2022年度 卒業論文

Scratchからの移行を意識したプログラミング言語の開発

指導教員　須田 宇宙 准教授

千葉工業大学 情報ネットワーク学科

須田研究室

1932034　氏名 川原 史哉

提出日　2023年1月17日

目次

図目次

表目次

プログラム目次

1　緒言

近年の急速な情報化に伴い，2020 年度から小学校ではプログラミング教育の必修化が行われた．これにより小学生でも扱いやすいビジュアルプログラミング言語が注目されている．代表的なビジュアルプログラミング言語に「Scratch」がある．「Scratch」は指示の書かれたブロックを並べることで視覚的にわかりやすく，直感的な操作でプログラミングができる．また，文字のタイピング量が少ない点や英単語の知識を必要としない点などから，初学者のプログラミング学習に適している．

本格的なプログラムを作成していくうえで，テキストプログラミング言語の学習が必要になる．しかし，Scratch に慣れていると，テキストプログラミング言語の学習を始めようとしたときに仕様の違いが挫折の要因になり得ると考えた．例えば，スプライトの有無やソースコードの記述方法などが挙げられる．

当研究室ではこれまで，上記の問題点を考慮したテキストプログラミング言語に関する先行研究が行われてきた．内堀は，ビジュアルプログラミングとテキストプログラミングについての調査を行い，初学者向けのテキストプログラミング言語の基礎を示した [1]．また，菅原は，開発するプログラミング言語の言語仕様を定め，インタープリタの開発を行った [2]．

しかし，先行研究の問題点として，開発するプログラミング言語の言語仕様はまとまりきっておらず，インタープリタのパーサーが正しく動作しないなどがある．

そこで本研究では，先行研究で開発されたプログラミング言語「Chiba-lang」の言語仕様をまとめることを目的とする．

2　小学校でのプログラミング教育について

2.1　概要

社会の情報化の進展により，コンピュータは人々の生活と密接に関わっており，様々な場面で活用されている．このような社会での職業生活や家庭生活，余暇生活などのあらゆる活動において，情報や情報技術を活用する能力が必要となる．

また，子どもたちが将来どのような職業になるかに関わらず，「プログラミング的思考」は必要となる．

以上の背景から，文部科学省は，学習指導要領改定において，小・中・高等学校を通じてプログラミング教育を充実することとし，2020 年度から小学校においてプログラミング教育の導入を決めた [3].

2.2　プログラミング的思考

「プログラミング的思考」とは，小学校のプログラミング教育において情報活用能力に含まれる育成されるべき資質・能力の一つで，文部科学省 (2020) では，

> 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，といったことを論理的に考えていく力

としており，プログラムの働きやよさ，情報社会が情報技術によって支えられていることに気づかせることを目的としている [3].

2.3　ビジュアルプログラミング言語

ビジュアルプログラミング言語とは，視覚的なオブジェクトでプログラミングをするプログラミング言語である．直感的な操作性や視覚的なわかりやすさから初学者のプログラミング教材として適している．指示の書かれたブロックを組み合わせるブロックタイプ，フローチャートのように，指示や機能，条件などのアイコンを線を繋ぐフロータイプ，独自のルールで作られた独自ルールタイプの三つに大別される．

ビジュアルプログラミングで代表的なものとして Scratch があり，小学校のプログラミング教育でも活用されている．

2.4　テキストプログラミング言語

テキストプログラミング言語とは，文字のみでソースコードを記述するプログラミング言語である．現在のソフトウェア開発では，テキストプログラミング言語が用いられている．

ビジュアルプログラミング言語と比較すると，直感的な操作性や視覚的なわかりやすさがないため，初学者にとっては難しく感じる可能性があると考えられる．

3　Chiba-lang のコンセプト

3.1　概要

本言語は，Scratch からテキストプログラミング言語への移行するときの仕様の違いを軽減するものであり，小学校で Scratch を学習した中学生が次の段階で学ぶことを想定している．そのため，以下のような言語コンセプトを定めた．

(1) Scratch と同様に，ステージ上でスプライトを動かすことができる．
(2) 中学で学ぶ範囲の英単語や日常で使われるような英単語を使用する．
(3) 上から下，左から右へと処理の流れがわかりやすい．
(4) C 言語における include 文のような不要要素を排除する．
(5) ブラウザ上での動作を前提とする．

