小学生によるゲームプログラミングでの テキスト型言語とビジュアル型言語の特徴比較

長谷部 竜司 [*1] , 香山 瑞恵 [*1]

[*1] 信州大学工学部

Quantitative comparison of game programs in visual based language and text-based language by japanese pupils

Tatusi Hasebe[*1] ,Mizue Kayama[*1]

[*1] Faculty of Engineering, Shinshu University

本研究の目的は, 小学生によるゲームプログラミングでのテキスト型言語とビジュアル型言語の特徴を比較することである. 同一ゲームに対するプログラムを, プログラム複雑度のメトリクス尺度と, ゲームの勝敗に影響するアイテム取得数の観点から特徴を整理した. その結果, 両言語でアイテム取得数に差がない一方, ビジュアル型言語ではプログラム複雑度が有意に低いプログラムを記述できることがわかった.

キーワード：プログラミング教育, ビジュアルプログラミング

1 はじめに

2020 年度から小学校でプログラミング学習が必修化することが文部科学省より通知された. 小学校プログラミング教育は, プログラミングを行うことを学習するのではなく, それらを手段として用い活用する力が求められている [(1)]. プログラミング学習教材の例として, Scratch を用いた研修資料が文部科学省から公開された [(2)]. その他, プログラミン [(3)] やプログル [(4)] などを用いた教材も存在する. 一方, 高等学校の情報 I 研修教材 [(5)] では Python のソースコードがプログラム例として示されている. また, Code.org[(6)] などのプログラミング学習コミュニティでは, Blockly[(7)], Python, Javascript といった多様なプログラミング言語が用いられている.

本研究では, Scratch やプログルに代表される, マウスを利用してプログラムを作成していくプログラミング言語をビジュアル型言語, Python や Javascrip に代表される, 文字入力によってプログラムを作成していくプログラミング言語をテキスト型言語と呼称する. 現状, 公開されている学習資料等の内容から, 小学生はビジュアル型言語, 高校生はテキスト型言語を用いた学習が中心となることが予想される. また, 中学校技術分野の学習指導要領解説 [(8)] では,「課題を解決するために, 適切なプログラミング言語」を用いることが示されている. すなわち, 教育課程や発達段階によってプログラミング学習に用いるプログラミング言語が異なることが予想される. しかし, 教育課程や発達段階とプログラミング言語との関係は, 共通の課題に対するプログラムを比較するといった方法では整理されていない.

2　研究目的

本研究の目的は, 小学生によるゲームプログラミングでのテキスト型言語とビジュアル型言語の特徴を比較することである. ここでのゲームプログラミングとは, 対戦型ゲームのプレイヤプログラムを作成することである. 小学生はこのゲームで勝利することを目的にプログラムの作成を行う. 2 つのプログラミング言語の特徴を, 対戦ゲームにおけるアイテム取得数と, プログラムの静的評価のメトリックス値とで比較する.

3　特徴比較の前提

3.1　プログラミング言語

本研究で比較するプログラミング言語は以下の 2 種とした. テキスト型言語は, C#を用い, 開発環境は Visual Studio とした. ビジュアル型言語は, Scratch を用い, 開発環境は Scratch 2 offline editor とした.

3.2　chaser ゲーム

chaser とは, マス目マップを移動しながらマップ上に存在するアイテムを取得していくターン制ゲームである [9]. ゲームフィールドは縦 17 横 15 マスのマップである. ゲームで利用されるマップの例を図 1 に示す. ゲームは 2 人のプレイヤ (図 1 中の C と H) による対戦であり, 下記に示す条件を満たしたプレイヤが勝利する.

- ターン終了時に取得アイテム (♥) が対戦相手より多い
- 対戦相手が上下左右動くことができない状態になる
- 対戦相手の上にブロック (☒) を置く
- 対戦相手がブロックの上に移動する
- 対戦相手がゲーム進行不能になる

プレイヤがアイテムを取得したとき, そのプレイヤが元いたマスにブロックが出現する. プログラムで利用する命令は 4 種類× 4 方向 (上下左右) と 1 種類の計 17 種類である. 5 種の命令を以下に示す.

