ロボット博士養成講座

ロボティクスプロフェッサーコース

不思議アイテムⅢ-2③

第6回

ゲームプログラムを読み解こう

講師用

目　　次

○　授業開始にあたって

授業のはじめは、着席させ、大きな声であいさつしてから始めます。

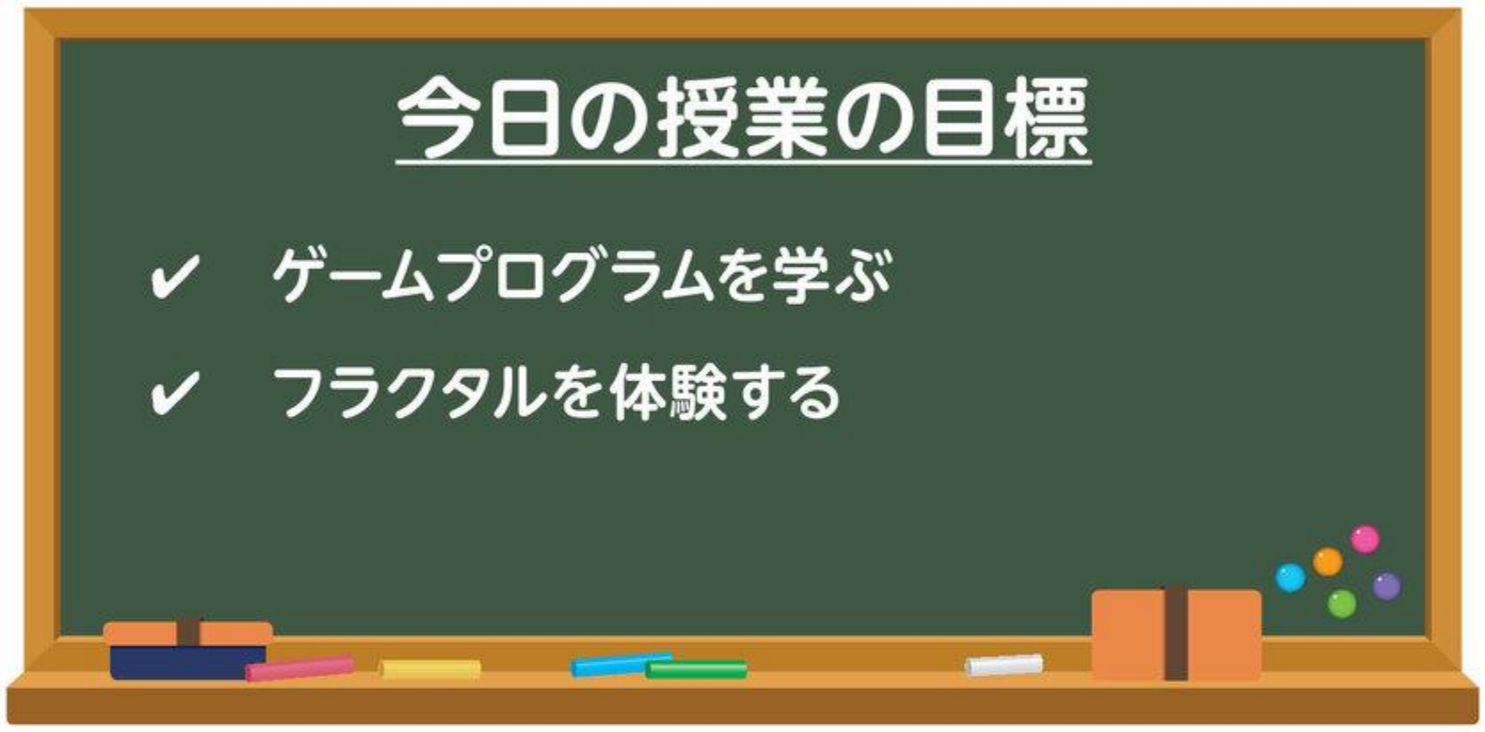
○　今回の目標をパネルで用意するか、黒板に予め書いておきます。

（授業の目標を明確化することは大変重要なことですので、生徒によく理解させます）

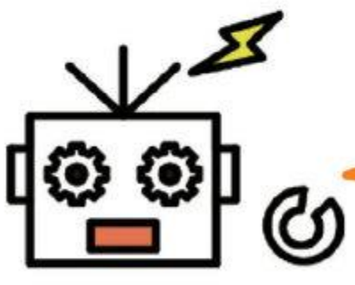
目安時間は授業時間 120 分のうち、休憩 10 分程度取ることを想定しています。
生徒の進捗状況により、休憩時間などを調整して授業を行ってください。

0. ゲームプログラムを読み解こう（目安５分）

0.0. 「ゲームプログラムを読み解こう」でやること



今回は前回ものぞいてみたゲームプログラム「Snake」を細かく読み取り、各種機能がどのようなつくりになっているかを確認していきましょう。ロボット制御用のプログラムではありませんが、いつも自分でかいているプログラムと構造や使用構文は似通っていることが実感できると思います。
