共同プログラミングを実現する初心者向けビジュアルプログラミングツール Jointry の開発と評価

角 征典[1,2]　金本 真左也[1]　Antoine Bossard[1]　秋口 忠三[1]

概要：
本研究では，複数人での共同プログラミングを実現する初心者向けビジュアルプログラミングツール Jointry の開発と評価を行った．近年，プログラミング教育への関心が高まっており，なかでも初心者に優しいビジュアルプログラミングツールに注目が集まっている．しかし，これまでの研究では，初期段階に初心者が独学でプログラミングを学習することは依然として難しく，先生や友達からの支援が必要であることがわかっている．そこで我々は，複数人で共同でプログラミングできるツールを開発し，プログラミング初心者を対象にして，質的な評価を行った．その結果，プログラミング初心者が初期段階に抱える不安は，複数人での共同プログラミングによって解消できる可能性があることがわかった．

キーワード：共同プログラミング, ビジュアルプログラミング, 初心者向けプログラミング環境, M-GTA

Development and Evaluation of Jointry, a Cooperative Visual Programming Tool for Novices

Abstract: In this research, we develop and evaluate a cooperative visual programming tool for novices entitled Jointry. Recently, interest in programming education has increased, and visual programming tools for novices are attracting public attention. However, previous studies have shown that learning programming on its own still is not satisfactory especially in an early stage, so that novices need help from their teachers or friends. Therefore we developed a tool which enables programming with multiple people, and evaluated it with novices by a qualitative method. As a result, we found that anxiety they faced in the early stage can be relieved by the solution of cooperative programming with more than one person.

Keywords: cooperative programming, visual programming, novice programming environment, M-GTA

1. はじめに

この数年でプログラミング教育の動きが活発化している．アメリカでは，NPO 法人 Code.org が作成したプログラミング教育の推進ビデオに多数の著名人が出演したことで話題となった [1]．エストニアの公立学校では，プログラミング教育を小学校一年生から開始すると発表されている [2]．イギリスでも 2014 年 9 月から導入する義務教育のカリキュラムのために，教師を対象にしたプログラミング教育訓練を開始すると発表されたばかりである [3]．また，日本政府が 2013 年に発表した「成長戦略（素案）」には，「義務教育段階からのプログラミング教育等の IT 教育を推進する」との記述がある [4]．プログラミング教育の重要性は今度もますます増していくだろう．

こうした状況において，マウスやタッチ操作でプログラムを開発できる「ビジュアルプログラミング」ツールに注目が集まっている．ビジュアルプログラミングの定義は時代とともに変化しているが [5]，本稿では，グラフィカルに表現された要素を使い，ユーザーが視覚的にプログラミングすることを指す．ビジュアルプログラミングツールのな

[1] 産業技術大学院大学
Advanced Institute of Industrial Technology
[2] ワイクル株式会社
Waicrew Inc.

かで特に人気があるのは，MIT メディアラボが開発した Scratch[6] である．日本でも 2013 年に書籍が出版され [7]，小学生向けのワークショップが各地で開催されている．

日本の学校でも Scratch を使った授業が行われており，良好な結果を残している．たとえば，小学生を対象に行われた授業では，受講者のほとんどが「楽しかった」（5 点満点で平均 4.78，SD:0.76）と高く評価している [8]．プログラミング未経験の文系大学生を対象に行われた授業でも，「すごく楽しかった」や「色々な欲が出てきた」などの肯定的な感想が述べられている [9]．

Scratch の作者であるミッチェル・レズニックは，ユーザーに受け入れられるための施策として 3 つの設計原則を設定している．そのひとつが「more social」である [10]．ユーザーは Scratch で作った作品をウェブサイトで共有できるようになっており，そこで他人の作品に投票したり，コメントをつけたり，その作品を元にした新しいリミックス作品を作ったりできる．このように「ソーシャル」であることが，プログラミング初心者に広く受け入れられ，モチベーションの向上や維持に一役買っていると言える．事実，一日に 1,500 件以上ものプロジェクトが新規にアップロードされており，全体の 15 %以上がリミックス作品である．

しかし，Scratch で作った作品をウェブで共有するところまでたどりつけず，初期段階でプログラミングにつまずく生徒もいる．先の事例でも，「初めの学習では苦戦」する小学生や，「1 日目はなかなか理解できなかった」文系大学生がいた [8], [9]．最終的な評価は高いものの，開始時の理解の難しさがプログラミングを始める大きなハードルになっていると考えられる．

