度回転させる L，キャラクターの向きを右に 90 度回転させる R の 3 種類である．また，ループもアルゴロジックと同様に，[$_n$⟨ 列 ⟩] で記述することができる．

関数は 4 つ（S，H，D，C）定義できるようにした．関数は引数を必ず 1 つ受け取る．関数の定義中で仮引数に束縛されたプログラムを実行するには P の命令を使う．関数呼び出しは，「⟨ 関数名 ⟩ [⟨ 列 ⟩]」で記述することができる．関数名は，S，H，D，C である．実引数は，「⟨ 関数名 ⟩ [⟨ 列 ⟩]」の ⟨ 列 ⟩ で指定するため，複数の命令の列を指定することができる．

$$\frac{get\_func(f) = p' \qquad eval(p'[\texttt{P} := p], d) = (s, d')}{eval(f\texttt{[} p \texttt{]}, d) = (s, d')}$$

図 4.2　関数呼び出しの評価規則

4.4　意味

アルゴロジックで関数を使えるようにするため，図 3.4 のアルゴロジックのプログラムの意味を表す関数 *eval* に関数呼び出しの評価規則を追加した．関数を扱えるように拡張したアルゴロジックの *eval* の定義のうち，関数呼び出しの規則を図 4.2 に示す．ここで f は `S`，`H`，`D`，`C` のうちのいずれかの関数名であり，$get\_func(f)$ は f の定義を表す．また，$p'[\texttt{P} := p]$ は，命令の列 p' 中の `P` を p で置き換えた命令列の列を表す．

関数呼び出しの規則で $eval(f\texttt{[} p \texttt{]}, d)$ の入力の $f\texttt{[} p \texttt{]}$ は，関数 f を呼び出す命令である．まず *get_func* を用いて関数名 f の定義であるプログラム p' を取る．取ってきたプログラム p' の中に仮引数があった場合，仮引数を実引数 p に置き換える．そのプログラム $p'[P := p]$ をキャラクターの向き d で評価して得られる動作の列 s と向き d' が $f\texttt{[}p\texttt{]}$ を向き d で評価した結果になる．

第 5 章

画面表示

本章では，関数を扱えるように拡張したアルゴロジックの画面表示の設計を行う．また，プログラム実行時の引数の表示方法についても説明する．

5.1 概要

アルゴロジックは，課題を出題し学習者が出題された課題を達成するプログラムを作成するゲームである．アルゴロジックのゲーム画面は，プログラム編集中の画面と，プログラム実行中の画面の二つで構成される．ゲームはプログラム編集中の画面で始まり，プログラムの実行ボタンを押すことでプログラム実行中の画面に切り替わる．実行が終わるとプログラム編集中の画面へと戻る．本研究で開発する，関数を扱えるように拡張したアルゴロジックも同様の画面の構成にする．プログラム編集中の画面は，図 5.1 のようにする．図 5.1 のように左側に課題となるステージを出題する．課題を達成する条件は，キャラクターを壁に衝突させないようにゴールである宝箱に移動させるプログラムを作成することである．プログラム実行中の画面は図 5.2 のようにする．アルゴロジックと同様に，実行中の命令をハイライトしキャラクターの動きと対応を取りやすいようにする．

