03

基般

子供の創造的活動とプログラミング学習

阿部和広（青山学院大学社会情報学部）

これからの子供に求められているのは，ATC21s[☆1]が提唱する21世紀型スキルだという．これには，批判的思考力，問題解決能力，コミュニケーション能力，コラボレーション能力，情報リテラシーなどが含まれる．ひたすら与えられた課題を解くことを求められてきた子供たちにそれが可能なのだろうか．また，プログラミング学習を通して，それが獲得できるのだろうか．

大学におけるプログラミング教育の例

筆者は，青山学院大学で「社会情報体験演習」という授業の一部を2012年度から担当している．これは社会情報学部1年次前期の必修科目で，90分×6回で行っている．この授業では，MITメディアラボで開発され，筆者が日本語版を担当したブロック型のビジュアルプログラミング言語，Scratch[☆2]と，センサやアクチュエータなどさまざまなデバイスを組み合わせて，自分たちのアイディアをグループごとにプロトタイプとして実現し，プレゼンテーションする（**図-1**）．

この内容は，MITメディアラボの初代所長であるNicholas Negroponte教授が唱えたスローガン，"Demo or Die"を実践するものだ（このスローガンは現所長の伊藤穰一氏により，"Deploy or Die"にアップデートされている）．

授業の導入として，ボールをパドルで跳ね返すゲーム（いわゆるPONG）を使った課題を課している．具体的には，センサボードに取り付けられたスライダ（横型の可変抵抗器）を操作して，画面のパドル（スプライト）を画面の端から端まで左右に動かすというものである．

このとき，スライダからは，0～100が変数の値として得られること，パドルが移動できる範囲はx座標上の-240～240であることは所与である．したがって，これは**図-2**のような単純な変換式で表すことができる（別解もある）．

しかし，多くの学生は**図-3**のような単純な代入式から先に進めない．これだと，パドルは画面の右側の半分弱の範囲でしか動かせない．

このとき，仕組みは分かっていても，それをScratchで表現する方法が分からないことも考えられる．そこで，個別に聞きとりを行ってみると，中学校で学んだはずの一次関数の傾きと切片の話と，いま与えられている課題の間に関連が見いだせていないようだった．これは複数の年度を通して共通の傾向である．

このことに限らず，既存の知識を応用することや，示された例題から逸脱すること，教えられていない

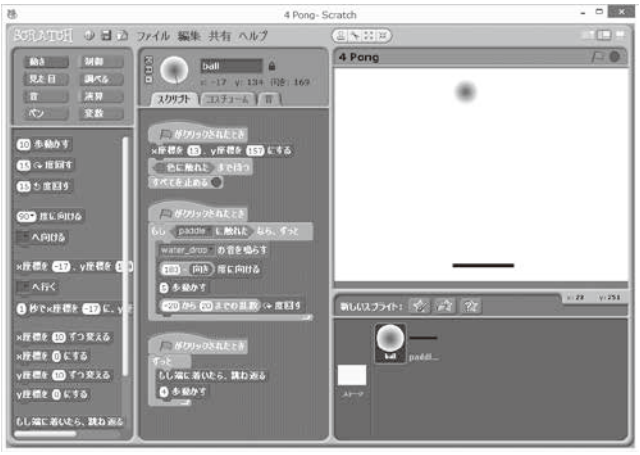

図-1　Scratchの画面（PONG）

☆1 http://www.atc21s.org/
☆2 https://scratch.mit.edu/

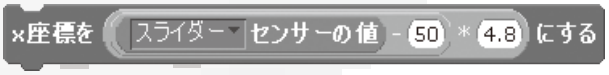

図 -2　パドルをスライダで動かす式（正解）

図 -3　パドルをスライダで動かす式（間違い）

機能やブロック（メソッド）を試してみる傾向は一概に低いように見える．一方で，自分で考えるのではなく，正解を求める声はいくつか聞かれた．

この授業を開講している 1 年次の前期は，入学試験からさほど間がない時期である．つまり，この状況は入学試験に合格する学力があったとしても，それを問題解決に活かせるかどうかは別の能力である可能性を示している．

この演習では，口を開けた学生に教員が知識を注ぎ込むのではなく，学生自身が手を動かして自ら問題を解決することを意図的に強いている．その際，インターネット検索を含む，資料や文献の活用は制限していない．マニュアルも教科書も与えられず，教員からの答えも期待できないと気付いた学生は，最終的には課題に真剣に取り組むようになる．