そこで本研究では，ビジュアルプログラミングを始めるときの苦痛や不安を取り除き，初心者が「1 日目から理解できる」ような新しい形の「ソーシャル」なビジュアルプログラミングツール Jointry を開発した．Jointry は，複数人がそれぞれの画面を使い，お互いに協力して共通のプログラムを作成する「共同プログラミング」の支援機能に特徴がある．Jointry の有効性を評価するために，実際に複数人の初心者に 4 つの課題について共同プログラミングを実施してもらい，その様子を観察した．そして，その観察の結果をもとに，共同プログラミングの有効性を評価した．

2. 開発方針

Jointry を開発するにあたり，プログラミング初心者が考える苦痛や不安を取り除く方法を考えた．プログラミングを阻害する要因には，大きく「Mechanical Barriers」（仕組みの障害）と「Sociological Barriers」（社会的な障害）の 2 つがあると考えられている [11]．それぞれの項目を検討し，Jointry の設計原則および機能の参考とした．

2.1 仕組みの障害

初心者向けプログラミング言語には，プログラミング環境の仕組みの難しさにとらわれることなく，プログラミングの本質（ロジックや構造など）に集中できるような工夫が必要である．Scratch では，文字列の代わりにブロックというビジュアル要素を使い，これを解決している．あるいは，プログラムの構文を話し言葉（日本語）のようにするという取り組みもある [12]．非プログラマでもプログラミングができるように，特定のドメインに特化した言語（DSL：ドメイン特化言語）というものも存在する [13]．いずれもプログラミング言語の構文や操作を簡略化することで，仕組みの障害を排除しようとしている．

ただし，最終的な目的が「プログラミングの学習」なのであれば，その後にテキストベースのプログラミング言語へと移行ができるような仕組みも合わせて考える必要がある．たとえば，ビジュアルプログラムから Java のコードを出力する方法が有効だろう [14]．

以上を考慮して，Jointry では Scratch などのブロックプログラミングの要素を踏襲し，さらにブロックから文字列のコードを出力できる機能を取り入れることにした．

2.2 社会的な障害

社会的な障害については，学習者のモチベーション維持が大きな課題となっている．そのための工夫としては，ゲーム性を取り入れたもの [15]，実体を伴ったもの [16]，その実体をロボットにしたもの [17] などがある．

ゲーム性を取り入れたものに関しては，Scratch のもうひとつの設計原則である「more meaningful」（自分好みのプログラムを作れる）を満たしていないので，今回は採用を見送ることにした．

実体を伴ったものに関しては，「プログラムが表示される CRT は複数人で共有することに適さない表示装置」[16] であるとして，共同作業と試行錯誤の重要性が提唱されている．先の小学生を対象にした研究においても「友達や先生から教えてもらってだんだんとわかってきてスクラッチを楽しむことができました」[8] との感想が述べられているように，複数人で協力しながらプログラミングを学習することで，学習者のモチベーションの維持につながることが確認できる．

Jointry では，この「共同作業」に Scratch の「ソーシャル」の設計原則を採用すれば，プログラミング初心者の不安を取り除くことができるのではないかと考えた．これはいわば，ペアプログラミング [18] の機能である．ビジュアルプログラミングにペアプログラミングの機能を統合したものを調査したが見つからなかったため，これを Jointry の中心的な機能と位置づけた．

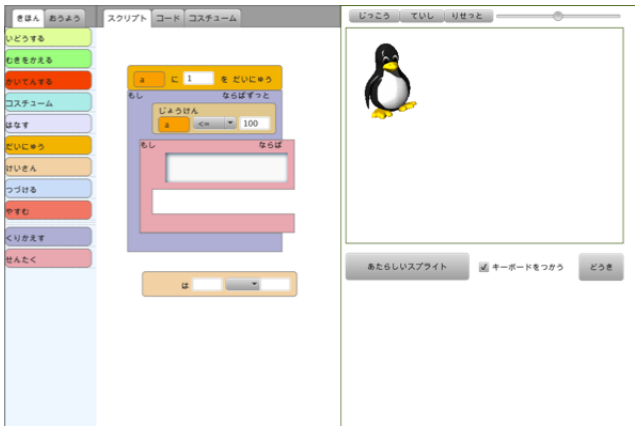

図 1 Jointry のスクリーンショット
Fig. 1 A screenshot of Jointry.