5.1　概要

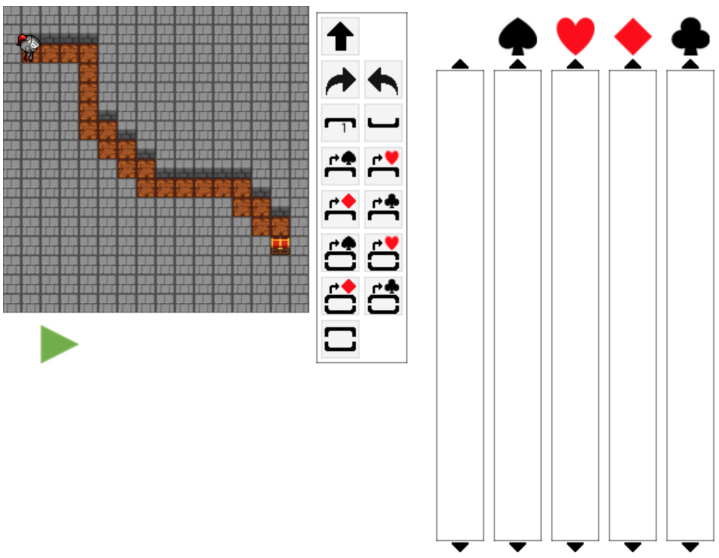

図 5.1　プログラム編集中の画面

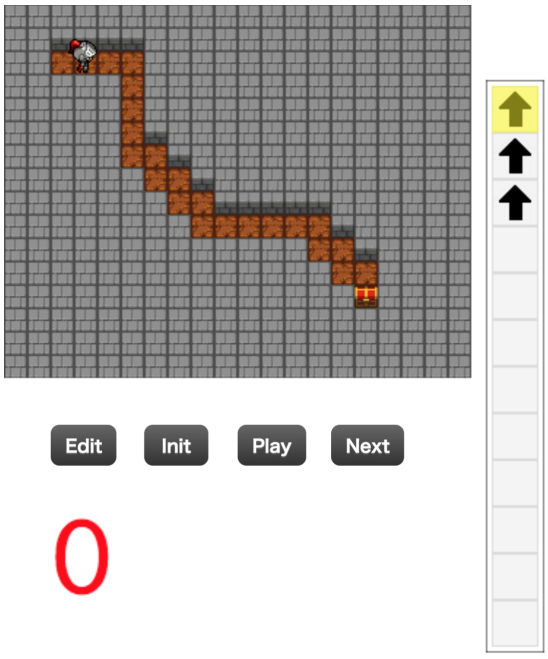

図 5.2　プログラム実行中の画面

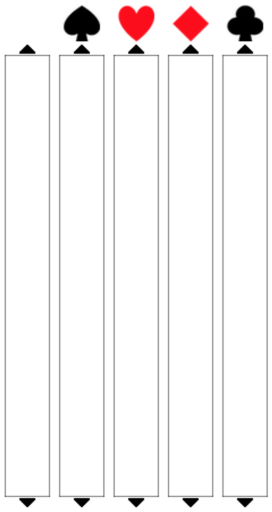

図 5.3　エディタ

5.2　プログラム編集中の画面

プログラムは，図 5.1 の右側にあるエディタ（図 5.3）で作成する．エディタの一番左の帯は最初に実行されるプログラムを作る領域である．上にトランプのマークのある帯は，関数を定義する領域である．トランプのマークは関数名に相当する．また，関数は 4 つまで定義できる．命令ブロックは，プログラム編集中の画面の中央に存在する．命令ブロックの一覧を表 5.1 に示す．引数にプログラムを渡す必要がない場合は，空列を渡す関数呼び出しブロックを使うことができる．

命令ブロックをマウスで移動させることで学習者が直感的に操作できるようにした．例えば，図 5.4 の左側のようにすでにブロックが二つ並んでいる場合，その二つのブロックの間に新しいブロックを挿入することができる．このように，学習者が配置したいと思った場所にブロックを挿入できるようにする．

関数を使う必然性があるステージは大きくなってしまう．それに伴い，学習者が作成するプログラムは大きくなると予想される．そのため本教材では，大きなプログラムを書けるようにする必要がある．大きなプログラムを書くための方法として，

1. プログラムとして命令ブロックを配置できる数の制限をなくす

5.2　プログラム編集中の画面

ブロック	ブロック名
	前進ブロック
	右 90 度回転ブロック
	左 90 度回転ブロック
	ループ開始ブロック
	終了ブロック
	スペード関数呼び出しブロック
	ハート関数呼び出しブロック
	ダイヤ関数呼び出しブロック
	クローバー関数呼び出しブロック
	空列を引数に渡すスペード関数呼び出しブロック
	空列を引数に渡すハート関数呼び出しブロック
	空列を引数に渡すダイヤ関数呼び出しブロック
	空列を引数に渡すクローバー関数呼び出しブロック
	仮引数ブロック

表 5.1　命令ブロックの種類

2. 関数を定義できる数を多くする

の 2 通り考えられる．本研究では，一つの関数定義で命令ブロックを配置できる数をなくすことにした．

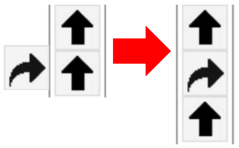

図 5.4　ブロックの挿入の例

5.3　プログラム実行中の画面

作成したプログラムを実行した際，図 5.2 のように最初に実行される命令の列のみが表示される．アルゴロジックと同様に，キャラクターの動作と実行中のプログラムの対応が取れるように実行中のプログラムはハイライトしている．そうすることで課題を達成できなかった場合，デバッグを容易にすることができると考えた．