また、後半では、液晶ディスプレイの集大成、コンピュータグラフィックスの技術を用いたフラクタルという不思議な世界を体験します。前回の応用ですが計算式を見ただけではどんな図柄になるか、ほとんど想像できないと思います。とてもシンプルな数式で複雑（ふくざつ）な絵をかけるという、不思議な理論です。さらに言えば、自然にある、木であったり葉っぱであったり、というものは「実は数学、数式と関わりがあるのではないか」というお話に発展します。そんな不思議な世界を少しだけ体験してみましょう。



ゲームとロボットプログラミングはとても似ているのダ！

0.1.　必要なもの

以下のパーツを準備しておきましょう。

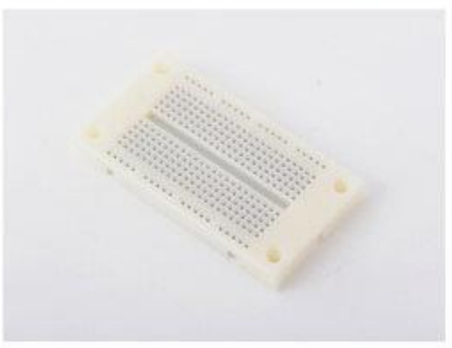
USB ケーブル 1	マイコンボード 1	ロボプロシールド 1	301 ブレッドボード 1
			
ジャンパー線 65	**タクトスイッチ 10**	**姿勢検出シールド 1**	**液晶ディスプレイシールド 1**
			

図 0-0　必要なもの

0.2.　マイコンユニットの準備

前回同様に組み合わせたマイコンユニットに、ブレットボードの回路を接続します。マイコンユニットの組み合わせは、下からマイコンボード、ロボプロシールド、姿勢検出シールド、液晶ディスプレイシールドの順番です。

