解説

プログラミング教育者のための Scratchの並列処理機能についての解説

帝塚山学院大学教授　喜家村　奨

1．はじめに

新学習指導要領において，高等学校の情報科では，情報Ⅰが必履修科目となり，その中に「コンピュータとプログラミング」という項目が追加された。このような状況の中，いかにプログラミングを教育するかが情報教育の課題の１つになっている。一方，初心者のプログラミング教育ツールとして世界で利用されているツールの１つに米国MITで開発されたScratchがある。Scratchはビジュアルプログラミング言語の１つで，ブロックを組み立てるようにプログラミングができ，細かな文法ミスなどにとらわれずにプログラムの構造を設計することに注力できる。また，主に画面上のキャラクタを操作する（動かしたり，色を変えたり，音を出したりする）ゲームを作る感覚でプログラミングができるようになっており，画面をみながら直感的に自分のプログラムが正しく動いているかを確認できる点で初心者にも有用であるとされている。また，最近では，英国BBCが開発した教育用ワンボードコンピュータmicro:bitの開発環境，MakeCode Editorも，同様にビジュアルプログラミング言語でプログラミングができ，STEM教育を見据えたフィジカルコンピューティングを体験できるようになってきている。

しかし，忘れてはならない点は，これらのビジュアルプログラミング言語で記述したプログラムは**（イベント駆動型の）並列処理プログラム**であるという点であり，並列処理特有の同期や排他制御の必要性について，少なくとも指導者は理解しておく必要がある。また，高等学校では，Pythonなどのテキスト型のプログラミング言語を利用するため，Scratchの並列性について，理解する必要はないという考え方も可能かもしれないが，学生達が小・中学校で，このプログラミングスタイルを学んでくること，これから，このようなイベント駆動型の並列処理が一般化される傾向にあることを考えると，高等学校の情報科教員も並列処理プログラミングについて理解しておくことは必要であろう。そこで，本稿では，Scratchの並列処理機能について，検証し，プログラミングする上での注意点を説明する。

2．多重送信時の処理について

Scratchの通信処理について述べる前に，一般的に，送信が多重に発生した場合，受信側がどのように振舞うかを述べる。

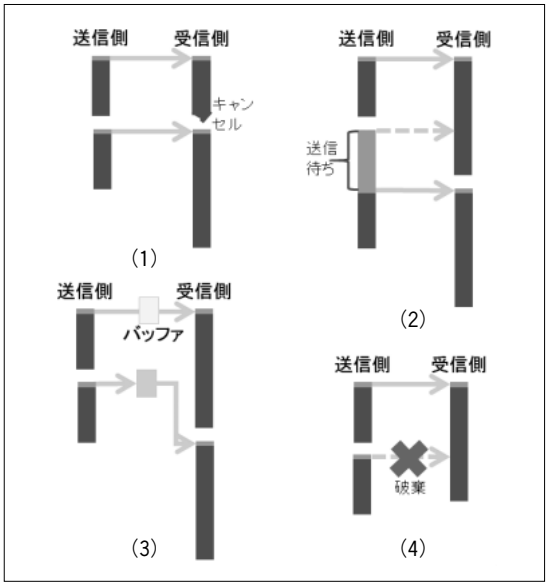

図１　多重送信の対処例

図１の(1)は受信側が処理中に，再びメッセージが送信されてくると，現在，実行している受信

側の処理がキャンセルされる。(2)は受信処理が終わるまで，送信側が待つ。いわゆる同期処理である。(3)は受信メッセージがバッファーリングされる場合で，これはBluetoothなどの無線インターフェイスを用いた通信時の処理である。(4)は処理中にメッセージが送られてくると，そのメッセージは受け付けずに破棄される。

このように，送信が多重に発生した場合の受信側の処理にはいくつかの場合が考えられる。では，Scratchで多重に送信が発生した場合，どのように振舞うのか次章で検証してみる。

3．Scratchのおける並列処理機能の実現

Scratchには，いくつかの並列処理のための機能がある。ここでは，並列実行機能，通信機能について，例をあげて解説する。また，本稿では，並列動作する実体について，一般的な並列処理に関する文章では“プロセス”と表記し，Scratchのプログラムの説明に関する文章では,“オブジェクト（またはスプライト）”と表記することにする。