しかし，これを大学で行うのはいかにも遅く，初等中等教育段階で習慣付けできているべきではないだろうか．

小学校におけるプログラミング学習の例

筆者は主に小学生を対象に Scratch を使った民間のワークショップや，学校での授業と課外活動も全国で行っている [1]．回数は年間で数十件ほどで，そこでよく用いているのが，「ネコから逃げろ」（ネコ逃げ）という入門用の例題である☆3．

ネコ逃げは，ネコとネズミのキャラクタ（スプライト）が追いかけっこをするゲームを題材にした 30 ～ 120 分（通常は 45 分× 2 の 90 分）の内容で，Scratch の基本操作から，逐次・繰り返し・分岐，オブジェクト指向プログラミング，並列処理までを扱う．90 分の場合は，自由製作時間に 30 分くらいを当てるので，実質は 60 分くらいの内容になる．

こう書くと詰め込みすぎではないかと感じられるかもしれない．しかし，小学 3 年生くらいであれば，ほぼ全員がこれらの意味を理解し，自分のものとして使えるようになる．ただし，用語は一切説明しない．また，単元ごとに時間を区切ることもしない．子供たちにとっての目的はゲームを作る（遊ぶ）ことであり，前述の概念はそのために必要なものとして，その都度現れてくる．

スクリプト（プログラム）は，漢字仮名交じり文か，平仮名のみに切り替えることができ，完全ではないが，日本語として読み下せるように作成してあり，単に書くだけでなく，ところどころで音読を行っている．

このワークショップを進める過程で，共通して見られることがある．それは，ネコを歩かせる速さを過大な数（99999…9 など）にすること，ネコのコピーを大量に作ること（子供によっては 100 匹近く），ネコを極端に大きくする，あるいは小さくするなどである．これらは，ファシリテータ（従来の教員に相当する人．助力者）がそう指示したからではなく，いずれも自発的に試している．そして，その結果を面白がり，周りの友だちにも教えることで教室中に広まっていく．これは地域や学年を問わない．

また，説明していないスプライトやブロック（命令）を勝手に使ったり，自由製作に入る前に独自の改造を試みることもよくある．設定した休憩時間に休まないのも共通している．

自由製作では，シューティング風だったり，RPG（Role-Playing Game）風だったり，ビジュアルノベル風だったり，もはや分類不能だったりと，原型をとどめないほどにオリジナリティが発揮された作品が作られる．これは創造性の発現と言って差し支えないだろう．

この活動を通して感じられるのは，子供たちが作ることと遊ぶこと，さらには学ぶことの間に違いがないように見えることだ．その中で自発的に課題に

☆3　http://swikis.ddo.jp/abee/77

取り組む姿勢や試行錯誤する態度も生まれている．やりたいことを達成するためなら，代数や三角関数を学ぶこともいとわない（これは毎回聞かれる）．この子たちが，ほかの授業ではそうではないことは，普段子供たちを見ている教員のコメントから得られている．典型的なものは，「こんなに真剣にやっているところを初めて見た」，「間違いを恐れずに試している」などである．

創造性はどこから来るのか

このような子供たちの積極性と創造性は何によってもたらされるのだろうか．

まず考えられるのは，新奇なものによる効果である．つまり，物珍しいおもちゃを与えられた子がそれで遊んだだけではないかということだ．よほどひどいものでない限り，どのような教材であっても，最初の数時間は子供たちの興味を惹くだろう．

では，2 年前に Raspberry Pi[☆4]（きわめて安価な超小型パソコン）が全児童に配られ，Scratch を教科の授業で使うことが日常化している品川区立京陽小学校の場合はどうだろうか[2]．教室に集まった子は，慣れた手つきで自分の Raspberry Pi をディスプレイやキーボードに接続し，OS を起動する．Scratch が立ち上がっても特別はしゃぐこともない．これは，書道セットや鍵盤ハーモニカと同じ態度だ．しかし，授業が始まり，先生からテーマが示されると，初めての子と同じような集中力を発揮する．このことから，プログラミングを使った学習の効果は一過性ではないと考えられる．

次に，対話性も可能性の 1 つに挙げられるだろう．Scratch は対話的なプログラミング環境である．つまり，何かの操作を行えば，すぐに反応が返るようにできている．このことは，ボタンをクリックしたときのフィードバックから，スクリプトの実行時にブロックを組み替えると即座に挙動が変わることに至るまで一貫している．すなわち，思い付きをすぐに試して，その結果を得られるということだ．この特性と子供たちの好奇心が結び付くことが，先に挙げたような大きな数を試してみることにつながっている．そこで面白い結果が得られることが，さらに試してみることの動機付けとなる．