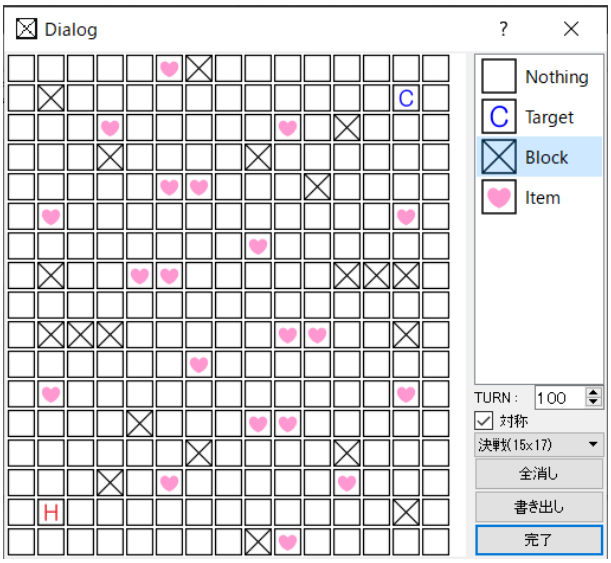

図 1　ゲームで利用されるマップの例

- walk :指定された上下左右どれかの方向に移動する
- look :指定された上下左右どれかの方向に対して, 正方形状に 9 マスの情報を取得する
- search :指定された上下左右どれかの方向に対して, 直線状に 9 マスの情報を取得する
- put :指定された上下左右どれかの方向に対して, ブロックを配置する
- getready :自分のターンが来るまでプログラムの実行を待機する

また上記命令を実行した際の返り値を参照するための変数として block がある.

3.3　ゲームプログラムの例

ゲーム中に上方向に移動し続けるプログラムの例を図 2, 図 3 に示す. テキスト型言語の例である図 2 では, ライブラリの指定も含め 23 行のプログラムである. 一方, ビジュアル型言語の例である図 3 は 6 行で記述されている.

4　静的評価のメトリックス

静的評価とはプログラムを実行せずに解析することである. ここではプログラムの静的評価のメトリックスとして, 循環的複雑度と Halstead's com-

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Drawing;
using CHaser;