3. Jointry の開発

Jointry の開発は，産業技術大学院大学（以下，AIIT）の PBL（Project Based Learning）プロジェクトの一貫として，2013 年の 6 月から 12 月までの約 5 か月間をかけて行った（夏休みを除く）．

Jointry のスクリーンショットを**図 1** に示す．左端がブロックのメニュー，中央が選択したブロックを配置してプログラムを組み立てる場所，右端が組み立てたプログラムでスプライト（キャラクタ）を動かす場所である．

3.1 開発環境とアーキテクチャ

開発に使用した環境やツールは以下のとおりである．

- Java 7
- JavaFX 2.2
- NetBeans 7.4
- JavaFX Scene Builder 1.1

ソースコードの配置は，標準的な JavaFX アプリケーションの構成を参考にした．最初に，JavaFX Scene Builder を使って大まかな画面を設計し，それを FXML と呼ばれる XML ファイルに保存する．FXML には JavaFX のコントローラクラスを設定できるようになっているので，ユーザーからの操作を処理するメソッドをそれぞれのアクションに関連づける．以上で，基本的な GUI 操作が可能となる．FXML およびコントローラのファイルはすべて `controllers` パッケージに配置した．

コントローラの処理が大きすぎる，あるいは共通化できる場合は，独自のクラスを追加して処理を委譲することで，コードの見通しを改善した．これらは `services` パッケージに配置した．

ビジュアルプログラミングの「ブロック」などは `JavaFX` には存在しないため，標準ライブラリを継承しながら，独自にクラスを定義する必要がある．これらは `models` パッケージに配置した．

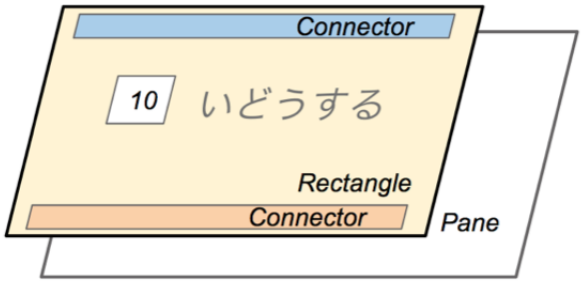

図 2 ブロックの構造
Fig. 2 Block structure.

3.2 ブロックとスクリプト

「ブロック」の部分については，`JavaFX` の標準ライブラリである `Pane` クラスや `Rectangle` クラスを利用した独自の `Block` クラスを作成した（**図 2**）．これがすべてのブロックの親クラスとなる．

ブロックを移動して他のブロックと重なったときは，ブロック同士を接続する必要がある．それぞれのブロックには，他のブロックと接続するための `Connector` クラスを配置している．この `Connector` 同士が衝突検知を行い，構文的に接続可能であれば，ブロックの接続を許可する．

それぞれのブロックは，`Jointry` 独自のプログラム言語の構成要素を持っている．たとえば，10 歩移動するブロックであれば，「`move 10;`」を持っている．これは文字列として保持されており，ブロックを接続すると，関連するブロックの文字列が結合される．結合された文字列は，`Jointry` のプログラムとして［コード］タブで確認できる．これが第 2 節で述べた「ブロックから文字列のコードを出力できる機能」である．

3.3 スクリプトとインタプリタ

`Jointry` は，独自のスクリプト言語と，それを解釈するインタプリタを内蔵している．それぞれの開発においては，`Stone` 言語 [19] の作り方を参考にした．ただし，ブロックプログラミングに最低限必要とされるものに機能を制限しており，通常のプログラム言語であれば必要となる「配列」や「関数」の機能は実装していない．

その代わり，ブロックプログラミングに特化した `move`（移動する）や `rotate`（回転する）などの命令が存在する．命令とブロックは一対一に対応しているわけではなく，ブロックにスクリプト言語を記述できるようにしている（第 3.2 項参照）．そのため，「移動しながら回転する」というような複数の命令を実行するブロックも作成可能である．

3.4 共同プログラミング

複数人がそれぞれの画面を使い，お互いに協力して共通

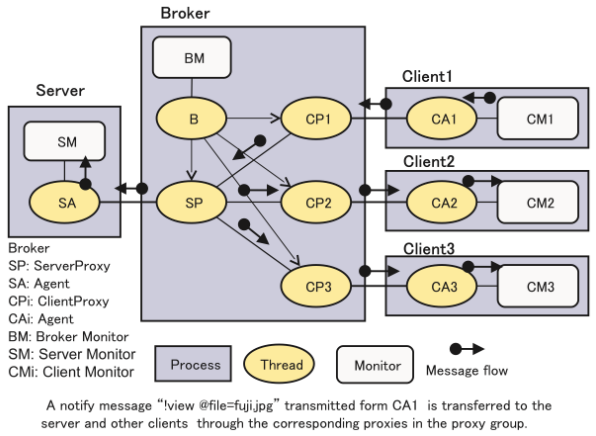

図 3 NSBroker による双方向通信 [20]
Fig. 3 NSBroker enables bidirectional communications [20].