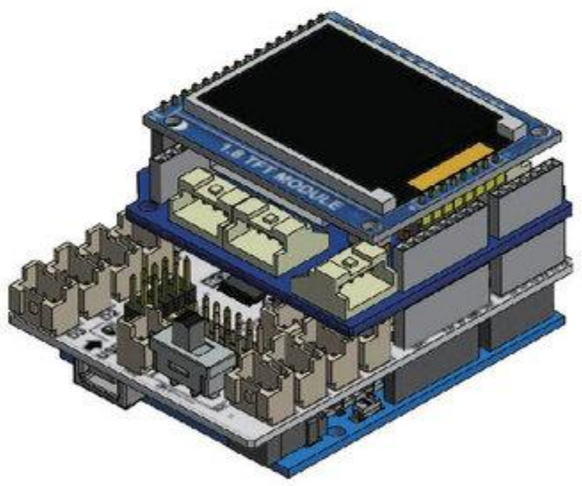


図0-1　マイコンユニット

1．ゲームプログラムを学ぶ（目安 70 分）

1.0. 「Snake」のメインループ

では、前回と同様プログラム「Snake」を見ていきましょう。

タクトスイッチの接続も前回と全く同じですが、もう一度配線が必要な人は下の図を参照してください。

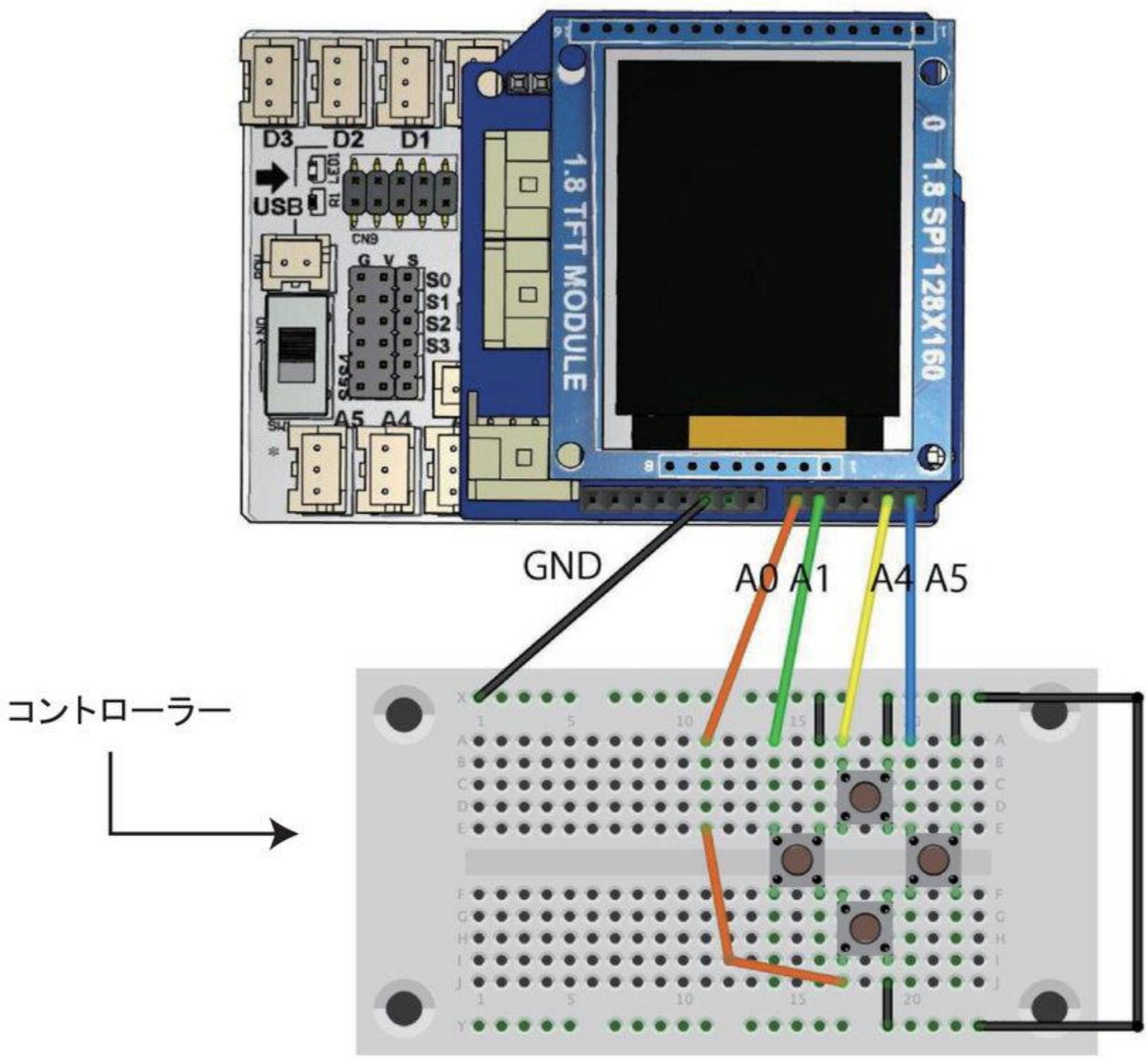


図1-0　タクトスイッチ接続の回路

今回は、「Snake」のメインループ部分を見るところから始め、そのあとメインループ内に使われている各種関数をひも解いていきます。

具体的にこのゲームプログラムはどのような機能でできているのか、それぞれの機能はどのような動作が組み合わさっているのか、なるべく読み解いていきましょう。

確実に理解ができるように、1つひとつの機能についてジックリ説明していきますが、今回の目的は自分の力でプログラムの複雑（ふくざつ）な処理（しょり）も把握（はあく）できるような読解力をつけることなので、「自力で読み取れそうだ！」と思えるようになったら必要に応じて読み飛ばしても構いません。

では、まず void loop() を見てみましょう。

プログラム「Snake」より抜粋（ばっすい）

```
// メインループ
void loop(){
    uint8_t b = readButton();          // コントローラ読み込み
    if (b != BUTTON_NONE && b != BUTTON_SELECT)
        buttonPressed = b;
    if (!collision){
        appleLogic();
        checkIfAppleGot();
        specialApple();
        checkIfSpecialGot();
        specialTimer();
        updateSnakePosition(buttonPressed);
        if (timer >= snakeSize - 1)
            removeLastFromTail();
        else
            timer++;
    }
    else {
        if (displayEnd == true)
            displayEndingScreen();
        showTitle = true;
        uint8_t buttonPressed = readButton();
        if (buttonPressed == BUTTON_RIGHT) ///**************