namespace u15NaganoBot
{
class Program
{
    //接続用インスタンス
    private static Client Target = Client.Create();
    static void Main(string[] args)
    {
        while (true) //制御文
        {
            //GetReady 自分のターンまで待つ
            int[] value = Target.GetReady();
            value = Target.WalkUp();
        }
    }
}
}
```

図 2　C ＃でのゲームプログラム例

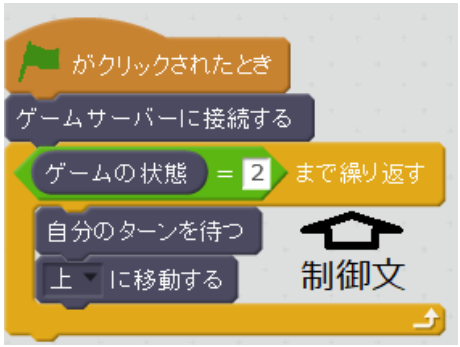

図 3　Scratch でのゲームプログラム例

plexity measures(以下, HCM) を取り上げる.

4.1　循環的複雑度

循環的複雑度とは, プログラムの複雑度を評価するために Thomas McCabe が開発したものである [10]. ソースコード内の線形的に独立した経路 (制御文) の数から算出される. 制御文とは, 条件分岐命令の if, 繰り返し命令の while や for などである. 循環的複雑度の求め方を以下に示す.

$$\text{循環的複雑度} = \text{制御文の総数} + 1 \tag{1}$$

図 2 と図 3 の制御文の総数は共に 1 であり循環的複雑度は 2 である.

4.2　Halstead's complexity measures:HCM

HCM は,「ソースコードが長くなるほど, 使用する変数や演算子の数や種類が増え, ソースコード全体が複雑になる」という考えのもと考案された尺度である [11]. 本研究では vocabulary, length, Volume, Difficulty, Effort を使用する[*1]. 本研究における 5 つの尺度の意味を以下に示す.

- h_vocabulary(n) :プログラム内の語彙 (演算子とオペランド) の種類数.

$$n = n_1 + n_2 \tag{2}$$

- h_length (N) :プログラム内の語彙 (演算子とオペランド) の総数.

$$N = N_1 + N_2 \tag{3}$$

- h_Volume (V) :プログラムのコーディングに必要な最小ビット数を表す.

$$V = N \times \log_2 n \tag{4}$$

- h_Difficulty(D) :語彙に基づく複雑度を表す.

$$D = \frac{n_1}{2} \times \frac{N_2}{n_2} \tag{5}$$

- h_Effort (E) :プログラムの開発や内容の理解に必要な作業量の推定値を表す.

$$E = D \times V \tag{6}$$

ただし, n_1=個別の演算子の数, n_2=個別のオペランドの数, N_1=演算子の総数, N_2=オペランドの総数である.

図 2,3 に示した C#と Scratch のプログラムにおける HCM の 5 つの尺度は図 1 に示される. この例では, C#の値が高い結果となった.

[*1] アルファベット表記の尺度はこれ以降「h_vocabulary」のように,「h_」が接頭語とする.

表 1　HCM 算出例

	C#	Scratch
h_vocabulary	36	8
h_length	71	8
h_Volume	367.06	24
h_Difficulty	6.54	3.5
h_Effort	2400.04	84
n_1, n_2, N_1, N_2	10, 26, 37, 34	7, 1, 7, 1

表 2　解析対象のプログラム言語とプログラム数

	C#	Scratch
2018	7(15)	–
2019	–	17(20)

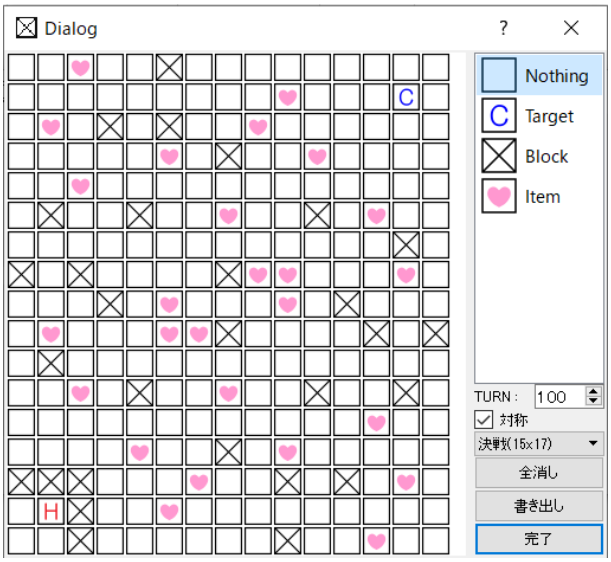

図 4　特徴比較で用いたマップ例 (マップ 2)

5　特徴比較

5.1　解析対象のプログラム

ここでは,2018 年と 2019 年のゲームプログラミングコンテスト[*2]に参加した小学生のプログラムを解析対象とした. このコンテストは小学 4 年生～中学 3 年生を対象としている. 小学生は 2018 年は 15 名, 2019 年は 20 名が応募した. 応募者は, 1 ヶ月の間 4 回の講習会に参加し, 複数回の個別指導に任意参加した. 利用したプログラミング言語は, 2018 年は C#, 2019 年は Scratch であった. 解析対象のプログラムは, これらの応募者が作成したプログラムの内, コンテストように提出し, かつ異常終了しないものとした. 表 2 に解析対象としたプログラム数を示す. () 内は応募者数である.