(1) は，Scratch とテキストプログラミング言語の違いであるスプライトの有無についての問題点を軽減している．
(2) は，Scratch のブロックが日本語で表されているのに対して，テキスト言語の多くは英単語や記号で表現されるという差異を軽減している．本言語で使用されている英単語は中学生で習う英単語及び日常生活の中で使われるような英単語に限定することで抵抗感を軽減するという狙いがある．
(3) は，Scratch における視覚的なわかりやすさを本言語でも再現しようと試みている．
(4) における不要要素とは，C 言語における include 文のような動作のためには必要だが，プログラミングを学習するとき優先的に学ぶ必要がある事柄ではないものを指す．
(5) は，テキストプログラミング言語のような複雑な環境構築を必要としないことを目的としている．

3.2　参考言語

以下に本言語を作成する上で参考にした言語を示す．

3.2.1　Scratch

Scratch とは，指示の書かれたブロックを組み合わせるようにソースコードを記述することができるブロックタイプのビジュアルプログラミング言語である．特徴として，直感的な操作性や視覚的なわかりやすさが挙げられる．Scratch におけるステージやスプライトの概念や動作，図 1 に示したような実行環境をベースに本言語の言語仕様を定めた．

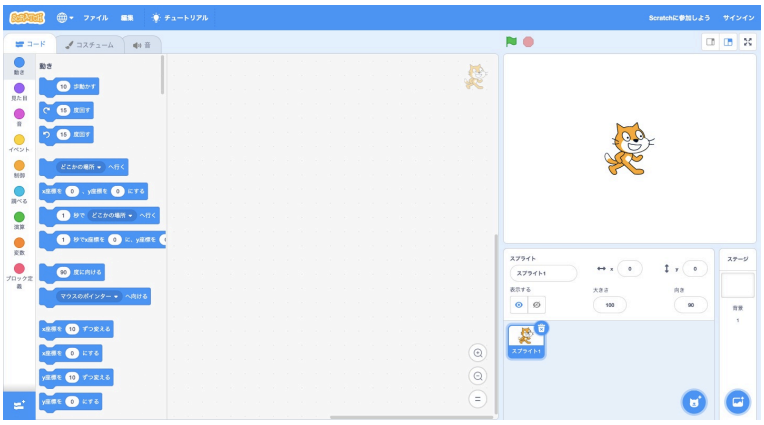

図 1　Scrach の実行環境

3.2.2　JavaScript

JavaScript とは，Web サイトやシステム開発などで用いられるテキストプログラミング言語である．JavaScript はとても汎用的なプログラミング言語で，HTML や CSS と組み合わせて使用することでブラウザ上での動的な処理を実現したり，Node.js などのブラウザ側だけでなくサーバー側でも使用される．

本言語では，イベント処理の実現のために promise と then の仕組みを参考にしている．

3.2.3　Ruby

3.2.4　Smalltalk

4　言語仕様

4.1　概要

前章において紹介したコンセプトや先行研究 [1, 2] をもとに，言語仕様を作成した．簡単な英単語や処理の流れのわかりやすさを重視している．

4.2　データ型

本言語におけるデータ型の種類について示す.

4.2.1　数値型

数値型は,「5」や「1.5」などの数値で表される．数値型には二種類あり，整数型と浮動小数点数型の二種類がある．また，数値に対して「-」を先頭につけることで負の数を表すことができる．プログラム 1 に数値型のプログラム例を示す.

プログラム 1　数値型のプログラム例

```
1 5 -> int;
2 -5 -> minus;
3 1.5 -> float;
```

4.2.2　文字列型

文字列型は，ダブルクォート「”」やシングルクォート「’」で囲まれた文字，数字，記号で表される．プログラム 2 に文字列型のプログラム例を示す.

プログラム 2　文字列型のプログラム例

```
1 "Hello, World!!" -> str1;
2 'こんにちは!' -> str2;
```

4.2.3　論理型

論理型は，真偽値である「true」および「false」で表される.

4.2.4　範囲型

範囲型は,「開始値 終端値」で表される．チルダ演算子「 」の左右の開始値と終端値には数値が入り，開始値から終端値までの範囲を生成する．繰り返し処理である each メソッドで使用される.

プログラム 3　範囲型のプログラム例

```
1 {1~10}.each ()=>{}
```

4.2.5 配列型

配列型は，角括弧「[]」の中身をカンマ「,」で区切ることで表される．プログラム 4 に配列型のプログラム例を示す.

プログラム 4　範囲型のプログラム例

```
1 [1,2,3,4,5] -> arr;
2 ["a",0,"b"] -> arr;
```

4.3 演算子

4.3.1 代入演算子

代入演算子は，右矢印「->」で表される．右矢印の左側のデータを右側の変数に代入することができる．プログラム 5 に代入演算子のプログラム例を示す.