図 4 評価の様子
Fig. 4 Experimental evaluation setup.

のプログラムを作成することを本稿では「共同プログラミング」と呼ぶ．

Jointry では，秋口の開発した NSBroker（Network Service Broker）[20] を使用して，この共同プログラミングを実現した．NSBroker は，インターネット通信におけるグループ活動を支援する通信基盤ライブラリであり，HTTP をベースにした独自のプロトコルを使用して，双方向通信を実現するものである．

Jointry では，「代表者」がプログラミングの「ルーム」を作り，そこに「協力者」が接続するという概念モデルを採用している．それぞれが**図 3** の「Server」，「ServerProxy」，「Client」に該当する．ユーザーが GUI を操作したタイミングで通信を開始し，同一の「ルーム」に所属するユーザー（「代表者」，「協力者」）すべての画面に同じ操作を反映する．

通信データについては，JSON（JavaScript Object Notation）を採用した．画像などのバイナリデータも Base64 でテキストデータにエンコードして，JSON の値として送信している．

4. システムの評価

4.1 評価依頼と協力者

AIIT の 2014 年度の 2 年生 3 人を対象にして，システムの評価を行った．いずれもプログラミング初心者を自認している．

- A：20 代，男性，プログラミング経験 1 年未満
- B：20 代，男性，プログラミング経験 1 年未満
- C：20 代，男性，プログラミング経験 3 年未満

A，B に関しては，AIIT 入学後にプログラミングの授業を数コマ受講した経験はあるが，単独でのプログラミングは難しい．C に関しては，C 言語と Java のプログラミング経験はあるが，十分に習得できたプログラミング言語はない．なお，3 人とも本格的なアプリケーションの開発経験はない．

それぞれにビジュアルプログラミングツールの評価に協力してほしい旨をメールで連絡し，内容に同意した上で評価に協力してもらった．また，当日（2013 年 4 月 19 日）に年代，性別，所属情報，評価風景を公開する可能性があること，評価中の発言はすべて録音することを口頭で説明し，いずれも同意を得た．録音機材については，A より借り受けたものを使用し，音声データをコピーしてから，評価終了後に返却した．

4.2 評価手順

こちらで Jointry を使用した問題を 4 問ほど用意しておき，協力者に以下の手順で 1 問ずつ解いてもらった．

(1)スプライトの動きを見る
(2)スプライトの動きを再現するブロックを組み立てる

1 問目に関しては個人作業で行ってもらい．残り 3 問に関しては共同プログラミング機能を使用して，A と B を中心に解いてもらった．C はプログラミング経験が長いため，サポート役をお願いして，A，B の 2 人よりも先に回答しないようにしてもらった．評価の様子を**図 4** に示す．

問題の詳細は，以下のとおりである．それぞれ 5〜15 分間の制限時間を設けた．

(1)スプライトが 3 歩だけ歩く［順次］
(2)スプライトがずっと歩き続ける［反復］
(3)スプライトがずっと歩き続けるが，壁にあたったら反転する［条件］
(4)スプライトがずっと歩き続けるが，3 回壁にあたったら止まる［変数］

3 人には問題を解く過程で考えていることを自由に話しながら，Jointry を操作してもらった．これは「思考発話法」と呼ばれるユーザーテストの手法である [21]．また，

図 5 問題（3）「スプライトがずっと歩き続けるが，壁にあたったら反転する」の解答例

Fig. 5 Answer example for Q3: “The sprite walks until it hits a wall, then turns over”.