5.2　アイテム取得数

5.2.1　特徴比較のためのゲームのマップ

表 2 に示した 24 プログラムを 2 種のマップ (マップ 1, マップ 2) 上でボットプログラムと対戦させた. 対戦にあたっては, 先攻後攻の 2 パターンでそれぞれ 100 ターン動作させた. ボットプログラムとは, 図 4 中の H であり, 実験者がコンピュータ上に用意したプレイヤプログラムである. この対戦では, ボットプログラムは, アイテムを取得せず, ブロックの上に移動せず, 対戦相手に近づかない振る舞いをする. 特徴評価のために使用したマップの例を図 4 に示す.

マップ 1 にはアイテム 48 個, ブロック 8 個が配置されている. 解析対象のプログラムはアイテム取得後に出現するブロックを避け, アイテムを取得し続けることが求められる. マップ 2 にはアイテム 26 個, ブロック 29 個が配置されている. 解析対象のプログラムはブロックの上に移動せず, アイテムを取得し続けることが求められる.

解析対象プログラムのアイテム取得数の平均値と代表値の差の検定結果を表 3 上部に示す. /の右側には, マップに存在するアイテム数である. ここではデータサイズが小さく, 正規性の確認が難しいため, 代表値の差の検定には, Mann–Whitney の U 検定を用いた.

両言語においてマップ 2 よりマップ 1 のほうがアイテム取得数が多い. マップ 1 合計, マップ 2 合計, マップ 1,2 の総計いずれの代表値の差に関して Scratch プログラムと C#プログラムの間に有意差は確認できなかった.

5.3　静的評価のメトリックス値

静的評価のメトリクスとして, 循環的複雑度と HCM の代表値の差の検定を行う. ここではデータ

[*2] U-15 長野プログラミングコンテスト [12]

表 3　scratch と C#の代表値の差の検定結果

	要素	Scratch 平均値	C#平均値	p 値
アイテム取得数	マップ 1 合計	35.00/96	32.57/96	0.48
	(先攻, 後攻)	(16.50, 18.50)	(17.43, 15.14)	
	マップ 2 合計	13.61/52	13.14/52	0.35
	(先攻, 後攻)	(7.39, 6.22)	(5.86, 7.29)	
	マップ 1,2 総計	48.61/148	45.71/148	0.44
静的評価	循環的複雑度	55.06	44.14	0.17
	h_vocabulary*	27.44	81.00	$p < 0.05$
	h_length*	528.89	679.43	$p < 0.05$
	h_Volume*	2579.87	4331.39	$p < 0.05$
	h_Difficulty	71.57	45.12	0.16
	h_Effort	260382.96	217207.89	0.24

*：Scratch と C#の間で 5% 水準の有意差が確認された項目

サイズが小さく, 正規性の確認が難しいため, 代表値の差の検定には, Mann–Whitney の U 検定を用いた. 循環的複雑度と HCM の 5 つの尺度の値の平均値と検定結果を表 3 下部に示す. 循環的複雑度の代表値の差に関して Scratch プログラムと C#プログラムの間に有意差は確認できなかった. Scratch プログラムと C#プログラムの間で制御要素数の平均値に差はなかった. HCM に関しては h_vocabulary, h_length, h_Volume の代表値の差に関して両者の間に有意差が確認された. この 3 つの尺度に関して, Scratch プログラムの方が有意に小さい値を示した. 一方, h_Difficulty, h_Effort には有意差は確認されなかった. すなわち, Scratch の方が少ない語彙数, 少ない語彙総数, 少ない情報量でプログラムを記述していた. また, 語彙に基づく複雑度とプログラムの内容の理解に必要な作業量には差がない.

5.4　アイテム取得数と HCM の関係

HCM はプログラムの複雑さを表現するメトリクスである. アイテム取得数とプログラムの複雑さの関係を整理する. ここでは, ブロックがより多く配置されているマップ 2 に着目する. マップ 2 でアイテムを取るためにはブロックを避けつつアイテムを取るという戦略が必要となるため, 複雑なプログラムが有利であると予測される.