プログラム 5　代入演算子のプログラム例

```
1 123 -> num;
2 "abc" -> str;
3 [1,2,3] -> arr;
```

4.3.2 パイプライン演算子

パイプライン演算子は，記号「|>」で表される．記号の左側の値を右側の関数の引数として実行する．関数を連続で実行する場合，左から優先的に実行される．パイプライン演算子を使用することでシンプルに記述することができる．プログラム 6 にパイプライン演算子のプログラム例を示す.

プログラム 6　パイプライン演算子のプログラム例

```
1 1 |> func;
2 {a, b} |> func1 |> func2;
```

4.3.3 チルダ演算子

チルダ演算子は，チルダ「 」で表される．記号の左側が開始値，右側が終端値の範囲型の値を生成する．プログラム 7 にチルダ演算子のプログラム例を示す.

プログラム 7　チルダ演算子のプログラム例

```
1 1~10;
2 0 -> start;
3 start~10;
```

4.3.4 算術演算子

算術演算子は，数値型の値を用いて算術演算を実行する．表 1 に算術演算子の一覧を，プログラム 8 に算術演算子のプログラム例を示す.

表 1　算術演算子一覧

演算子	名称
+	加算演算子
-	減算演算子
*	乗算演算子
/	除算演算子
%	余剰演算子

プログラム 8　算術演算子のプログラム例

```
1 10 + 10;
2 10 - 10;
3 10 * 10;
4 10 / 10;
5 10 % 10;
```

4.3.5 比較演算子

比較演算子は，二つの値の比較を行い，結果として真偽値を返す．表 2 に比較演算子の一覧を，プログラム 9 に比較演算子のプログラム例を示す.

表 2　比較演算子一覧

演算子	名称
=	等価演算子
<=	以下演算子
>=	以上演算子
<	小なり演算子
>	大なり演算子

プログラム 9　比較演算子のプログラム例

```
1 10 = 10;
2 10 <= 100;
3 100 >= 10;
4 10 < 100;
5 100 > 10;
```

4.3.6 論理演算子

論理演算子は，真偽値に対して論理演算を行う．表 3 に論理演算子の一覧を，プログラム 10 に論理演算子のプログラム例を示す.

表 3　論理演算子一覧

演算子	名称
and	AND 演算子
or	OR 演算子

プログラム 10　論理演算子のプログラム例

```
1 true and true;
2 true or false;
```

4.4 変数

変数を宣言する場合，必ず初期化が必要となる．代入演算子を用いて，データを変数に代入することで，変数を宣言することができる．プログラム 11 に変数のプログラム例を示す.

プログラム 11　変数のプログラム例

```
1 100 -> num;
2 () => {} -> func;
3 {10, 20} |> func -> ans;
```

4.5 関数

関数とは，一連の処理をまとめたものを表す．関数の宣言には，ラムダ式を用いる．括弧「()」内に引数を，中括弧「{}」内に処理を記述する．また，return で戻り値を指定する．関数は変数に代入することもできる．プログラム 12 に関数のプログラム例を示す.

プログラム 12　関数のプログラム例

```
1 (a,b) => { return a + b } -> add;
2 {10, 20} |> add;
```

4.6 クラス

クラスとは，オブジェクトの設計図に相当するものである．Splite クラスや Stage クラスなどが存在する．クラスは大文字から始まる．

4.7 メソッド

メソッドとは，あるオブジェクトに対する手続きのことを指す．「オブジェクト名. メソッド パラメータ」で表される．メソッドを使用することで，スプライトの動作，見た目の変更などの処理を行う．プログラム 13 にクラスとメソッドのプログラム例を示す．例では，Splite クラスにある new メソッドにより，cat 内に「ねこ」という名前のスプライトを生成している．また，walk メソッドを使用し，生成したスプライトを 10 歩進ませる処理を行なっている．

プログラム 13　クラスとメソッドのプログラム例

```
1 Splite.new name:"ねこ"-> cat;
2 cat.walk 10;
```

4.8 構文規則

5　Scratch との比較

Scratch のブロックは指示の種類に分類されており，「動き」，「見た目」，「音」，「イベント」，「制御」，「調べる」，「演算」等の種類がある．現時点では，全ての機能を本言語内で取り扱うことは想定しておらず，必要最低限の機能の実装を想定している．種類ごとに Scratch のブロックと本言語の記述の比較したものを示す．

