東北女子大学・東北女子短期大学　紀要　No.56：57～66　2017

プログラミング導入教育の為の Scratch の利用

友　田　志　郎*

Scratch for Introduction Programming Education

Shirou TOMODA*

Key words : Scratch
プログラミング導入教育　Introduction Programming Education

はじめに

2020 年度より完全実施される新学習指導要領では、小学校段階でのプログラミング教育が必修化される。ただし、プログラミング教育の為の教科が新設されるわけではなく、算数や理科、生活などの既存教科の中でプログラミング教育を実施していくと言う事なので、具体的な実施体制や内容については今後さまざまに検討されていくものと考えられる。

ここで言う“プログラミング教育”については文部科学省の有識者会議による論点まとめ[1]として、「プログラミング教育とは、子供たちに、コンピュータに意図した処理を行うよう指示することができるということを体験させながら、将来どのような職業に就くとしても、時代を超えて普遍的に求められる力としての『プログラミング的思考』などを育むことであり、コーディングを覚えることが目的ではない。」と述べられている。つまり、特定のプログラミング言語やプログラミング環境に依存せず、普遍的な問題解決能力としての『プログラミング的思考』を育むことを目的としている。(「コーディング」とはすでに設計のできあがっているプログラムを特定のプログラミング言語を用いて記述する作業のこと)

プログラミングとは、対象を分析してどのような情報・データが必要であるかを適切に判断してモデル化し、それをどのように処理すれば目的が達成できるかという手順を考え出す作業である。これは問題解決能力を育てる為の訓練としては極めて有効なものであり、小学校段階からそうした訓練を行うことは意義深いと言えるであろう。

プログラミング導入教育に適したシステム

では、実際に小学生を対象にプログラミングを教える場合にはどのようなプログラミング環境、システムが望ましいであろうか？

一般に、プログラムはテキスト文字を用いたプログラミング言語によって記述される。これらの中には、入門向け、あるいは教育用とされる言語も様々に存在する。しかしその場合も、プログラムを作成する為には固有の予約語や文法、扱えるデータ型など、最低限のコーディングを覚えておく必要がある。一方、コーディングは実際にその言語でプログラムを作成してみないとなかなか覚えにくい事も確かである。このような、プログラミング言語を学ぶためには実際にプログラムを作成してみる事が早道であるが、その為にはそのプログラミング言語を一通り覚えておく必要があるという点は、ひとつのジレンマであり、初学者にとってはかなり高いハードルとなる。仮に小学生にプログラミングを教えるのであれば、事前に何の予備知識も持たないでも、その場の操作で何らかの「モノ」が作成できるようなプログラミング環境が望ましいと言えるであろう。

＊東北女子大学

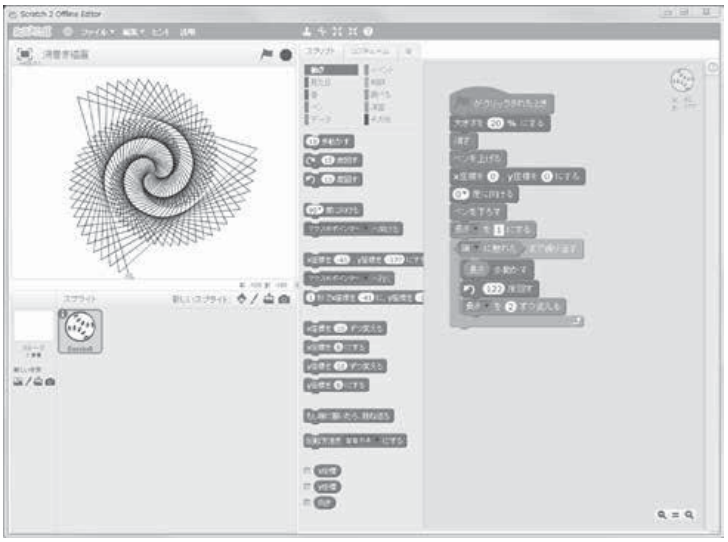

図 1：Scratch の作業画面

また、プログラムの動作の様子や処理結果が視覚的で判りやすい事も重要である。何の知識も無い段階で、目に見えないものをイメージしながら作業していくことは簡単ではないからである。しかしながら、多くのプログラミング言語ではこれらを視覚化しようとすると何らかのサブルーチンライブラリが必要になり、それを使用するためには「呪文」のような前処理や下準備が必要となる。「呪文」を唱えることにエネルギーを使い果たしてしまっては意味がないので、そうしたものを必要とせずに視覚化されているプログラミング環境であることが望ましいといえる。