マップ 2 のアイテム取得数と HCM の相関係数を表 4 に示す. C#では, アイテム取得数と HCM には負の相関傾向が確認された. h_vocabulary には負の相関が有り, 他の尺度には弱い相関が確認された[*3].

表 4　マップ 2 のアイテム取得数と HCM との相関係数

HCM ＼使用言語	C#	Scratch
h_vocabulary	-0.78	0.67
h_length	-0.62	0.62
h_Volume	-0.64	0.64
h_Difficulty	-0.53	0.46
h_Effort	-0.60	0.55

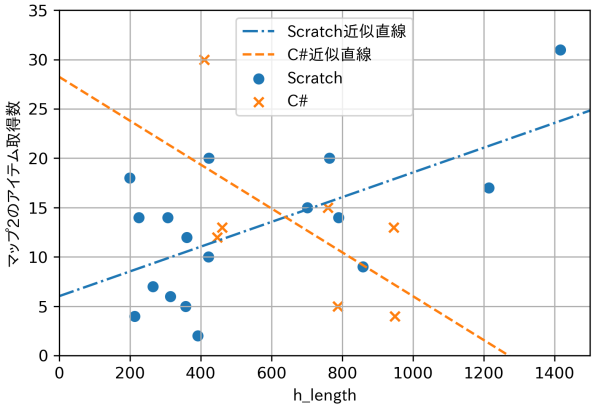

図 5　h_length とアイテム取得数

Scratch では, アイテム取得数と HCM には正の相関傾向が確認された. h_vocabulary, h_length, h_Volumen には高い正の相関が有り, h_Difficulty,

[*3] 一般に 7 データでの相関係数は, 0.67 より大きい場合, 10% 水準で有意であり, 相関があると判断される. 0.81 より大きい場合, 5% 水準で有意となり高い相関があるとされる.

h_Effort には正の相関が確認された[*4].

6 考察

表 3 の結果から, Scratch ではアイテムを取得する機能が C#と同程度なプログラムをより少ない語彙量で記述していた. よって, Scratch はゲームプログラムに必要な語彙量をアイテム取得数に影響を与えることなく, 少なくできる. 表 4 の結果から, Scratch は語彙の種類, あるいは語彙総数が増えるとアイテム取得数が増える. 一方 C#は語彙の種類, あるいは語彙総数が増えるとアイテム取得数が減る.

図 5 に HCM の h_length とアイテム取得数をそれぞれ軸にとった散布図を示す. ○のプロットは Scratch, ×のプロットは C#のプログラムを示す. また, 図 5 には各言語に対する近似直線を示す. 破線が Scratch で, 点線が C#である. この近似直線の傾きは表 4 の相関係数となる. Scratch で最多のアイテム取得数を示すプログラムの h_length は最大である. 一方, C#で最多のアイテム取得数を示すプログラムの h_length は最小である. また, マップ 2 の平均アイテム取得数は Scratch は約 14, C#は約 13 である. これらの個数以上のアイテムを取得するプログラムの h_length の範囲は, Scratch では 199〜1416 であり, C#では 410〜945 である. Scratch では, 最小の h_length であっても平均数以上のアイテムを取得するプログラムが存在している. すなわち, 多様な h_length でアイテムをより多く取得するプログラムを表現できていたと考えられる.

h_vocabulary と h_length から求めた各語彙の平均利用回数は Scratch は 19.27 回, C#では 8.38 回である.Scratch は C#の 2.4 倍となった. 図 6 と図 7 にプログラム例を示す.

Scratch ではゲームの状態を判別するために, 条件分岐が繰り返し利用されていることが多かった (図 6 参照). その際, 条件文には上下左右のマスが利用

図 6　Scratch のプログラム例

され, 条件が真 (, あるい偽) の場合の処理にはゲームの命令が記述される. そのため, 特定語彙の利用回数が多くなったと考えられる. 一方,C#では, 様々な変数が用いられていた. それらの変数はマップの状態や適用戦略の優先順位を表現していた (図 7 参照). これらの変数の評価や代入がプログラムに記述されていた.

これらのことから, 小学生のゲームプログラミングではテキスト型言語よりビジュアル型言語の方が少ない記述量で課題解決を行えていた可能性が示唆された. また, ゲームに勝利するという課題解決の手段として, アイテム取得を行うと考えられる. C#では, アイテム取得数が平均以上のプログラムと平均未満のプログラムでの h_length の範囲はほぼ同じである. そのため, アイテム取得数の違いはプログラムの戦略の違いに関係していると考えられる. プログラムの質的解析は今後の課題とする.

[*4] 一般に 17 データでの相関係数は, 0.41 より大きい場合, 10% 水準で有意であり, 相関があると判断される. 0.58 より大きい場合, 5% 水準で有意となり高い相関があるとされる.