5.1　動き

Scratch における「動き」に関連するブロックと本言語における類似した処理との比較を以下に示す．

5.1.1　walk メソッド

図 2 で示された Scratch における「〜歩動かす」ブロックは，スプライトの向いている方向への移動処理を行う．本言語では，walk メソッドにより同様の処理を行う．

walk メソッドは，「スプライト名.walk」で向いている方向へ，指定した数値分の移動を行う．プログラム 14 に walk メソッドのプログラム例を示す．

図 2 「〜歩動かす」ブロック

プログラム 14　walk メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.walk 50;
```

5.1.2　move メソッド

図 3 で示された Scratch における「座標を〜にする」ブロックは，スプライトの元座標から指定した座標への移動処理を行う．また，移動にかける時間も指定することができる．本言語では，move メソッドで同様の処理を行う．

move メソッドは，「スプライト名.move」で x 座標の値，y 座標の値，時間を指定し，指定された座標への移動処理を行う．プログラム 15 に move メソッドのプログラム例を示す．

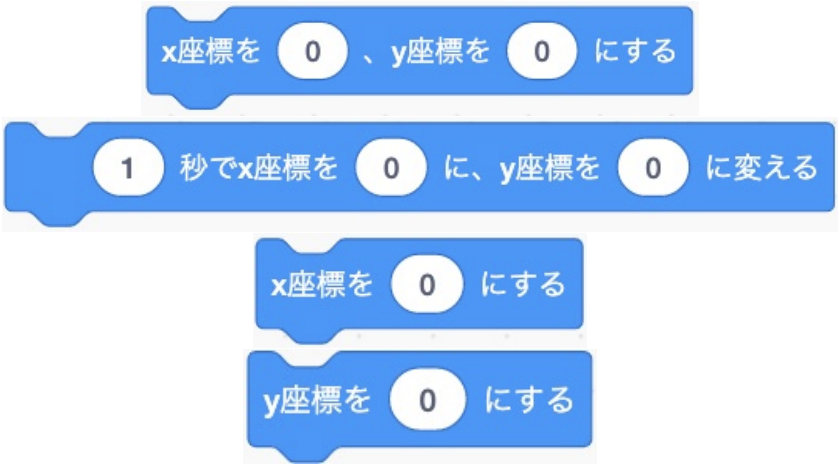

図 3 「座標を〜にする」ブロック

プログラム 15　move メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.move x:10, y:10, time:2;
```

5.1.3　move_step メソッド

図 4 で示された Scratch における「座標を〜ずつ変える」ブロックは，スプライトの元座標から指定した値分の座標移動を行う．本言語では，move_step メソッドで同様の処理を行う．

move_step メソッドは，「スプライト名.move_step」で x 座標の変化量，または y 座標の変化量を指定し，指定された値分の移動処理を行う．正負どちらの値も指定することができる．プログラム 16 に move_step メソッドのプログラム例を示す．

図 4 「座標を〜ずつ変える」ブロック

プログラム 16　move_step メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.move_step x:10;
3 cat.move_step y:-10;
```

5.1.4　turn メソッド

図 5 で示された Scratch における「〜度回す」,「〜度に向ける」ブロックは，スプライトの向きの変更を行う．本言語では，turn メソッドで同様の処理を行う．

turn メソッドは，「スプライト名.turn」で角度を変更することができる．向きは 0 度から 360 度まであり，0 度と 360 度が上向き，90 度が右向き，180 度が下向き，240 度が左向きを表す．正負の値どちらも指定することができる．プログラム 17 に turn メソッドのプログラム例を示す．

図 5 「〜度回す」,「〜度に向ける」ブロック

プログラム 17　turn メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.turn 90;
3 cat.turn -180;
```

5.1.5　return メソッド

図 6 で示された Scratch における「もし端に着いたら，跳ね返る」ブロックは，スプライトがステージの外枠に触れた時，スプライトの向きを反転させる．本言語では，条件分岐と touch メソッド，return メソッドで処理を再現する．

touch メソッドとは，二つのオブジェクト同士が接触しているかどうかを真偽値で返す Know クラスのメソッドである．また，return メソッドとは，接触しているステージの端に接触しているとき，スプライトの向きを逆にするメソッドである．プログラム 18 に return メソッドのプログラム例を示す．プログラム 18 では，ステージの外枠とスプライトが接触しているかどうかを判定している．そして，if 文で true が与えられた場合は，return メソッドで跳ね返り処理を行う．