Scratch

こうした条件を満たすシステムとして、Scratch が挙げられる[2]。Scratch はマサチューセッツ工科大学（MIT）のメディアラボで開発されたプログラミング教育システムで、同様のシステムのなかでも最有力のものとして位置づけられている。小学校でのプログラミング教育導入に於いても、まず Scratch が検討されることが多いのではないだろうか。

筆者が担当する東北女子大学児童学科の「ICT の基礎２」の授業では、Scratch を用いたプログラミングを演習題材の一つとして行っている。本稿では、Scratch の特徴、プログラムの作成や授業で用いる上での留意点、及び、演習題材その他のプログラミング例について簡単に述べる。

Scratch の概要

Scratch は 2006 年、MIT メディアラボで最初のバージョンが開発された。現在のバージョンは 2.0 で、Web ベースアプリケーションとして Web ブラウザ上でも動作させる事ができる。作成したプログラムは仮想機械上で実行され、開発環境と実行環境は一体化している。また、各国の言語に対応しており、日本語化された開発環境を使用することもできる。本稿では日本語環境の Scratch2.0 を用いて解説・議論する。

Scratch はオブジェクト指向のプログラミング環境であり。画面上に存在するオブジェクトに様々な属性と、そのオブジェクトを処理するためのプログラムである“スクリプト”を設定して動作させる。オブジェクトのうち、ステージ画面上で動かすオブジェクトは“スプライト”と呼ばれる。また、ステージ背景もオブジェクトとして扱われる。

スプライトにはスクリプトの他、スプライトの見た目を表現する“コスチューム”、スプライトが鳴らす“音”といった属性を持たせる事ができる。

スプライトに設定されたスクリプトは何らかの“イベント”によって起動される。イベントには、

ステージ右上の旗クリックイベント(多くの場合、スタートボタンとして使われる)、キー押下、スプライトのクリック、背景の切り替え、音量・タイマー・ビデオ、メッセージ受信、クローン作成がある。イベントが発生すると、各スプライトに設定されたイベントハンドラとなるスクリプトが起動される。発生したイベントに対するハンドラが設定されていなければ、そのスプライトは該当イベントを無視する。

実際に、Scratch を用いてプログラムを作成している画面を図 1 に示す。画面左上には動作結果が表示されるステージ、左下にはスプライトの一覧、右側には選択したスプライトに設定したスクリプト、コスチューム、音の編集のためのエリアが配置される。

Scratch によるプログラミングは、一覧から選択した“ブロック”をスクリプトエリアにドラッグ＆ドロップしてスクリプトを組み立てることによって行う。文法・構文的に正しいかどうかは、ブロックの形状、及び組み立ての際にうまく適合するか否かで判断できる。従って、文法・構文をあらかじめ覚えておかないでも、その場で判断しながら組み立てることができる。ブロック内には数値・文字列などの定数値や演算式、条件式を格納する場所が用意されているものもあり、それらには定数値を直接記述したり、演算ブロックや条件式ブロックをはめ込むことができる。

スクリプトを構成するブロックは、スプライトの“動き”、“見た目”、“音”を操作するブロックの他、動きの軌跡を使って描画する為の“ペン”、変数やリストを扱う“データ”、イベントによってスクリプトを起動させる“イベント”、アルゴリズムの記述とクローン作成のための“制御”、スプライトやマウスなどの状態を調べる為の“調べる”、四則演算や条件演算、論理演算等の“演算”、定義ブロックや拡張機能のための“その他”の 10 種類に分類される。これらのブロックは、スクリプトエリアの左端で、分類ごとに一覧できるようになっている。

図 2-1、図 2-2 に繰り返しループや条件分岐などアルゴリズムを記述する為の制御ブロックを例として示す。

図 2-1：制御ブロックの例（繰り返し処理）

図 2-2：制御ブロックの例（条件分岐）

データブロックではデータの格納場所である“変数”と“リスト”を作る事ができる。どちらも、全てのスプライトから利用できる共有メモリとして作成するか、該当スプライトからのみ利用可能にするかを選ぶ事ができる。C 言語などにある、実行時に動的にメモリ領域が確保される自動変数に該当するものは無い。

変数には数値や文字列などの値を格納する事ができる。リストはデータ構造の一種で、それぞれの要素が次の要素を指し示すポインタを持ち、ポインタをたどる事で各要素にアクセスできるというものであり、データの挿入や削除が容易に行えるという特徴を持つ。ただし Scratch の場合、リスト内の要素にアクセスするためには「○○番目」という添数を指定する以外に手段がないため、実際の使用感としては配列に近い。また、リストの要素にリストを含めてアクセスすることはできないので、ツリー構造や二次元配列的な構造は使用できない。