すべての問題が終了したあとに，総括的なインタビューを行った．これらをすべて録音し，テクストに書き起こした．

参考までに（3）の問題の解答例を**図** 5 に示す．

4.3 分析方法

書き起こしたインタビューのテクストからシステムの有効性を評価するには，質的評価を行うのが適切であると考えた [22]．なかでもボトムアップに概念モデルを構築できる木下 [23] の修正版グラウンデッドセオリーアプローチ（M-GTA）を今回は使用することにした．

M-GTA のプロセスでは，テクストから類似した具体例（ヴァリエーションと呼ばれる）を集めて概念を作り，概念の定義を加えて分析ワークシートを作る．ある程度の概念が集まったら，複数の概念をカテゴリに分けて，カテゴリの関係性を示す分析結果図にまとめる．最後に概念やカテゴリの名称を使いながら，分析結果図の概要を示すストーリーラインを文書化する．

4.4 結果と考察

協力者たちは，4 問すべてに正解を出すことができた．ただし，変数を扱った（4）の問題については，変数の概念もしくは Jointry の操作方法が難しかったためか，C が A，B をサポートしながらようやく正解にたどり着くことができた．

以上の結果から作成した概念とカテゴリを**表** 1 に，分析結果図を**図** 6 にそれぞれ示し，これらの考察としてのストーリーラインを以下に示す．《》は概念名，〈〉はカテゴリ名を示している．なお，分析段階に作成した分析ワークシートのヴァリエーションについては，付録に掲載する．

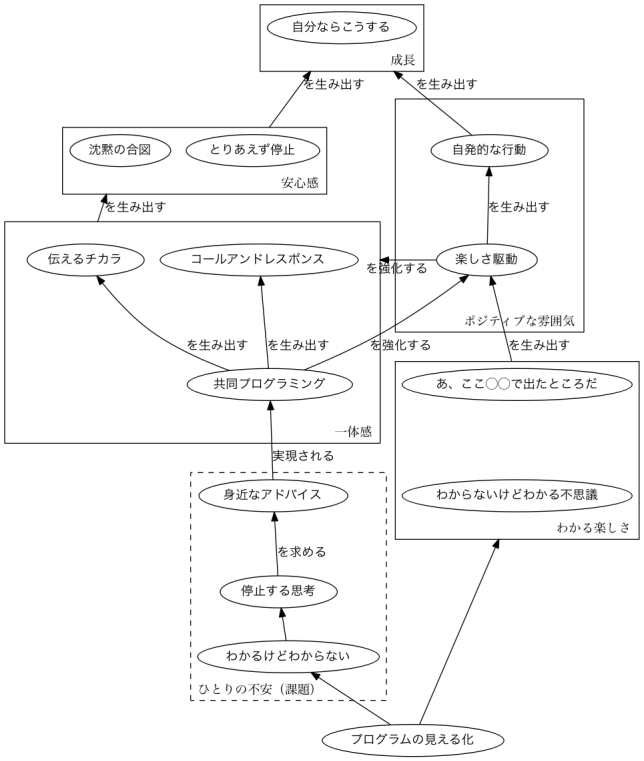

図 6 分析結果図

Fig. 6 Analysis result diagram.

ビジュアルプログラミングによる《プログラムの見える化》によって，プログラミング初心者はプログラムを視覚的にとらえられるようになる．その結果，類似したプログラムを《あ，ここ◯◯で出たところだ》と直感的に把握できるようになる．また，考えるよりもまず手を動かすことができるので，《わからないけどわかる不思議》といった感覚を持つ．これらはすべて〈わかる楽しさ〉の体感につながる．

操作が楽しければ，積極的に《自発的な行動》ができるようになる．このことを《楽しさ駆動》と呼ぶ．こうした一連の行為によって〈ポジティブな雰囲気〉が生み出される．

だが一方で，初心者は《わかるけどわからない》という感覚を持つ．これは《プログラムの見える化》によって，目の前に対象物が表現されているにもかかわらず，プログラム本来の難しさから理解が追いつかない状況を示している．その結果，操作の手が止まり，《停止する思考》の状態に陥る．こうなるとひとりでは解決できないので，《身近なアドバイス》を求めるようになる．このときにアドバイスが得られないと，〈ひとりの不安〉を抱くことになる．

ここで《共同プログラミング》の機能を使用すると，初心者同士に〈一体感〉が生まれる．ブロックを同時に操作することになるので，操作が衝突しないように自然と《コールアンドレスポンス》で声を掛け合うようになる．また，

表 1 生成されたカテゴリと概念とその定義
Table 1 Generated categories, concepts, and their definitions.