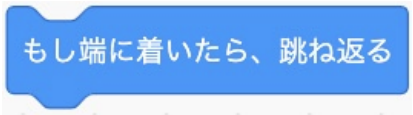

図 6 「もし端に着いたら，跳ね返る」ブロック

プログラム 18　return メソッドのプログラム例

```
1 Stage.new name:"いえ" -> stage;
2 Splite.new name:"ねこ" -> cat;
3 { cat.touch stage.around }.if {
4   cat.return;
5 };
```

5.1.6　x メソッドと y メソッド

図 7 で示された Scratch における「座標」ブロックは，あるスプライトが所持している座標情報を呼び出すことができる．本言語では，x メソッドと y メソッドで同様の処理を行う．

x メソッドと y メソッドは，「スプライト名.x」と「スプライト名.y」でそれぞれスプライトの x 座標の値と y 座標の値を呼び出す．プログラム 19 に x メソッドと y メソッドのプログラム例を示す．

図 7 「座標」ブロック

プログラム 19　x メソッドと y メソッドのプログラム例

```
1 Splite.new name:"ねこ", x:10, y:-10 -> cat;
2 cat.x -> num_x;
3 cat.x -> num_y;
```

5.2　見た目

Scratch における「見た目」に関連するブロックと本言語における類似処理との比較を以下に示す．

5.2.1　say メソッド

図 8 で示された Scratch における「〜と言う」「〜と考える」ブロックは，スプライトの頭上に吹き出しと台詞を表示させる．本言語では，say メソッドで「〜と言う」ブロックと同様の処理を行う．「〜と言う」ブロックと「〜と考える」ブロックの違いは吹き出しの形状のみなので，同一のものとして考えている．

say メソッドは，「スプライト名.say」で文字列のデータを与えることで，スプライトの頭上に吹き出しを表示する．プログラム 20 に say メソッドのプログラム例を示す．say メソッドは実行された場合，常に吹き出しが表示され続ける．そのため，吹き出しを消す場合は，「スプライト名.say ””」のように，空の文字列を与えることで上書きする必要がある．プログラム 20 では，「こんにちは！」を表示してから 3 秒後に吹き出しを消すという処理を行なっている．

図 8 「〜と言う」「〜と考える」ブロック

プログラム 20　say メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.say "こんにちは!";
3 { 3.second }.wait;
4 cat.say "";
```

5.2.2　costume メソッドと next_costume メソッド

図 9 で示された Scratch における「コスチュームを〜にする」ブロックは，スプライトの所持している「コスチューム」という差分イラストへの変更処理を行う．本言語では，costume メソッドと next_costume メソッドで同様の処理を行う．

本言語では，Splite クラスに事前にスプライトのイラストやコスチュームが用意されている．コスチュームにはそれぞれ番号と名前が割り当てられている．costume メソッドでは，「スプライト名.costume」で番号か名前を指定することで，コスチュームの変更を行う．

また，next_costume メソッドでは，「スプライト名.next_costume」で次の番号のコスチュームへの変更を行う．プログラム 21 に costume メソッドと next_costume メソッドのプログラム例を示す．

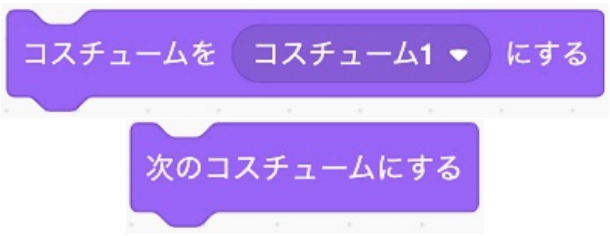

図 9 「コスチュームを〜にする」ブロック

プログラム 21　costume メソッドと next_costume メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.costume "コスチューム1";
3 cat.next_costume;
```

5.2.3　background メソッドと next_background メソッド

図 10 で示された Scratch における「背景を〜にする」ブロックは，ステージの所持している「背景」という差分イラストへの変更処理を行う．本言語では，background メソッドと next_background メソッドで同様の処理を行う．

本言語では，Stage クラスに事前にステージのイラストや背景が用意されている．背景にはそれぞれ番号と名前が割り当てられている．background メソッドでは，「ステージ名.background」で番号か名前を指定することで，背景の変更を行う．

また，next_background メソッドでは，「スプライト名.next_background」で次の番号のコスチュームへの変更を行う．プログラム 22 に costume メソッドと next_background メソッドのプログラム例を示す．

図 10 「背景を〜にする」ブロック

プログラム 22　background メソッドと next_background メソッドのプログラム例

```
1 Stage.new name:"ステージ" -> stage;
2 stage.background "ゆき";
3 stage.next_background;
```