並列処理

現在では、コンピュータ上で複数のプログラムを同時に実行させて作業を行うことが一般的であ

る。また、一つのプログラムの内で処理単位を複数同時に実行させる並列処理も珍しくない。Scratch では各スプライトの各スクリプトは、起動されれば全て並列に動作する。このように、並列処理を行うプログラムを簡単に作成出来る事は、Scratch の大きな特徴だと言える。

一般に、並列処理の単位としてはプロセスとスレッドを挙げる事ができる。どちらも「実行中のプログラム」であるが、プロセスは独立したメモリ空間をオペレーティングシステムから割り当てられているのに対して、スレッドはプロセス内で単一のメモリ空間を共有している複数の並列処理単位であるという点が異なる。また、並列実行されている複数のプロセスが相互にデータをやりとりする場合には、オペレーティングシステムが実装しているプロセス間通信機能（パイプ、シグナル、メッセージ、共有メモリ、セマフォ、ソケット等）を利用する必要がある。

Scratch の場合、各スプライトがプロセスに該当し、スプライトの持つスクリプトがスレッドに該当すると考えれば理解し易い。各スプライト及びそのクローンの間の通信手段としてはメッセージと共有メモリが使用でき、スプライト内のスクリプトは単一のメモリ空間を共有している事になる。

クローンは動作中のスプライトの、座標位置やスケール、変数値なども含めた複製のことである。クローンを作ると、オリジナルのスプライトとクローンの２つが同時に同じプログラムを実行している事になる。両者を区別して処理内容を変えるためには、“クローンされたとき”のイベントハンドラを使用する。

授業で使用した例題プログラム

実際に、児童学科の「ICT の基礎２」の授業内で、演習題材として用いたプログラムをいくつか示す。

①簡単なアクションゲーム

Scratch ではスプライトの座標を変える、或いは経過時間を指定して動かす為のブロックの他、スプライトが端や他のスプライトに触れたかどうかの判定ブロック、秒数を指定して待つブロックなどが用意されており、また各スクリプトが並列

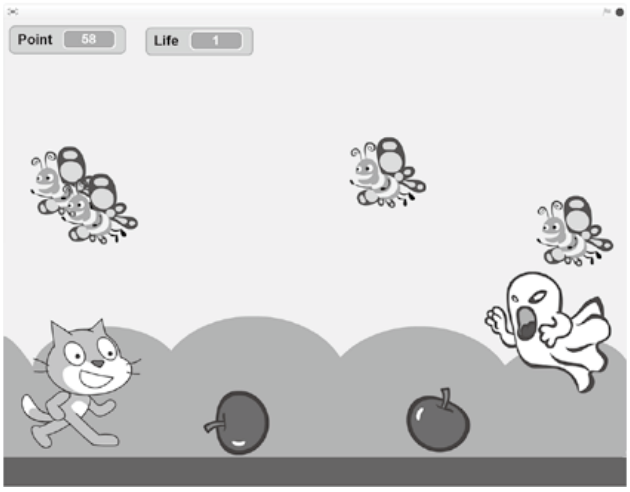

図 3-1：簡単なアクションゲーム画面

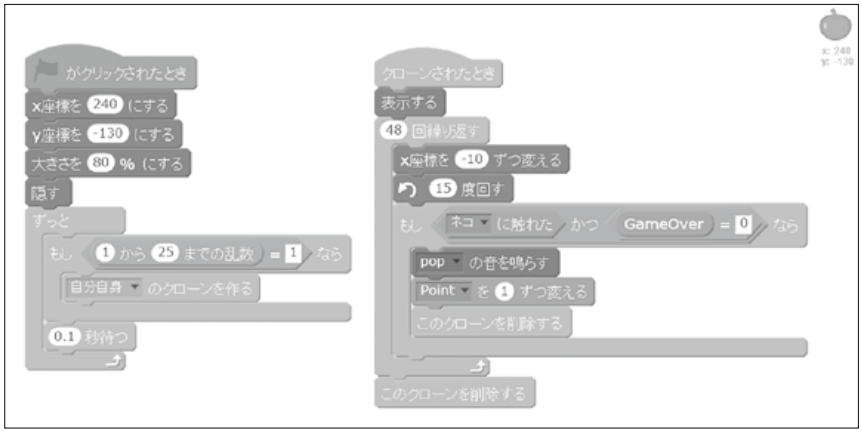

図 3-2：「りんご」のスクリプト

に動作することなどから、複数のスプライトが画面上で同時に動くようなプログラムを容易に作ることができる。このようなプログラムとして、児童学科の「ICT の基礎２」の授業では、簡単なアクションゲームを演習題材として取り上げている。