3.1　並列プロセスの記述

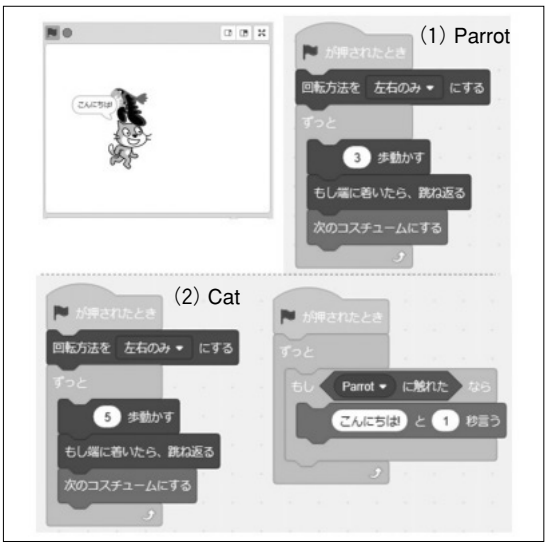

図2　Scratchにおける並列プロセスの記述例

最初にScratchにおける並列プロセスの記述方法について説明する。CやJava言語などのプログラムでは，ライブラリを利用するなど特殊な方法を利用しないと並列動作するプロセスを記述することができない。Scratchでは，発生したイベント(例えば緑の旗が押されたなど)に対して，それぞれのオブジェクトがどのように振舞うかを記述していく。このようなプログラムをイベント駆動型のプログラムという。またそれぞれのイベントごとの処理をイベント(メッセージ)ハンドラという。

図2はCatとParrotがステージ上を行き来するプログラムで，(1)がParrotのプログラム，(2)がCatのプログラムである。CatとParrotは，それぞれ，緑の旗がクリックされると動き出し，端にあたると反転する。この2匹の動作は全く独立して実行される。このようにScratchでスプライトのためのプログラムを記述することは，並列動作する個々のプロセスを記述することに他ならない。さらに，(2)のCatのプログラムは緑の旗が押されたとき，歩き回る処理とParrotに挨拶する処理の2つに分けて記述されている。つまり，2つのイベントハンドラが記述されている。この2つのイベントハンドラも同時並列に実行される。このようにScratchでは，イベントハンドラが並列実行の単位となっていることを理解しておくことが大切である。

3.2　プロセス間通信

並列動作するオブジェクト同士が情報を交換，または同期をとるためには，プロセス間での通信が必要である。通信の方法としては，共有変数を利用する方法と，メッセージを送って通信する方法の2つが考えられるが，Scratchではこの両方の方法が利用可能である。以下，それぞれの方法について説明する。

(1) 共有変数を利用した通信

共有変数による通信では，あるオブジェクトが共有変数に値を書き込み，それを他のオブジェクトが読み出すことによって情報を交換（または同期）する。Scratchで変数を定義すると，それは標準で共有変数となり，どのオブジェクトからでもアクセスできるようになる。ただし，どのオブジェクトからでも読み書きが可能で，しかも上書きができてしまうので，書き込みを制限するなどのアクセス制御が必要な場合は，プログラムの作成者が注意しなければならない。

(2) メッセージ通信

メッセージ通信する場合，Scratchには送信用ブロックとして「メッセージを送る」ブロックと「メッセージを送って待つ」ブロックの2種類のブロックが用意されている。送信側はこの2種類のブロックのうち，いずれかのブロックを使ってメッセージを送信する。受信側は「メッセージを受け取ったとき」ブロックで通信を待つ。送信側が送信ブロックを実行すると，ただちに受信側がメッセージを受け取り，処理が実行される。またScratchのメッセージ通信の形態はブロードキャスト通信であり，受信側が複数のオブジェクトでも構わない。送信が起こったときに，受信ブロックで待っている全ての受信オブジェクトがそのメッセージを受け取って処理を実行する。

3.3 Scratchのメッセージ通信

(1)「メッセージを送る」ブロックを用いた通信

まず「メッセージを送る」ブロックを使った通信についてみてみる。図3の(1)がメッセージを送信するCatのプログラムである。このプログラムではメッセージを3回，繰り返し送信し，送信する度に，共有リスト ログに送信したことを示す値“送信済”を書き込んでいる。このログへの書き込みは，後で通信の検証の時に用いる。(2)はParrotの受信プログラムで，メッセージを受信すると直ちに受信したことを示す値“受信回数”を共有リスト ログに書き込んでいる。図3 (3)がログに記録された通信記録であり，“送信済”と“受信回数”が3回書き込まれており，通信処理が正常に完了していることが分かる。

次に，先ほどのプログラムで，受信側のイベントハンドラ（Parrotのプログラム）において，受信後，計算処理を実行し，いくらか時間がかかる場合，処理がどうなるか検証してみる。

プログラムの変更は，送信側はそのままで，受信側のイベントハンドラのみメッセージの受信後，計算処理をし，処理の最後に再びログに“受信処理完了”と書き込みするように変更する（図4 (1)）。このように変更し，ログに書き込まれた値を確認すると（図4 (2)），送信側のログへの書き込みは3回行われている。しかし，受信側は受信直後の書き込みは3回行われているが，計算処理後の“受信処理完了”の書き込みは，最後の1回しか行われていない。つまり，これは，受信側のイベントハンドラが処理中に次のメッセージを受信すると，実行中の処理がキャンセルされることを表している（図1 (1)の処理）。このことは，送信した側は受信側が処理を途中でキャンセルされたことが分からないので，場合によっては処理に矛盾が生じる。