```
前略
else if (value[5] == 1)
{
    putmode = "Rigft";
}
else if (value[7] == 1)
{
    putmode = "Down";
}
else
{
    putmode = "off";
}
if (putmode == "Up")//putmode の方向に put
{
    value = Target.PutUp();
}
else if (putmode == "Left")
{
    value = Target.PutLeft();
}
後略
```

図 7　C#のプログラム例

7　おわり

本研究の目的は, 小学生によるゲームプログラミングでのテキスト型言語とビジュアル型言語の特徴を比較することである. 2 つのプログラミング言語の特徴を比較するための指標は, 対戦ゲームにおけるアイテム取得数と, プログラムの静的評価のメトリックス値とした. 比較の結果, ビジュアル型言語はアイテムを取得する機能がテキスト型言語と同程度なプログラムをより少ない語彙量で記述できていたことがわかった. また, ビジュアル型言語での語彙の平均利用回数はテキスト型言語の約 2.4 倍であった. これらから, 小学生ではビジュアル型言語はテキスト型言語に比べ少ない記述量でプログラムを作成し, 課題解決を行えていたことが示唆された.

今後は, 中学生や高校生に対象を広げ, 各教育課程におけるビジュアル型言語及びテキスト型言語の特徴の比較を行い, 教育課程や発達段階とプログラミング言語との関係を, 共通の課題に対するプログラムを比較するといった方法で整理を行っていく. また, プログラミングされたアルゴリズムの具体的な違いの特徴比較を行っていく.

参　考　文　献

(1) 文部科学省 : “小学校プログラミング教育に関する概要資料,” http://www.mext.go.jp/component/amenu/education/microdetail/icsFiles/afieldfile\newline/2019/05/21/1416331001.pdf (2020/01/27 アクセス).

(2) 文部科学省 : “小学校プログラミング教育に関する研修教材,” http://www.mext.go.jp/component/amenu/education/microdetail/icsFiles/afieldfile\newline/2019/05/21/1416331001.pdf (2020/01/27 アクセス).

(3) 文部科学省 : “プログラミン,” https://www.mext.go.jp/programin/ (2020/01/27 アクセス).

(4) 特定非営利活動法人みんなのコード : “プログル,” https://proguru.jp/ (2020/01/27 アクセス).

(5) 文部科学省 : “高等学校情報科「情報 1」教員研修用教材（本編）第 3 章コンピュータとプログラミング,” https://www.mext.go.jp/component/a_menu/education/micro_detail/__icsFiles/afieldfile/2019/10/09/1416758_005.pdf (2020/01/27 アクセス).

(6) Code.org: “Code.org,” https://code.org/ (2020/01/27 アクセス).

(7) Google Developers: “Blockly,” https://developers.google.com/blockly (2020/01/27 アクセス).

(8) 文部科学省 : “中学校学習指導要領解説技術・家庭編,” https://www.mext.go.jp/component/a_menu/education/micro_detail/__icsFiles/afieldfile/2019/03/18/1387018_009.pdf (2020/01/27 アクセス).

(9) U-16 旭川プログラミングコンテスト実行委員会 : “コンテスト内容,” http://www.procon-asahikawa.org/contest.html (2020/01/27 アクセス).

(10) T.J. McCABE: “A complexity measure,” IEEE Tran. on Software Engineering, vol.SE-2, no.4, pp.308–320, (1976).

(11) Halstead: Elements of Software Science, edition edition, North-Holland, (1977). New York.

(12) 長野商工会議所 : “U15 長野プログラミングコンテスト ホームページ,” https://www.nagano-cci.or.jp/u15procon/ (2020/01/27 アクセス).