図 3-1 に示すゲームでは、画面左に「ネコ」がいて、画面の右からは「リンゴ」「蝶」「お化」けが次々とやって来る。「ネコ」が「リンゴ」や「蝶」に触れると Point が加算され、「お化け」に触れると Life が減少する。プレイヤーはスペースキーを押して「ネコ」をジャンプさせ、「お化け」を避けながら Point を獲得する。Life がゼロになればゲームは終了することになる。

データブロックとしては、変数として Point、Life の他、ゲームオーバー処理の為のフラグとして変数 GameOver を作成しておく。

図 3-2 には、「リンゴ」に設定したスクリプトを示す。旗クリックによってゲームがスタートすると、「リンゴ」のスプライトは、ステージ右端に移動して自分自身を非表示にする。あとは、0.1 秒ごとの繰り返しループ内で、乱数を用いて 1/25 の頻度で自身のクローンを作る。ゲーム画面上で実際に表示されている「リンゴ」は全てクローンである。クローンは生成されると、自分を表示状態にして、回転しながらステージを左に移動していく。その繰り返しループの中で、「ネコ」に触れたかどうかを判定し、触れたなら Point を加算して自身を消滅させる。ただし、GameOver 処理中であればそれは行わない。

「蝶」「お化け」の場合は、空中をふらふらと飛んで来るようにするので、乱数を使って Y 座標も変化させながらクローンを左に動かす。また、「お化け」の場合は「ネコ」に触れると、Life を減少させて“ダメージ”メッセージを送信する。

「ネコ」に設定したスクリプトは図 3-3 に示す。「ネコ」は旗クリックによってゲームが開始されると、Point、Life、GameOver の各変数を初期化、ステージ左の初期位置に自分を配置すると、あとは 0.2 秒ごとの繰り返しループ内で“次のコスチュームにする”。この「ネコ」のスプライトはコスチュームを繰り返して切り替えることで、足を動かして歩いているようなアニメーションを表現する事ができる。足の動きと平行して、スペースキー押下によって「ネコ」をジャンプさせることになるが、ここで、スペースキー押下によって“ジャンプ”メッセージを送信させ、“ジャンプ”メッセージのハンドラ内の処理によって「ネコ」がジャンプするという回りくどい方法をとっているが、この理由については後述する。他に、「お化け」から送信された“ダメージ”メッセージに対して、Life の値をチェックして GameOver の処理を行うスクリプトが設定されている。

②線画グラフィック

Scratch では、ペンを用いて線画を描くプログラムを容易に作成できる。ペンを下げた状態でス

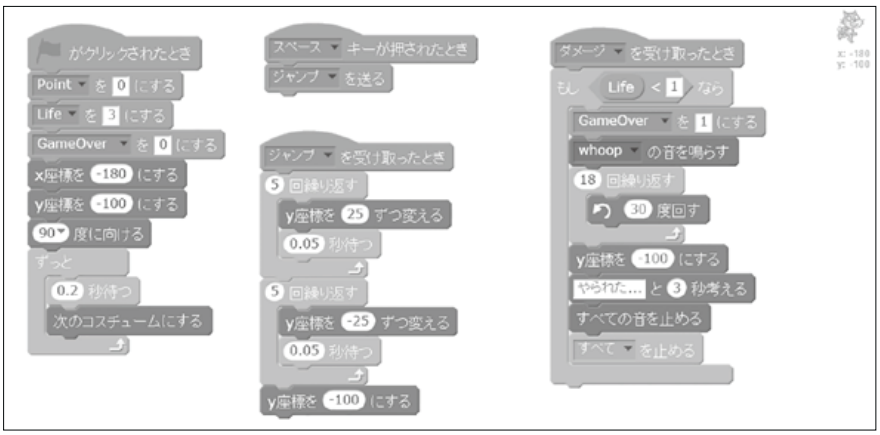

図 3-3：「ネコ」のスクリプト

テージ上でスプライトを移動させれば、移動先まで直線が引かれる事になる。したがって、任意の位置に線分を描くのであれば、ペンを上げてから始点まで移動し、その後ペンを下げてから終点まで移動すれば良い。