さらには，自由度の高さもある．自由製作のくだりで紹介したように，子供たちが作りたいものはさまざまである．そして，コンピュータは，その基本的な特性として，どんなものでもモデル化してシミュレートできるという性質を持つ．

文字だけでなく，グラフィクスや音といったマルチメディアをサポートし，オブジェクト指向と並列処理をパラダイムとする Scratch は，子供たちが関心を持ちやすいゲームやアニメーションに適合している．このことと，ブロック言語によるコーディングの敷居の低さが相まって，短時間で意味のある作品を作ることを可能にしている．これは，厳密な分析・設計を行わずに修正の繰り返し（イテレーション）で開発するラピッドプロトタイピングやアジャイル開発に近い．

でき上がった作品は，Web サイトにアップすることで CC BY-SA（クリエイティブ・コモンズ 表示―継承）をライセンスとする共有の資産となり，それをベースにしたリミックスもできる．これはトレースが可能で，GitHub[☆5]（ソースコード共有サービス）とよく似ている．

また，正しさが客観的に分かるのもポイントである．ある振舞い，たとえば，ネコがネズミにぶつかったらニャーと鳴かせたいとき，ぶつかって鳴かなければそれは間違っており，ぶつからなくても鳴くとすればそれも間違っている．正解にたどり着く道筋は複数あるとしても，正誤の判定ができることにより，仮説とその検証を進めることができる．

そして，複数の正解の中にも最適な解があり，それはさまざまな条件によって変わってくる．計算量を重視するか，メモリサイズを取るか，リアルタイム性か，保守性を考慮するかなど，定量的に測れる指標と定性的な見方も加味しながら，異なる制約を考慮して決めることになる．

☆4 https://www.raspberrypi.org/

☆5 https://github.com/

ここで挙げたようなさまざまな特徴は，その一部だけであれば，音楽や図工でも実現可能だろう．しかし，全体を総合的に扱うことができることと，物理的な制約がなく現実に存在しないものも対象にできることは，プログラミング学習の優位性である．

小学生と大学生の違い

このような小学生と，さきほどの大学生の違いはどこにあるのだろうか．大学生もかつては小学生だったはずである．大学生は，せっかく自由が目の前にあるのに，それを使うことを恐れているようにも見える．彼ら彼女らは，試験問題のような形式化された課題を解くことについては，高い能力を持っている．しかし，「自分のアイディアをかたちにせよ」のような，曖昧模糊とした課題を前にすると，「何も思い付かない」，「（単位を取るには）どうすればよいか教えてください」ということを平気で言う．

この根底にあるのは，受験勉強を始めとした目的を達成するための最適化を至上とする考え方があるのではないか．それに対して，小学生は他者から与えられた目的意識が希薄で，純粋に自己の楽しさを追求するが故に，創造性を発揮しているのではないか．両者を観察していると，一般に小学生のパフォーマンスは期待より高く，大学生は期待より低い．

これは，大学生に小学生と同じことをやらせるべきという主張ではない．大学生と小学生では，興味も関心も異なっている．それぞれにマッチした各人が作ることに意味を見いだせる課題を考える必要がある．

パズル型チュートリアルの問題

米国の NPO である Code.org が中心になって進めている "Hour of Code"（HoC．1 時間だけでもプログラムを書いてみよう）運動は，数年前から日本でも行われるようになった．特に 12 月の 1 週間を "Computer Science Education Week"（計算機科学教育週間）と定め，企業や NPO が協力して各地でワークショップなどのイベントを開催している．

図 -4　Hour of Code の例

HoC にはいろいろな参加スタイルがあり，Scratch を開発した MIT メディアラボも独自のカリキュラムを作成している．その中でもよく使われているのが，Code.org が無償で提供しているパズル型の課題を解くチュートリアルである（このパズル自体を Hour of Code と呼ぶこともある）（**図 -4**）．

パズルには有名な映画やゲームのキャラクタを使ったものなどが複数用意されており，それぞれが 15 ステージ程度の問題で構成されている．子供たちは，升目に区切られたステージ上のキャラクタを，与えられた指示に従って動かすプログラムを書くことを求められる．ステージごとに逐次・繰り返し・分岐などのテーマがあり，進むにつれて難易度が上がるようになっている．少ないステップ数で作る方が評価が高くなるようになっており，最終ステージを終えると，修了証がもらえる（得点は書かれていない）．