本教材は，アルゴロジックにはない関数が存在する．関数呼び出しのブロックが実行された際，関数を呼び出している様子を表示する必要がある．そこで，図 5.5 のように実行画面にコールスタックを表示する．関数が呼び出された場合，コールスタックの中に呼び出した関数の定義を表示し，ハイライトするようにする．

5.4　プログラム実行中の引数ブロックの表示

引数で渡されているものが，命令ブロックの列なので，引数ブロックを実行すると，実引数として渡された命令ブロックを実行することになる．そのため課題を達成できずデバッグする際，デバッグを容易にするため，次に実行されるブロックを分かりやすくまたは，どのような経緯でその命令ブロックが実行されているかを表示する必要がある．そこで本研究では，わかりやすい表示方法を調査する．引数の実行の表示を，引数ブロックを実行した時に，呼び出した側の関数の定義中にある実引数の命令ブロックをハイライトする方法（以下，戻り表示），プログラムが呼び出された際に仮引数を実引数で置き換える方法（以下，置換表示），引数ブロックの実行時に実引数をコールスタックに追加し表示する方法（以下，ス

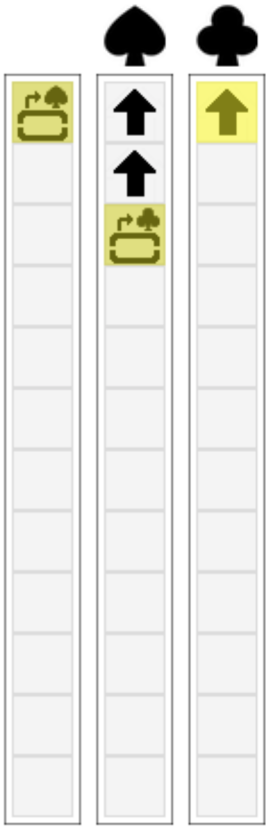

図 5.5　関数が呼びされた時の例

タック表示）の 3 種類を考えた．以下でそれぞれの表示方法について説明する．説明の際に使うプログラムを図 5.6 に表示する．図 5.6 のプログラムは，関数呼び出しを行い，引数として前進ブロックを 2 個渡す．呼び出された関数は，前進ブロックの後，仮引数ブロックを実行するプログラムである．図 5.7，図 5.8，図 5.9 は，図 5.6 のプログラムで呼び出された関数クローバーが仮引数ブロックを実行し，その実引数の 1 ブロック目を実行中の様子をそれぞれの表示方法で示している．

5.4.1　戻り表示

戻り表示は，呼び出した側の関数の定義中にある命令ブロックをハイライトする表示方法である．図 5.7 に図 5.6 のプログラムの実行時の様子を示す．コールスタック中の仮引数ブロックが実行された場合，呼び出し元のコールスタックにハイライトが移り，関数に渡した

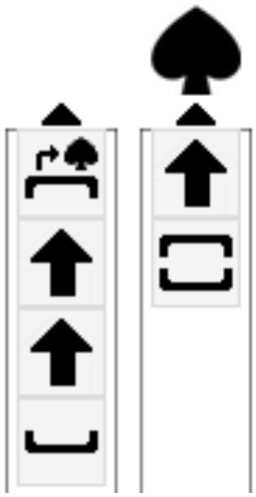

図 5.6　引数を持つ関数呼び出しのプログラム例

図 5.7　戻り表示での実行例

実引数ブロックをハイライトする．そうすることで，作成したプログラム上で実行の様子を表示するため，デバッグしやすいと考える．

5.4.2　置換表示

置換表示は，関数の呼び出し時に仮引数を実引数に置き換える表示方法である．図 5.8 に図 5.6 のプログラムの実行時の様子を示す．関数を呼び出し，コールスタックに呼び出された関数の定義を積む際に仮引数を実引数に置き換えている．この表示方法では，最新のコー