ステージ画面のサイズは横 480 ピクセル、縦 360 ピクセルで、中央を原点とする X-Y 座標系によって画面上の位置を指定できる。この座標値によってスプライトの位置を指定しながらペンを上げ下げして描画するというのが一つの方法である。

また、タートルグラフィックという、座標を指定せずに線画を描く方法もある。タートルグラフィックでは、スプライトの現在地を始点として、方向と距離で終点の位置を決めて描画する。名前の通り、カメが線を引きながら歩き回る様子をイメージすれば分かり易いかもしれない。例えば小学生に教える場合、座標という概念はまだ学んでいないので、タートルグラフィックによる描画プログラムのほうが適していると言える。Scratch では、スプライトの向き設定と距離を指定しての移動のためのブロックが用意されているので、タートルグラフィックを利用した描画プログラムの作成は極めて容易である。

線画グラフィックは、プログラミングを初めて学ぶ場合の題材として大変に適している。処理結果が分かりやすく、必要とする命令もペンの上げ下げと移動だけなので、飲み込みやすい。その一方で、同じ図形を描画する場合にも複数の方法が存在するので、一つの方法で描画したのちに、また別の方法での描画させるなど、幅を持たせた演習を行いやすい。

例えば、図 4-1 に示すような単純な正方形の格子を描画する場合も、座標を指定しての描画、及びタートルグラフィックによる描画の両方で、複数の描画方法を演習例題として設定することができる。縦線を全て描いてから横線を描く、或いは縦線と横線を交互に描く、大きさを変えながら正方形を描くなど、同じ処理結果を様々な方法で実現することで、プログラミングに必要な発想力、水平思考能力を養う訓練となる。

また、Scratch の場合にはスプライトのクローンを作成して並列描画させることも容易である。図 4-2 は実際に複数のクローンによって図 4-1 の格子図形を並列描画させている様子を示している。但しこの例では描画過程が分かり易いように、スプライトをゆっくりと移動させながら線を引く定義ブロックを作成して使用している。

Scratch を用いて線画グラフィックプログラムを作成する場合、描画範囲には注意する必要がある。描画範囲が 480 × 360 ピクセルと狭い上、スプライトを描画範囲外に移動させることができないので、座標値が範囲を超えてしまった場合は意

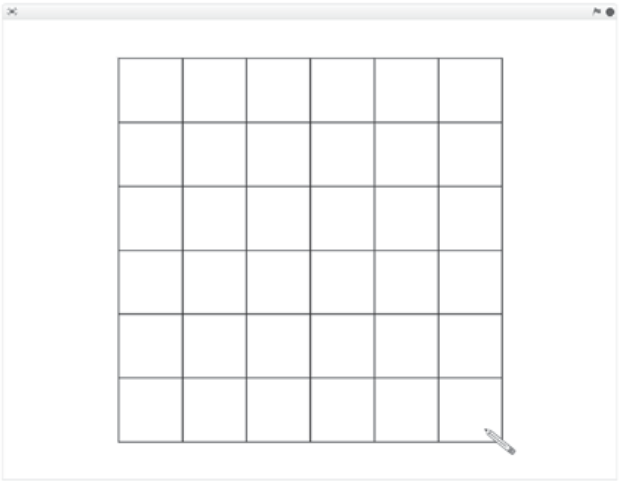

図 4-1：線画グラフィックによる格子の描画例

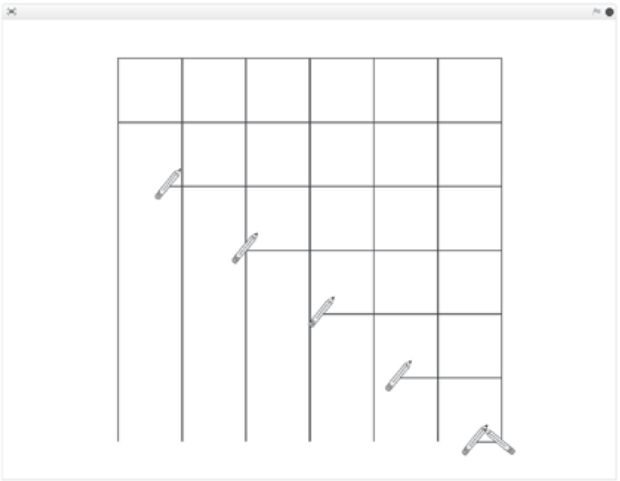

図 4-2：並列動作による格子描画

図した描画結果とならず、乱れた図形になってしまう。

Scratch プログラミングの注意点

Scratch は大変優れたプログラミング環境であるが、実際にプログラムを作成する上では幾つか注意しておく点がある。授業用の例題プログラムを作成した際に気づいた幾つかの点について以下で述べる。