カテゴリ	概念	定義
—	プログラムの見える化	プログラムの構成がブロックで視覚的に表現されていること
ひとりの不安（課題）	停止する思考	ひとりで考えていては答えが出ないために，操作の手が止まってしまうこと
	身近なアドバイス	知りたいときにすぐにアドバイスをもらいたい様子
	わかるけどわからない	答えがまったくわからないわけではないが，自分では見つけられないこと
わかる楽しさ	あ，ここ○○で出たところだ	類似した問題は直感的に把握できること
	わからないけどわかる不思議	自分では理解していないが，操作していればなぜか答えがわかること
ポジティブな雰囲気	楽しさ駆動	楽しさがプログラム学習を前進させること
	自発的な行動	自分から手を動かしてみようと思って行動すること
一体感	共同プログラミング	複数人で一緒にプログラミングをすること
	コールアンドレスポンス	共同プログラミングをするときに声を掛け合うこと
	伝えるチカラ	共同プログラミングをするときに必要な意図の伝達力
安心感	沈黙の合図	思考停止ではなく「説明がよくわからない」という前向きな意思表示のこと
	とりあえず停止	停止やリセットができるので失敗が恐くない様子
成長	自分ならこうする	自分ひとりでプログラミング学習を進めること

問題を解くたびにプログラミングの意図を相手に《伝えるチカラ》が必要だと痛感していく．

《共同プログラミング》によって，〈一体感〉とは別に〈安心感〉も生まれる．ひとりのときの「沈黙」は，《停止する思考》のネガティブな印象だったが，複数人のときは「説明がよくわからない」ことを相手に伝えるポジティブな《沈黙の合図》となる．プログラミングの操作が《とりあえず停止》できることも失敗を恐れない〈安心感〉につながっている．

また，《共同プログラミング》によって，ひとりでやっていたときとは別の意味で《楽しさ駆動》を発動できる．それがさらに《楽しさ駆動》によって〈一体感〉を強化する．

以上の要素が，最終的にひとりでプログラミングを学習することにつながる．《共同プログラミング》を経験することで，《自発的な行動》と《自分ならこうする》という思いが強化され，さらに〈成長〉したいと考えるからである．

5. おわりに

本研究では，ビジュアルプログラミングツールに共同プログラミング機能を追加した Jointry を開発し，実際にプログラミング初心者を対象にして評価した．その結果，共同プログラミングによって一体感や安心感が生まれ，単独でのプログラミング学習にも効果があることがわかった．

今後はソースコードを MIT ライセンスでオープンソースとして公開し[*1]，第三者の協力を得ながら，機能の追加や改良をしていきたい．また，評価対象のバリエーションを増やし，プログラミング初心者の課題を解決する研究を継続したい．

謝辞 評価に協力していただいた AIIT の 2014 年度の 2 年生 3 人に感謝する．

[*1] http://github.com/jointry

参考文献

[1] Code.org: What Most Schools Don't Teach, https://www.youtube.com/watch?v=nKIu9yen5nc.
[2] WIRED: エストニア、小学 1 年生からプログラミングの授業実施へ, http://wired.jp/2012/09/07/estonia-reprograms-first-graders-as-web-coders/.
[3] TechCrunch: イギリスは義務教育（5-16 歳）公式カリキュラムにプログラミング教育を導入，http://jp.techcrunch.com/2014/02/05/20140204uk-government-backs-year-of-code-campaign-boosts-funds-to-teach-code-in-schools/.
[4] 産業競争力会議：成長戦略（素案），http://www.kantei.go.jp/jp/singi/keizaisaisei/skkkaigi/dai11/siryou1-1.pdf.
[5] 増井俊之：インタフェースの街角（10）ビジュアル・プログラミング，*UNIX MAGAZIN 1998.9*（1998）.
[6] Scratch Team Lifelong Kindergarten Group MIT Media Lab: Scratch - 想像、プログラム、共有，http://scratch.mit.edu/.
[7] 阿部和広：小学生からはじめるわくわくプログラミング，日経 BP 社（2013）.
[8] 森秀樹，杉澤学，張海，前迫孝憲：Scratch を用いた小学校プログラミング授業の実践: 小学生を対象としたプログラミング教育の再考，日本教育工学会論文誌，Vol. 34，No. 4，pp. 387--394（2011）.
[9] 森秀樹：Scratch を用いた文系大学生向けプログラミング教育，日本教育工学会論文誌，Vol. 34，pp. 141--144（2010）.
[10] Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B. and Kafai, Y.: Scratch: Programming for All, *Commun. ACM*, Vol. 52, No. 11, pp. 60--67 (online), DOI: 10.1145/1592761.1592779 (2009).
[11] Kelleher, C. and Pausch, R.: Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers, *ACM Comput. Surv.*, Vol. 37, No. 2, pp. 83–137 (online), DOI: 10.1145/1089733.1089734 (2005).