これは Google Blockly という JavaScript で書かれたブロックプログラミング言語フレームワークで作成されており，パソコンだけでなく，タブレット上の Web ブラウザでも動作する．学校で使うことも考慮され，ユーザ管理や進捗管理もできるようになっている．また，ブロックを JavaScript のソースコードに変換したものも参照できるようになっている．

このパズルは子供たちの興味や関心を惹く上で大変良くできている．筆者が行うワークショップで普段は Scratch を使っている子供たちを対象に行ったところ，子供たちは夢中になって取り組んでいた．

しかし，このパズルは，子供たちの中から出てきたものではない．ほかの誰かから与えられたもので

ある．確かに，最終ステージまで進めば，ある程度の自由製作を行うことができる．しかし，使えるブロックや素材など，Scratch と比較してもできることは限られている．また，パズル自体を自分たちで作り出せるようなメタな仕組みは提供されていない．そのため，解き終わった子供たちは短い時間で飽きてしまった（一部の子はチート（システムの裏をかく技）の方法を発見して，それで遊んでいた）．一方，ファシリテータとしては，子供たちからの，あれをやりたい，これをやりたいという例外的な要求がなく，システムに進行を任せられるので，その点は大変楽だった．

このとき，子供たちが行っている内容をつぶさに観察したり，子供たちへの聞きとりを行わない限り，はた目には Scratch の活動との差異を見つけるのは難しいだろう．教員や保護者の目から見ても，楽しく取り組んでいるように見える．

問われるべきは，このパズルを通して子供たちが何を学んだのかということである．与えられた課題を解くことがゴールであるとするならば，先に挙げたようなプログラミング学習の意義を損なっている可能性がある．これで逐次・繰り返し・分岐を理解したとしても，大学生が線形変換をプログラムの中で使うことを思い付かなかったように，パズルの中の出来事として，一般化して用いることができないのではないか．さらには，子供たちが持っているポテンシャルをチュートリアルのレベルに制限してしまうことになっていないだろうか．

これらは，HoC に限らず，プログラミング学習のカリキュラムを作成する上で留意すべきことである．これは，単に Scratch にすればよいということではない．極端なことを言えば，Scratch を使って 4 択の穴埋め問題を解かせることもできる．

筆者が開発した「にんげんプログラミング」☆6 というカリキュラムでは，子供たちにパズルを提示して解かせるだけでなく，そのパズル自体も同じ仕組みで作られていることを示し，子供たちがパズルを作ったり，それ以外のものも作れるようにしている．過去の例では，子供たちは与えられた問題を解くことよりも，互いに問題を出し合うことをより楽しんでいた．

子供たちの管理が楽で，教員の負担が少なく，子供たちが楽しんでいるからと言って，それが良いカリキュラムとは限らない．

プログラミング学習の必要性

世の中は常に変化する．私たちを取り巻く環境も社会構造も変わっていく．その際に，教科書やマニュアルが与えられなければ，あるいは，人から教えられなければ何もできないのでは困る．

仮にそれらがあったとしても，書かれていることを鵜呑みにするのではなく，問題を適切にモデル化し，シミュレーションを行い，バグとデバッグを繰り返して検証できる人材が求められる．これが，21 世紀型スキルに加えて，Computational Thinking（プログラマ的思考法）が注目されている理由の 1 つだろう．

世界最先端 IT 国家創造宣言☆7 で謳われている高度 IT 人材の育成は重要な課題である．しかし，目的への過度な最適化は，逆にその達成を遠ざける．第 3 代アメリカ合衆国大統領 Thomas Jefferson が言うような，自ら学ぶことのできる啓発された国民の育成の方が，よりメタであり，より重要である[3]．もし，プログラミングを通して見られる学習態度の変容がほかにも転移するとすれば，それがプログラミング学習の必要性を主張する裏付けとなる．この検証が今後の課題である．

参考文献

1）阿部和広：幸せなパソコン教室のために，情報処理，Vol.55, No.6, pp.598-601 (June 2014).
2）阿部和広：子どもの創造的活動と ICT 活用，情報処理，Vol.56, No.4, pp.350-354 (Apr. 2015).
3）Kay, A.：Dynabook とはなにか？，小学生からはじめるわくわくプログラミング，日経 BP，pp.142-149 (2013).

(2015 年 12 月 31 日受付)

☆6 http://scratch-ja.org/human

☆7 http://www.itdashboard.go.jp/Achievement/

阿部和広 ■ abee@squeakland.jp

青山学院大学社会情報学部客員教授．Etoys と Scratch の日本語版を担当．2003 年度 IPA 認定スーパークリエータ．専門は構築主義と初等教育におけるプログラミング教育．