①同一イベントの多重発生

Scratch プログラムはイベント駆動型であり、何らかのイベント発生に対するハンドラという形でスクリプトを作成していく。あるイベントに対してハンドラが処理を実行している間に別のイベントが発生すると、それぞれのイベントハンドラは並列実行される事になるが、この事には問題はない。問題になるのは、あるイベントに対するハンドラが処理を完結する前に同じイベントが発生した場合で、この場合、最初のハンドラの処理を未完結のまま放り出して、同じハンドラを頭から再び実行することになり、その新たなイベント処理が終了しても元のハンドラに制御が戻ることがない。どのタイミングでイベントが発生してハンドラが処理を止めてしまうかは予測できないので、これはやっかいな問題となる。その為、通常はハンドラの最初で“割り込み禁止”の処理を行うが、Scratch にはそうした機能は存在しない。

簡単な対応手段としては、ロック用の変数を用意し、ハンドラ処理中であれば１、そうでなければ０といった値の意味づけをし、ハンドラ処理の冒頭でロック変数の値を調べ、値が０であれば値を１にして処理を実行し、そうでなければ何もしないといった方法をとることができる。ただし、この方法にも後述する排他制御の問題が関与する点には留意する必要がある。

なお、同一イベントの多重発生の問題は、イベントの種類によって様相が異なる点にも注意が必要である。マウスクリックやメッセージなどではイベントの多重発生が起きるが、キー押下イベントでは起きない。前述の図 3-3 で示したゲームの「ネコ」のスクリプトでは、スペースキーを連打する事で「ネコ」が更に高くジャンプするようにする為、キー押下のイベントハンドラでメッセージを送信し、メッセージイベントのハンドラ内でジャンプ動作をさせている。つまり、ゲームとしての操作性の為に、敢えてイベントを多重発生させていることになる。その為、そのままでは「ネコ」がちゃんと着地しない事になるので、最後に強引に Y 座標を地面の位置に戻している。

②排他制御

並列処理プログラムでプロセス間で互いに情報やデータの遣り取りを行う為の手段の一つとして、共有メモリ等の共有資源を利用する方法がある。データを渡す側が共有メモリに書き込み、受け取る側がそれを参照するという方法で、データの高速な受け渡しが可能となる。ただし、共有資源を書き換えるプロセスが複数存在する場合、その内容に容易に矛盾が生じる。そうした場合、共有資源へのアクセスが排他的に行われるようにする必要がある。

その為の簡便な方法として、イベント多重発生への対応として解説したロック用の変数を用いる手段がある。ロック用変数の値を調べることで、他者がアクセス中であるかどうかを判別するものだが、実の所、この方法では確実ではない。ロック用変数の値を調べてから更新するまでの間に他者の処理が割り込んでくるかもしれないからである。厳密に排他制御を行うためには、UNIX システムのセマフォのように、ロック値の参照と更新を原子性を持った処理（不可分で、その間に割り込まれない処理）として行う必要があるが、Scratch ではそれはサポートされていない。

ただ、Scratch では、プログラムの実行速度があまり速くないことや、スプリクトの実行が画面更新と同期することなどもあって、厳密な排他制御を行わなくても問題が生じない事が多いように感じる。データブロックの“リスト”を待ち行列として使う事で厳密な排他制御行う方法も提案さ

れているので[3]、必要な場合にはそうした方法を用いることで対応できるであろう。

③クローンとオリジナル

同じスプライトを複数登場させるのであれば、クローンを作る必要がある。図3-1のようなゲームの場合も、出現したり消えたりするオブジェクトは全てクローンである。そうした場合、オリジナルのスプライトはクローンの属性・データの初期化処理とクローン生成だけが役割である場合も多い。Scratciでのプログラミングでは、クローンの扱いが大変重要だと言える。

クローン自身は自分固有の属性値やデータを持て、自分自身についての判断はできる。しかし、他者がクローン同士を識別する事はできない。例えば、他のスプライトのクローンに触れた場合、自分自身が反応することは可能だが、触れた相手に何かさせることは難しい。自分が触れた相手の事は、どのスプライト（及びその派生クローン）かということ以外、相手の持つ属性値、更に相手がオリジナルなのかクローンなのかも判断できない。こうした点は念頭においておく必要がある。

実際に、特定のクローンに対して何らかのアクションを起こさせるのであれば、共有変数に条件値をセットしてメッセージを送信し、メッセージを受信した側が、自分が該当条件に合致するかどうかを判断して対応するといった手段をとる必要がある。