[12] 西田知博, 原田章, 中村亮太, 宮本友介, 松浦敏雄：初学者用プログラミング学習環境 PEN の実装と評価，情報処理学会論文誌，Vol. 48, No. 8, pp. 2736–2747 (2007).
[13] Fowler, M. and Parsons, R.: ドメイン特化言語パターンで学ぶ DSL のベストプラクティス 46 項目，ピアソン桐原 (2012).
[14] 松澤芳昭, 保井元, 杉浦学, 酒井三四郎：ビジュアル-Java 相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価，情報処理学会論文誌，Vol. 55, No. 1, pp. 57–71 (2014).
[15] 一般社団法人電子情報技術産業協会（JEITA）：アルゴロジック，http://home.jeita.or.jp/is/highschool/algo/.
[16] 鈴木栄幸, 加藤浩, 伊東祐司：実体を持つプログラム言語「アルゴブロック」：思考と共同作業の道具として，全国大会講演論文集， Vol. 46, No. 1, pp. 9–10（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110002881296/⟩ (1993).
[17] McWhorter, W. I. and O'Connor, B. C.: Do LEGO(R); Mindstorms(R); Motivate Students in CS1?, *SIGCSE Bull.*, Vol. 41, No. 1, pp. 438–442 (online), DOI: 10.1145/1539024.1509019 (2009).
[18] Wikkiams, L. and Kessler, R.: ペアプログラミング - エンジニアとしての指南書，ピアソンエデュケーション (2003).
[19] 千葉滋：2 週間でできる! スクリプト言語の作り方 (Software Design plus)，技術評論社 (2012).
[20] 秋口忠三：NSBroker: インターネット上での対話型通信基盤, 産業技術大学院大学紀要，Vol. 6, pp. 21–36（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009588115/⟩ (2012).
[21] 樽本徹也：ユーザビリティエンジニアリング (第 2 版) - ユーザエクスペリエンスのための調査、設計、評価手法 -，オーム社 (2014).
[22] 情報システムと社会環境研究会情報システム有効性評価手法研究分科会：情報システムの有効性評価質的評価のガイドライン (2013).
[23] 木下康仁：グラウンデッド・セオリー・アプローチの実践 - 質的研究への誘い，弘文堂 (2003).

付　　録

A.1 分析ワークシートのヴァリエーション

プログラムの見える化

- A「普通のプログラミングと同じようにできるから，考え方を学ぶにはすごくいいと思います」
- C「できることの少なさが，正解を生み出しやすいと思う．キーボードだと，セミコロンの打ち間違いとか，あとは英語のスペルミスとか，全角で打っちゃうとか，そういうのでコンパイルエラーが起きちゃうので，それで動かないのがよくわからなくて，またそこで悩んだりして．ビジュアルプログラミングだと，そういうプログラムそのもので悩むことが少ないと思う」
- A「（テキストだと）ハードル高いから，こっち（ビジュアルプログラミング）のほうがいいかもしれませんよね」

停止する思考

- B「ひとりでやってるときは考えているときに手が止まっちゃう．自分が思うようになかなかできない，手が出せない」
- C「ひとりでやっていると正しいかどうかがわからない」

身近なアドバイス

- B「もっとヒントがほしいと思った．ブロックがどうやって動くのかを先にヒントとしてくれれば，もっとわかりやすかったと思う」
- A「アドバイスを言ってもらったほうが助かる」
- C「誰かに横について見てほしいという感じがする」

わかるけどわからない

- A「動いているからわかりやすいけど，むずかしい」
- C「ペンギンが歩いているのはわかるんですが，それをどうやって再現すればいいのかがよくわからない．そこはちょっと自分で考えないといけない，というのはあるかもしれない.」
- A「いちおう動くんですけど，足踏みしてくれないこともあって……」