5.2.4　size メソッド

図 11 で示された Scratch における「大きさを〜% にする」ブロックは，スプライトの大きさを変更する処理を行う．本言語では，size メソッドで同様の処理を行う．

size メソッドは，「スプライト名.size」で変更する大きさを指定する．標準サイズは 100% である．プログラム 23 に size メソッドのプログラム例を示す．

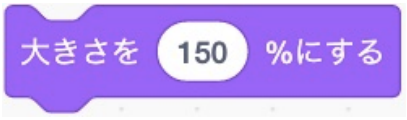

図 11 「大きさを〜% にする」ブロック

プログラム 23　size メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.size 150;
```

5.2.5　size_step メソッド

図 12 で示された Scratch における「大きさを〜% にする」ブロックは，スプライトの大きさを指定した値分の変更を行う．本言語では，size_step メソッドで同様の処理を行う．

size_step メソッドは，「スプライト名.size_step」で変化量を指定する．正負の値どちらも指定することができる．プログラム 24 に size_step メソッドのプログラム例を示す．

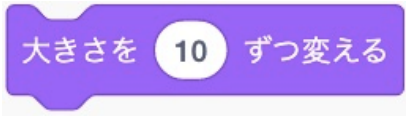

図 12 「大きさを〜ずつ変える」ブロック

プログラム 24　size_step メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.size_step 10;
```

5.2.6　show メソッド

図 13 で示された Scratch における「表示」ブロックはスプライトを表示し，「非表示」ブロックはスプライトを非表示にする．本言語では，show メソッドで同様の処理を行う．

show メソッドは，「スプライト名.show」に真偽値を与えることで，表示と非表示の変更を行う．true が与えられたとき，スプライトを表示し，false が与えられたとき，スプライトを非表示にする．プログラム 25 に show メソッドのプログラム例を示す．

図 13 「表示」「非表示」ブロック

プログラム 25　show メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.show true;
3 cat.show false;
```

5.3　イベント

Scratch における「イベント」に関連するブロックと本言語における類似処理との比較を以下に示す．図 14 で示された Scratch における「〜されたとき」ブロックは，イベント処理を表す．本言語では，KeyEvent クラスや ClickEvent クラスや MessageEvent クラスと if メソッドを使用することで処理を再現している．JavaScript における Promise を利用した非同期処理の仕組みを参考にしている．

プログラム 26 にイベント処理のプログラム例を示す．「ClickEvent.new」でクラスのインスタンス「ev」を作成している．ClickEvent クラスのインスタンスは，クリックしたオブジェクトのキーワードを受け取る．プログラム 26 では，旗が押されたときに発行される「flag」を受け取っており，if メソッドで flag を受け取ったときの処理を記述している．

KeyEvent クラスのインスタンスは，キーが押されたときにキーの名前を，ClickEvent クラスのインスタンスは，スプライトがクリックされたときにスプライトの名前をキーワードとして受け取る．

また，図 15 で示された Scratch における「〜を送る」「〜を送って待つ」ブロックは，Scratch プロジェクト全体にメッセージを送る．本言語では，メッセージを送る場合は，MessageEvent クラスのインスタンスにパイプライン演算子でメッセージを送る．

図 14 「〜されたとき」ブロック

図 15 「〜送る」「〜を送って待つ」ブロック

プログラム 26　イベント処理のプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 ClickEvent.new -> click;
3 MessageEvent.new -> mes;
4 { click = "flag" }.if {
5   cat.say "こんにちは!";
6   終了 |> mes;
7 };
```

5.4　制御

Scratch における「制御」に関連するブロックと本言語における類似処理との比較を以下に示す.

5.4.1　wait メソッド

図 16 で示された Scratch における「〜秒待つ」ブロックは，スクリプトの一時停止を行う.「〜を止める」ブロックは指定したスクリプトの停止を行う．本言語では，wait メソッドで「〜秒待つ」ブロックの処理を再現している.

wait メソッドは,「スプライト名.wait」で指定した時間分のスプライトの一時停止を行う．プログラム 27 に wait メソッドのプログラム例を示す.