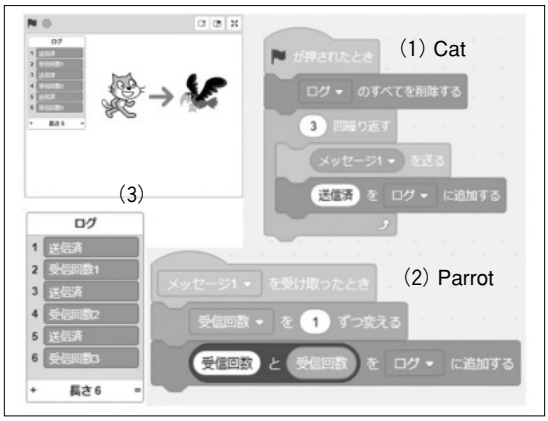

図3 「メッセージを送る」ブロックを用いた通信例

図4 計算処理を加えた受信側とログ

この問題を回避するためには，送信側のブロックを「メッセージを送って待つ」ブロックに変更するとよい。次に同じ例を「メッセージを送って待つ」ブロックを使って検証してみる。

(2)「メッセージを送って待つ」ブロックによる通信

図5が送信側を「メッセージを送って待つ」ブロックに変更したプログラムと，その時のログの結果である。このログの書き込み結果をみると，「メッセージを送って待つ」ブロックを使った場合は，受信側の計算処理後の書き込み“受信処理完了”も正しく3回書き込まれていることが分かる。これは，送信側が，受信側のイベントハンド

ラの処理が終了するまで待つことを示している。

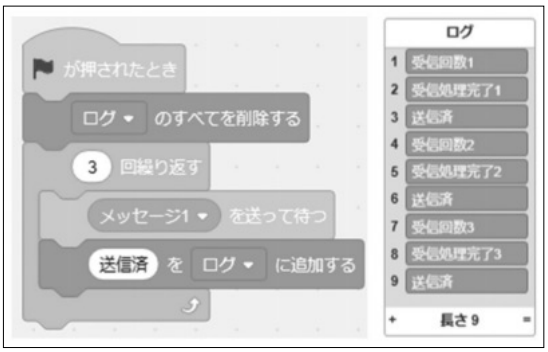

図5 「メッセージを送って待つ」ブロックに変更したプログラムとログ

(3) 送信が多重に起こったときの処理

次に送信側のオブジェクトが複数存在する場合は，どうなるか検証する。先ほどのプログラムの送信側のプログラム（Catのプログラム）をコピーしてDogのプログラムを作る。このプログラムを実行すると，CatとDogがParrotに対してメッセージを送信する。ログを確認すると図6のように，「メッセージを送る」ブロックを使ったときと同様に，受信処理が途中でキャンセルされている。この例のように，複数のオブジェクトが同時にリクエストを実行する場合は，後で，説明する排他制御のしくみを導入する必要がある。

図6 複数のオブジェクトが送信

(4) その他のScratchのおける通信処理の注意事項

この節の最後に，その他のScratchの通信処理における注意事項について述べる。例えば，図7のように「メッセージを送って待つ」ブロックを対向で使うと，どちらのイベントハンドラも「メッセージを送って待つ」ブロックが終了しないので，それ以下の処理（この場合はリストログへの書き込み）が実行されることはない。よって，対向で，「メッセージを送って待つ」ブロックを使わないように（メッセージループができないように）注意しなければならない。

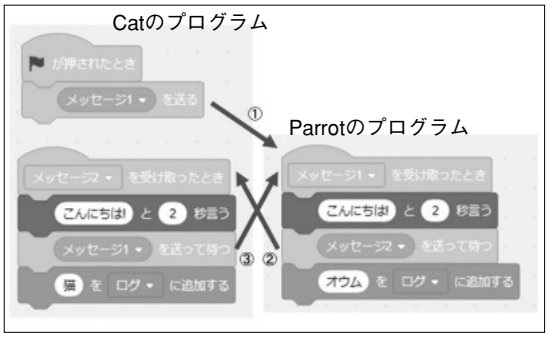

図7 「メッセージを送って待つ」ブロックを対向で使った例

3.4 並列処理のための他の制御方法について

複数のオブジェクトが並列に処理する場合は，それらの共有するリソースへのアクセスを排他的に制御する必要がある。排他制御する方法としては，セマフォを用いる方法などが考えられる。Scratchでは共有変数を用いてセマフォを簡単に実装できる。また，複数のメッセージによるリクエストを待ち行列に格納して，順次，処理することも，システムの状態を厳密に管理するために有用である。詳細はここでは紙面の都合上割愛する。

4．おわりに

本稿では，Scratchの並列処理機能のうち，特にメッセージ通信機能について，その動作を確認検証しながら，プログラミングのために注意点を述べてきた。ScratchやmicroːbitのMakeCode Editorのようなイベント駆動型の並列処理プログラミングが初心者のプログラミング教育に利用される現在，本稿で述べた内容が，これからプログラミングを学習する学習者，また，プログラミング教育に携わる指導者のために有意義なものになれば幸いである。

※今回の検証プログラムは2019年8月にWeb上に公開されているScratchの環境で実行し検証している。今後，Scratchの更新により異なる挙動する可能性もある。