図 5.8　置換表示での実行例

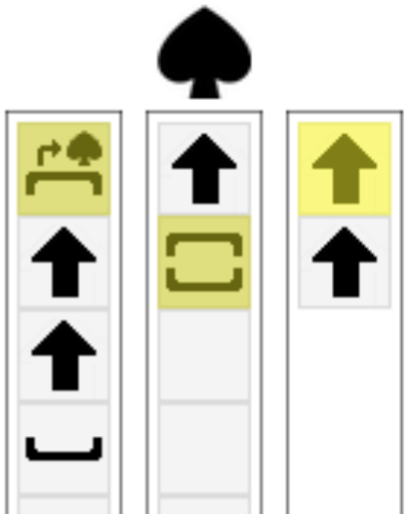

図 5.9　スタック表示での実行例

ルスタックのみを見るだけで実行中のブロックが分かるため，次に実行されるブロックの予想がつきやすい．

5.4.3　スタック表示

スタック表示は，仮引数の実行時に実引数をコールスタックに追加し表示する表示方法である．図 5.9 に図 5.6 のプログラムの実行時の様子を示す．仮引数が実行された際，コール

5.4　プログラム実行中の引数ブロックの表示

スタックに実引数を追加し表示する．表示された実引数にハイライトを移すことで実行中のブロックを分かりやすくしている．この表示方法でも，作成したプログラム通りに見えるためデバッグしやすく，また最新のコールスタックのみを見るだけで実行中のブロックが分かるため，次に実行されるブロックの予想がつきやすい．

第 6 章

引数の表示方法の比較

本章では，プログラムの実行時の引数の表示方法の比較を行った．ユーザに使ってもらいどの表示方法が良かったかアンケートを取った．その後，結果をもとにそれぞれの引数の表示方法の考察を行なった．

6.1　引数の実行時の表示方法の比較

5 人のユーザに，それぞれの引数の実行の表示方法を実装した，関数を扱う機能を追加したアルゴロジックを使ってもらい，どの表示方法が一番分かりやすかったか調査した．5 人のユーザのうち，2 人は情報系の大学院生，1 人は情報系大学の学部 4 年生，2 人は情報系大学の学部 3 年生である．5 人ともプログラミングを得意としており，うち 3 人は本ソフトウェアの実装を担当した者である．

ユーザには，三つの課題を達成するまで使ってもらった．一つ目と二つ目のステージ（図 6.1，図 6.2）では，このツールのループや関数の使い方に慣れてもらうために簡単なステージを用意した．三つ目のステージ（図 6.3）で実際に，引数を使って課題に取り組んでもらった．

調査は，最初の問いで一番わかりやすかった表示方法を一つ選んでもらい，次の問いで自由にコメントを答えるアンケートで行った．一番わかりやすかった表示方法の回答を図 6.4 にまとめた．結果より，大学院生は 2 人ともスタック表示を学部生は 3 人とも戻り表示が良いと答えた．また，アンケートでもらったコメントを戻り表示，置換表示，スタック表示，その他の部類に分け，同じ意見をまとめて簡略化したものを表 6.1 にまとめた． 置換表示

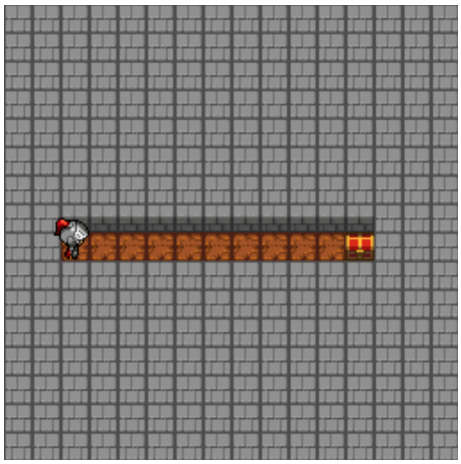

図 6.1　一つ目のステージ

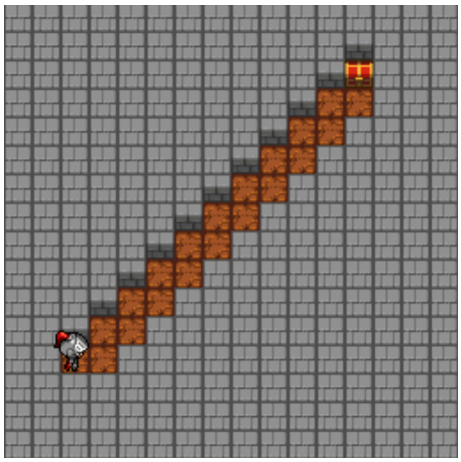

図 6.2　二つ目のステージ

は現在実行中のブロックが関数の本体なのか，仮引数を置き換えたものなのかわかりにくいというコメントが多かった．このため，置換表示はわかりやすいと思われなかったと考えられる．また同様に，スタック表現も新しく出現したものが何かわかりにくいと感じる人がいた．複数の引数を使いたかったというコメントもあった．