④データ構造

前述の通り、Scratchでは変数とリスト以外のデータブロックは作れない。リスト中の要素には添数（先頭から何番目の要素か）を指定して値参照・値置換・挿入・削除を行う事ができるが、リストを切断したり、部分リストとして取り出したりという操作は簡単にはできない。リストの用途としては、配列の代替、スタック、キューとして使用する形になるであろう。

データ構造に関して言えば、Scratchではあまり複雑なデータ構造は構築できないと考えるべきである。

⑤定義ブロック

Scratchではユーザーが定義できる“定義ブロック”を使用できる。これは他のプログラミング言語のサブルーチンに相当するもので、数値、文字列、論理型の引数を設定することがきる。ただし、定義ブロックは関数（ファンクション）のように値を返すことはできないので、処理結果をデータとして呼び出し側に渡す為には、変数やリストに格納して定義ブロックを終了することになる。その場合、複数のスクリプトやクローンから呼び出される可能性があるのであれば、値を格納するデータブロックへのアクセスに関しての排他制御が必要となる。

定義ブロックの大きな特長として、定義ブロック内で自分自身を呼び出すという、再帰的呼び出しが可能な点が挙げられる。対象となる問題を自己相似な部分問題に分けることができる場合に、再帰処理をうまく用いればスマートにプログラムを記述できる。ただし、定義ブロック内でローカルな自動変数を作成する事はできない。また、引数も値を参照する操作だけが可能で、値の置換等はできない。従って、再帰処理内で自動変数の類が必要な場合には、リストを用いて自分でスタックを準備しておく必要がある。

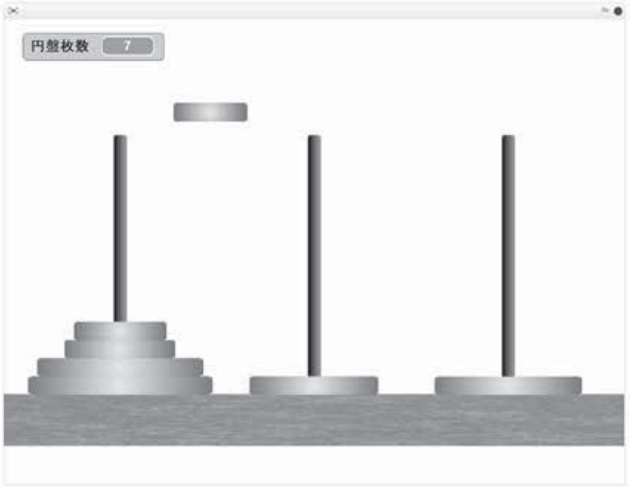

図5-1：ハノイの塔

ハノイの塔のプログラミング例

Scratch を用いたプログラム作例として、「ハノイの塔」のプログラムについて解説する。「ハノイの塔」は再帰処理の例として良く取り上げられるパズルである。三本の柱がたっていて、そのうちの一本の柱に複数枚の円盤が刺さっている。円盤はそれぞれ大きさが異なっていて、小さな円盤の上に大きな円盤をのせることはできない。柱から柱へ、一度に 1 枚の円盤を移動させながら、全ての円盤を別の柱に移動させるにはどうすれば良いか？という問題になっている。

「ハノイの塔」のプログラムの考え方はリスト 1 に示すような再帰的なものとなる。

リスト 1 のプログラムでは、n 枚の円盤を移動させる処理は、n-1 枚の円盤を移動させることができれば実現することになる。さらに n-1 枚の移動の為には n-2 枚の移動ができれば…としていけば、最終的に 1 枚の円盤移動に帰結できる。

Scratch で作成したプログラムの実行中画面を図 5-1 に示す。円盤の最大枚数は 7 枚として、円盤スプライトには 7 通りの大きさのコスチュームを用意しておき、コスチュームを替えながら 7 個のクローンを作る。地面と柱は開始時に所定の位置に配置して、その後は動かす事はない。

柱には番号（1 ～ 3）が設定されているものとし、共有リスト “towers” に各柱に刺さっている円盤の枚数を格納しておく（“towers” の 1 番目の要素に 1 番の柱に刺さっている円盤の枚数がセットされるようにする）。各円盤のクローンには、自身固有の変数として、“tower_id”（自分が刺さっている柱の番号）、“pos”（柱内で自分が下から何番目の円盤かという位置）を持たせる。また、円盤が移動中かどうかを判別する為のフラグとなる共有変数“移動中”も必要となる。