あ，ここ○○で出たところだ

- A「これは……さっきのを繰り返せばいいんですかね」
- B「これさっきと同じで，あと回転ですよね．壁まで行ったら戻るだけ」

わからないけどわかる不思議

- B「動いてるのはわかるけど，この条件の意味がわかんないですよね．1 == 1 ってどういう意味なんだろう」
- A「これでいいんですかね．条件がわかってなくても」
- C「条件がわかってなくても，動きを見たら正解だっていうのはわかるっていうのは，おもしろいですね」
- A「なぜわかったか，不思議です（笑）」
- A「わかんないけど，さわってれば，そのうちわかるかなあと思って」

楽しさ駆動

- A「やってみましょうよ．おもしろそうだから」
- C「わくわくするね」
- A「これ，楽しいですね．結構」
- B「うわ，止まっちゃったよ（笑）」
- C「これは，ダメですね（笑）」
- B「どんどんやっていくと面白いですね」

自発的な行動

- A「[実行] ボタンを押してもいいですか？」
- A「こっちでいじってもいいですか？」

- B「実行してみますよ」
- B「いったん（ブロックを）外しますね」
- A「一回，やってみますよ」
- A「じゃあ，触っていいですかね」
- B「じゃあ，やりましょうか」
- C「はい，実行しまーす」
- B「ここに条件を入れたらいいんじゃない？」

共同プログラミング

- C「みんなで同じものを共有できるって，いいですね」
- B「チームで話しながらできるのは，いいですよね．同じ画面見ながらできるし」
- A「ぼくは別に作りたいものがないんですよね．でもこれだと，みんなで一緒にやるから（いいと思う）」
- C「話し合いながらできるっていいですよね．みんなで同じものを扱っているっていう．この感覚が」
- B「課題を動く形で見せられたら，すごいなあって思うし．私も自分でもなんとか作ってみるかあって気持ちになる」
- B「誰かがやってるのをちょっと見てたら，それで気付くところがあったら，手を出そうかなと思う．手が出しやすい」
- B「もし同時に操作していると，名前が出ているとわかりやすい．誰が操作中か」

コールアンドレスポンス

- A「こっちでいじってもいいですか？」C「どうぞどうぞ」
- A「こっちでやっちゃっていいんですか？」C「どうぞ」
- A「いっかいやってみましょうよ．おもしろそうだから」B「お願いします」
- A「じゃあ，お願いします」C「はい，実行しまーす」
- A「プラスでいいですか」C「プラスでいいと思いますよ」
- C「条件のところに入ります？」A「あ，入りました」

伝えるチカラ

- C「ひとりでやるプログラミングとは使う能力がぜんぜん違う感じがしました．これって，みんな同じで，グループで考えを共有したいときに，自分はこうだから，こうやってブロックを置きますよって，話し合う必要もあるので．あの，チームで何かモノを作る訓練にもなるかなあって．ひとりで作ってるんじゃないぞ，と」
- A「就職試験のグループワークでよくやるやつみたい（笑）」
- C「自分が考えているロジックを相手に伝えて，納得してもらってプログラミングをしないと，チームがバラバラになってしまう．なので，交渉能力というか，別の何かを使う気がします．プログラミングっていうのは，チームで組むことがメインじゃないですか．なので，これはこれで，いい教材だと思いますね」
- A「チーム開発でコミュニケーション力がいるんで，できるひとが勝手にやったら，できない人は理解できないんで，自分がわかったところまでの内容をまわりに伝えられるか……．プログラム以外でも」

沈黙の合図

- C「これは歩数を制御しなきゃいけないと思うんですよ．なので，変数を作るっていうのをやりたいと思います」A「……」B「……」A「変数ってラベルみたいなやつですよね」
- C「たとえば，値が 20 より下だったら，この状態．20 以下だったら，ずっと動いてますよね」B「……」A「……」B「変数に 1 を足すんですか？」
- C「そうすると，歩数に 1 ずつたされるっていう」A「ん？」C「歩数に，歩数プラス 1」A「……」B「……」A「ああ，わかった」

とりあえず停止

- A「あれれ．おかしい．停止，停止を押そう」
- B「ちょっと中止してみますね」
- A「いったんリセットで（笑）」

自分ならこうする

- C「実行環境がひとつしかないと，取り合いになって，最後まで見たいのに，見れないっていうのがあるかも．試行錯誤用のパレットみたいなのがあると，またいいかもしれない」
- A「操作を引き取る（独占する）機能があってもいいかもしれないですね」
- C「ひとりのときは試行錯誤できるのに，オープンになると（共有されていると）やりにくい．でも，そういう部分も含めて，いい教育になりそう．みんなで試行錯誤したらいいと思います」