6.2　考察

図 6.4 より，プログラムの処理系を深く理解している人はスタック表示が分かりやすく，そうでない人は戻り表示が分かりやすいことが分かった．

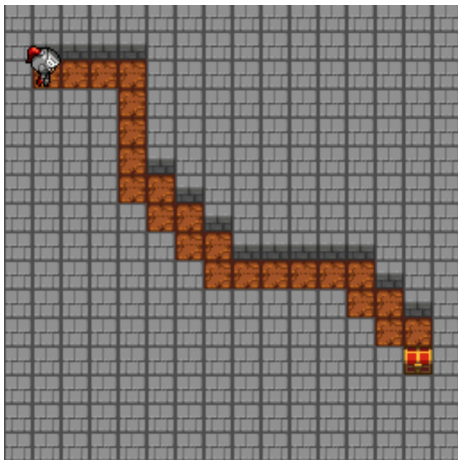

図 6.3　三つ目のステージ

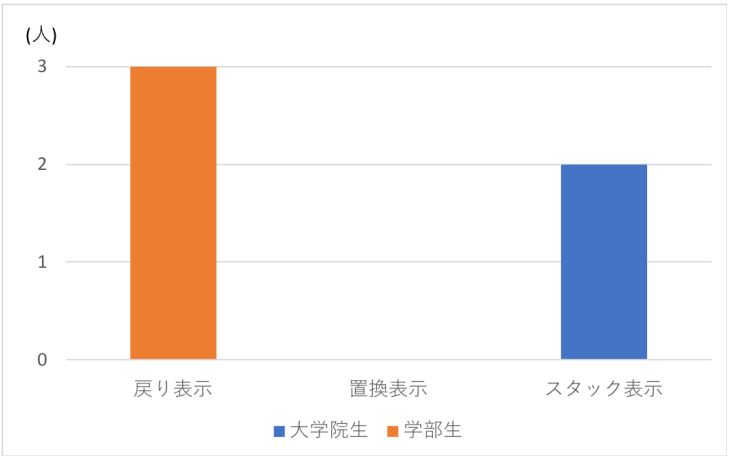

図 6.4　アンケートの結果

戻り表示は,「引数として与えられたプログラムがなんなのか分かりやすかった」というコメントより，どの関数呼び出しの実引数が実行されているのかが分かりやすいことが分かった．このように実行されている実引数がわかりやすいため，デバッグも容易になるのではないかと考える．しかし,「色を変えるだけだと見つけにくいと思った」というコメントがあった．例えば関数呼び出しの中で仮引数を呼び出した場合，図 6.5 の左部分のプログラムの二個目の実引数を実行後に，図 6.5 の右部分のように元のコールスタックに戻り次の関数を呼び出すため，ハイライトが遠くに移動して見失う可能性があることも分かった．

置換表示は，同じコールスタックの中に複数のハイライトが出ることがない．そのため，

6.2　考察

分類	コメント
戻り表示	引数として与えられたプログラムがなんなのか分かりやすかった
	色を変えるだけだと見つけにくいと思った
置換表示	今実行しているブロックが，関数内のものか，実引数なのか分からなかった
スタック表示	複数引数がある時なら有用に感じるが，引数が一つの場合あまり有用ではなかった
	新しくコールスタックに表示されたものがどこのプログラムなのか理解しづらい
	新しくコールスタックに出された方が注目しやすい
	コールスタックに表示されたプログラムと実引数のプログラムが対応しているということが分かりやすくして欲しい
その他	プログラムの実行の様子とキャラクターの動きが対応しているのでデバッグしやすかった
	複数の引数を使いたかった