プログラムの中核は、図 5-2 に示す、“move_one_disk” 及び “move_disks” という二つの定義ブロックとなる。ただし、個々の円盤クローンを “move_one_disk” 内から識別する事はできないの

リスト 1　ハノイの塔のプログラム

```
「“ 移動元の柱 ” から “ 移動先の柱 ” に “n” 枚の円盤を動かす」には {
        n が 1 なら {
                「“ 移動元の柱 ” から “ 移動先の柱 ” に円盤を一枚動かす」
        }
        そうでなければ {
                「“ 移動元の柱 ” から “ もう一つの柱 ” に “n-1” 枚の円盤を動かす」
                「“ 移動元の柱 ” から “ 移動先の柱 ” に円盤を一枚動かす」
                「“ もう一つの柱 ” から “ 移動先の柱 ” に “n-1” 枚の円盤を動かす」
        }
}
```

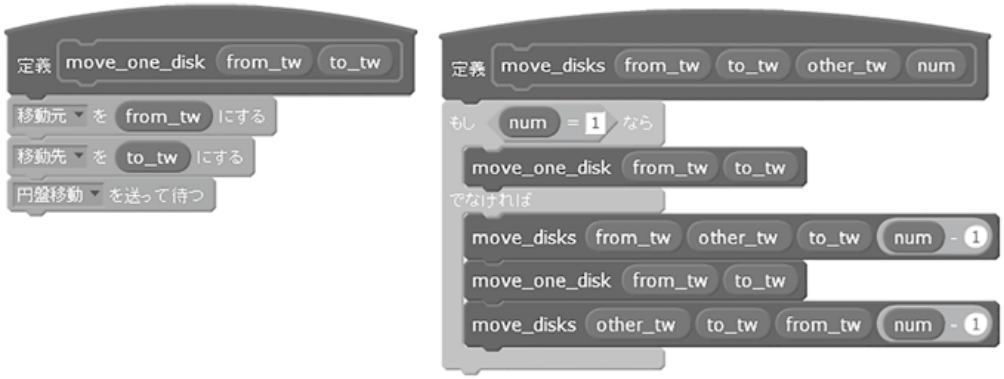

図 5-2：定義ブロックによる再帰呼び出し

で、“移動元”“移動先”という共有変数に柱番号を格納してから、“円盤移動”メッセージを発信して待つ。

メッセージは全ての円盤クローンが受け取ることになるので、各円盤クローンは共有変数“移動元”と自分が刺さっている柱番号“tower_id”を比較し、自分が該当の柱に居るなら、柱に刺さっている円盤の枚数と自分の“pos”を比較する。自分が柱の一番上の円盤であれば、“移動先”に向かって動き出すアクションを起こす。

入門用プログラミング環境としての Scratch

Scratch は気楽に使い始めることができ、事前に予備知識を覚えておくことなくプログラミングできる他、動作結果が視覚・聴覚的に分かり易いプログラムが簡単に作成できるなど、入門用の環境としては大変に優れている。また、並列実行に伴う排他制御なども、実際にプログラムを作成してみると、それほど厳密に施さなくとも問題が生じない場合も多い。

一方、複雑なデータ構造の構築が困難である事もあり、「何でも作れる」とは考えないほうが良いとも言える。プログラムとしての実現方法を色々と工夫することはプログラミングの醍醐味ではあるが、「プログラミングの入門用」として考えると、あまりトリッキーな手段を追求しても仕方がないであろう。入門用システムとして、Scratch に向いている演習題材を選びながら、その中でプログラミング的思考の訓練となるものを選択していくべきだと考える。

冒頭に述べたように、児童学科の「ICT の基礎２」の授業では Scratch によるプログラミング演習も行っている。これは、学生にプログラミングとはどういった作業であるかを教えること、及び、卒業後に小学校等で Scratch に触れる機会があった場合に困らないようにといった点を目的としている。今後は、学生に自分で創意工夫しながらプログラムを作成する作業を増やせるよう、演習題材を更に検討していきたい。

参考文献・ホームページ

［1］文部科学省：小学校段階における論理的思考力や創造性、問題解決能力等の育成とプログラミング教育に関する有識者会議、小学校段階におけるプログラミング教育の在り方について（議論の取りまとめ）、平成 28 年

［2］“Scratch - Imagine, Program, Share” https://scratch.mit.edu

［3］喜家村 奨、並列処理プログラミング教育における Scratch の可能性についての考察、帝塚山学院大学　人間科学部研究年報、平成 28 年