図 16 「〜秒待つ」「〜を止める」ブロック

プログラム 27　wait メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 cat.wait 60;
```

5.4.2　each メソッド

図 17 で示された Scratch における「〜回繰り返す」ブロックは，指定した回数分の繰り返し処理を行う．本言語では，each メソッドで同様の処理を行う．

each メソッドは，「範囲型.each 処理」のように，範囲型で示された回数分，指定された処理を行う．プログラム 28 に each メソッドのプログラム例を示す．範囲型の開始値から終端値までが処理の引数に与えられる．

図 17 「〜回繰り返す」ブロック

プログラム 28　each メソッドのプログラム例

```
1 0 -> sum;
2 { 1~10 }.each (i) => {
3   sum + i -> sum;
4 };
```

5.4.3　while メソッド

図 19 で示された Scratch における「〜まで繰り返す」ブロックは，指定した条件が真になるまで繰り返し処理を行う．本言語では，while メソッドで類似した処理を行う．

while メソッドは，「条件式.while 処理」のように，条件式の結果が true である間，指定された処理を行う．プログラム 29 に while メソッドのプログラム例を示す．

図 18 「〜まで繰り返す」ブロック

プログラム 29　while メソッドのプログラム例

```
1 10 -> num;
2 { num > 0 }.while {
3   num - 1 -> num;
4 };
```

また，図 19 で示された Scratch における「ずっと繰り返す」ブロックは，無限ループを行う．while メソッドは，「true.while」とすることで，永続的な繰り返し処理を行う．「break」で無限ループから抜け出すことができる．プログラム 30 に無限ループのプログラム例を示す．

図 19 「ずっと繰り返す」ブロック

プログラム 30　無限ループのプログラム例

```
1 0 -> sum;
2 true.while {
3   sum + 1 -> sum;
4   { sum > 10 }.if {
5     break;
6   };
7 };
```

5.4.4　if メソッド

図 20 で示された Scratch における「もし〜ならば」ブロックは，指定した条件が真である場合のみ処理を行う．本言語では，if メソッドで類似した処理を行う．

if メソッドは，「条件式.if 処理」のように，条件式の結果が true である場合，処理を行う．プログラム 31 に if メソッドのプログラム例を示す．

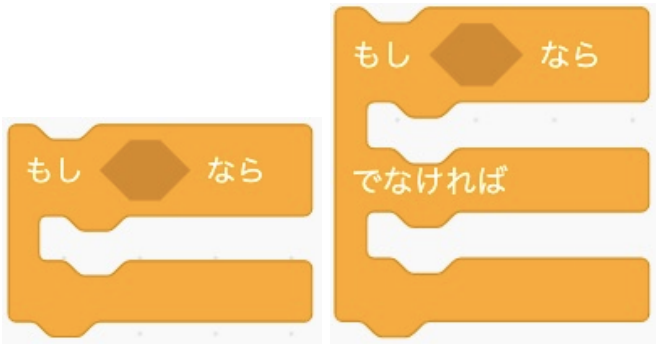

図 20 「もし〜ならば」ブロック

プログラム 31　if メソッドのプログラム例

```
1 3 -> num;
2 {num = 3}.if {
3   0 -> num;
4 };
```

5.5　調べる

Scratch における「調べる」に関連するブロックと本言語における類似処理との比較を以下に示す.

5.5.1　touch メソッド

図 21 で示された Scratch における「〜に触れた」ブロックは，スプライトがある要素に対して接触したかどうかの判定処理を行う．本言語では，touch メソッドで同様の処理を行う.

touch メソッドでは，接触したかの判定をしたい要素を touch メソッドに与えることで，その結果を真偽値で返す．プログラム 32 に touch メソッドのプログラム例を示す.

図 21 「〜に触れた」ブロック

プログラム 32　touch メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 Splite.new name:"いぬ" -> dog;
3 cat.touch dog;
```

5.5.2 length_between メソッド

図 23 で示された Scratch における「〜までの距離」ブロックは，スプライトとある要素との距離を返す．本言語では，length_between メソッドで同様の処理を行う．

length_between メソッドでは, 二つのスプライト間の距離を数値で返す. プログラム 33 に length_between メソッドのプログラム例を示す.

図 22 「〜までの距離」ブロック

プログラム 33 length_between メソッドのプログラム例

```
1 Splite.new name:"ねこ" -> cat;
2 Splite.new name:"いぬ" -> dog;
3 cat.length_between dog;
```

5.5.3 mouse_x メソッドと mouse_y メソッド

図 23 で示された Scratch における「マウスの座標」ブロックは，マウスのカーソルの座標の値を持っている．本言語では，mouse_x メソッドと mouse_y メソッドで同様の処理を行う．

mouse_x メソッドは，マウスの x 座標の値を呼び出す．mouse_y メソッドは，マウスの y 座標の値を呼び出す．プログラム 34 に mouse_x メソッドと mouse_y メソッドのプログラム例を示す.