表 6.1　アンケートでもらったコメント

常に最新のコールスタックのハイライトを見るだけでいいためキャラクターとプログラムの対応が取りやすい．しかしコメントより，図 5.6 のプログラムを実行した図 5.7 のように，関数本体の命令ブロックなのか，それとも仮引数を置き換えたものかわからない．これでは，デバッグの際に実行中の関数を修正するべきか，関数呼び出しを修正するべきかわからない．

スタック表示は，実行されるブロックが最新のスタックコールにあるため，ハイライトの移動が最小限ですむ．しかし,「新しくコールスタックに表示されたものがどこのプログラムなのか理解しづらい」というコメントより，コールスタックに表示された実引数がどの関数呼び出しのものかわかりにくいことがわかった．

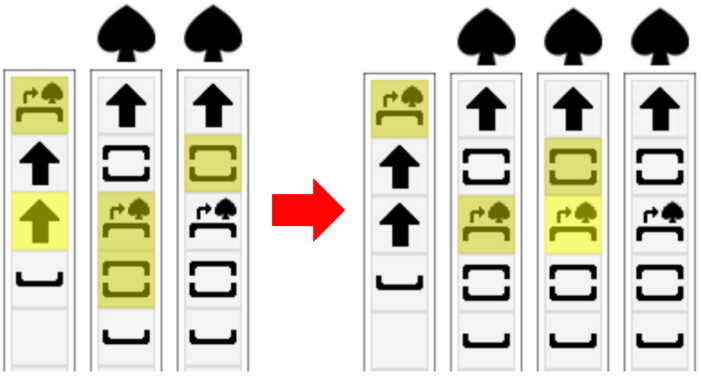

図 6.5　戻り表現のハイライトが遠くに移動する様子

戻り表示か，スタック表示のどちらかが分かりやすいことが分かった．しかし，改良したアルゴロジックの対象を小学生にした場合，戻り表示が良いことが分かった．置換表示のように完全に置き換えてしまうとデバッグの際に問題になる可能性があることが分かった．

第 7 章

おわりに

本研究では，アルゴロジックにモジュール化の機能として関数を追加した．各関数は引数を一つだけ渡すことができ，引数としてプログラムの列を渡すようにした．アルゴロジックに関数を追加するために，アルゴロジックをプログラミング言語として見たときの文法と意味を定義した．定義したアルゴロジックの文法と意味に関数呼び出しを追加した．さらにプログラムを実行した際の引数の実行の表示方法について，戻り表示，置換表示，スタック表示の 3 種類を考えた．この 3 種類の表示方法を実装したアルゴロジックを用いて試行錯誤しながらステージを解いてもらい，どの表示方法が一番わかりやすいかを調査した．その結果，戻り表示がわかりやすいことが分かった．また，複数の引数を使いたかったというコメントもあり，今後の課題として引数の数を検討する必要がある．

謝辞

本研究を行うにあたり，一からご指導していただいた本学情報学群鵜川始陽准教授に深く感謝致します．また，研究にご指導，ご指摘をいただきました本学情報学群妻鳥貴彦准教授に深く感謝致します．副査を引き受けてくださった先生方にも深く感謝致します．本研究に協力して頂いた本研究室のメンバーにも深く感謝致します．最後に，支えていただいた全ての方々に感謝致します．

参考文献

[1] 文部科学省，小学校プログラミング教育の手引き（第二版），2018

[2] 大山祐,「アルゴリズム体験ゲーム「アルゴロジック」」，情報処理，Vol.53，No.3，pp.316–320，2012

[3] Wing, J. M.,「Computational thinking」，Commun，ACM，Vol.49，No.3，pp.33–35，2006

[4] シーモア・パパート（1982）,「マインドストーム 子供，コンピューター，そして強力なアイデア」，奥村貴世子訳，未来社

[5] MIT メディアラボ，Scratch，https://scratch.mit.edu/

[6] 坂本一憲，本田澄，音森一輝，山崎頌平，服部真智子，松浦由真，高野孝一，鷲崎弘宜，深澤良彰,「まねっこダンス：真似て覚えるプログラミング学習ツール」，コンピュータソフトウェア，Vol.32，No.4，pp.74–92，2015

[7] Google，Blockly，https://developers.google.com/blockly/

[8] Google，Blockly Games，https://blockly-games.appspot.com/