図 23 「マウスの座標」ブロック

プログラム 34 mouse_x メソッドと mouse_y メソッドのプログラム例

```
1 kn.mouse_x -> num_x;
2 kn.mouse_y -> num_y;
```

5.6 演算

Scratch における「調べる」に関連するブロックと本言語における類似処理との比較を以下に示す.

5.6.1 算術演算

図 24 で示された Scratch における「算術演算」ブロックは，算術演算を行う．本言語では，表 1 に示された算術演算子を用いる．

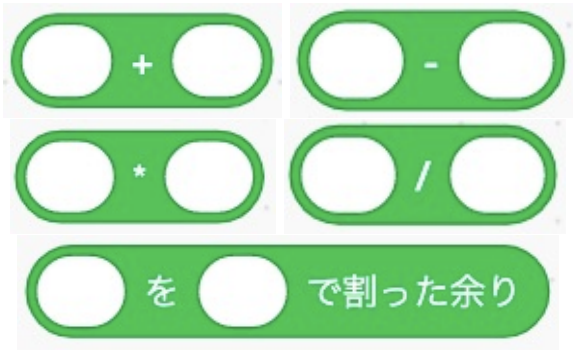

図 24 「算術演算」ブロック

5.6.2 比較演算

図 25 で示された Scratch における「比較演算」ブロックは，比較演算を行う．本言語では，表 2 に示された比較演算子を用いる．

図 25 「比較演算」ブロック

5.6.3 論理演算

図 27 で示された Scratch における「論理演算」ブロックは，論理演算を行う．本言語では，表 3 に示された論理演算子を用いる．

図 26 「比較演算」ブロック

5.6.4 乱数

図 27 で示された Scratch における「〜から〜までの乱数」ブロックは，指定された範囲内の乱数を生成する．本言語では，Random クラスによって同様の処理を行う．

「Random.new from: 開始値, to: 終端値」で，開始値から終端値までの乱数を生成する．プログラム 35 に乱数のプログラム例を示す.

図 27 「〜から〜までの乱数」ブロック

プログラム 35　乱数のプログラム例

```
1 Random.new from:1, to:10 -> rand;
2 rand -> num;
```

6　結言

近年の急速な情報化に伴い，2020 年度から小学校ではプログラミング教育の必修化が行われた．これにより小学生でも扱いやすいビジュアルプログラミング言語が注目されている．代表的なビジュアルプログラミング言語に「Scratch」がある．「Scratch」は指示の書かれたブロックを並べることで視覚的にわかりやすく，直感的な操作でプログラミングができる．また，文字のタイピング量が少ない点や英単語の知識を必要としない点などから，初学者のプログラミング学習に適している．

本格的なプログラムを作成していくうえで，テキストプログラミング言語の学習が必要になる．しかし，Scratch に慣れていると，テキストプログラミング言語の学習を始めようとしたときに仕様の違いが挫折の要因になり得ると考えた．例えば，スプライトの有無やソースコードの記述方法などが挙げられる．

当研究室ではこれまで，上記の問題点を考慮したテキストプログラミング言語に関する先行研究が行われてきた．内堀は，ビジュアルプログラミングとテキストプログラミングについての調査を行い，初学者向けのテキストプログラミング言語の基礎を示した [1]．また，菅原は，開発するプログラミング言語の言語仕様を定め，インタープリタの開発を行った [2]．

しかし，先行研究の問題点として，開発するプログラミング言語の言語仕様はまとまりきっておらず，インタープリタのパーサーが正しく動作しないなどがある．

そこで本研究では，Scratch を学んだ学習者が，次の段階として抵抗感を減らすことを目的としたテキストプログラミング言語「Chiba-lang」のコンセプトを定め，サンプルプログラムと言語仕様書の作成を行った．

今後は，本研究でまとめられた言語仕様より，インタープリタや実行環境の開発を進めていく．

7　謝辞

本論文の作成にあたり、多くの方々にご指導ご鞭撻を賜りました。須田浩准教授や須田研究室の仲間には,多大なる御指導及び御助言を頂きました．深く感謝申し上げます.

参考文献

[1] 内掘美幸: “Scratch からの移行を考慮したプログラミング言語と環境の開発”, 2019 年度卒業研究

[2] 菅原直輝: “Scratch からの移行を考慮したプログラミング言語の開発”, 2020 年度卒業研究

[3] 文部科学省, “小学校プログラミング教育の手引(第三版)”, (2020),https://www.mext.go.jp/content/20200218-mxt_jogai02-100003171_002.pdf