            setup();
    }
    delay(difficulty);
}
```

黄色い部分が、このプログラム内で設定されているオリジナル関数です。

ヘビの動きに関わる関数だけに着目して、1つずつ見ていきましょう。

1.1. 各種機能を読み解いていく

1）readButton()

名前の通り、Button（ボタン）の状況を読み取った際の処理を担当する関数です。

プログラム「Snake」より抜粋

```
uint8_t readButton(void){
    if(digitalRead(A0) == LOW) return BUTTON_DOWN;
    else if(digitalRead(A1) == LOW) return BUTTON_LEFT;
    else if(digitalRead(A4) == LOW) return BUTTON_UP;
    else if(digitalRead(A5) == LOW) return BUTTON_RIGHT;
    else return BUTTON_NONE;
}

// メインループ
void loop(){
    uint8_t b = readButton();          // コントローラ読み込み
    if (b != BUTTON_NONE && b != BUTTON_SELECT)
        buttonPressed = b;
```

関数 readButton() は、 A0 、 A1 、 A4 、 A5 のスイッチが押されたとき、それぞれ1、5、4、2という数値を返すという処理になっています。プログラム内では BUTTON_DOWN などといった文字列に置きかえられていますが、おなじみの #define 部分を見ればわかりますね。

void loop 内では b という変数に readButton() から返ってきた値を入れ、それをさらに buttonPressed という変数に移しています。この buttonPressed が今後たびたび登場するので覚えておきましょう。

2）updateSnakePosition()

この関数は、Snake（ヘビ）のPosition（位置）に関する処理を行っています。
プログラムの中身を見ていく前に、「恐らくこういう処理がかかれているはずだ」というのを予想しておきましょう。

やってみよう！

「ボタンを１回押したらヘビが１マス移動する」という動作を行ったとき、内部的にはどのような処理があったかな？
具体的に１つずつ挙げてみよう！

「押されたボタンに応じてｘ座標、ｙ座標の値をかえる」

「かえた座標が画面端に届いていたらはみ出ないように処理する」

「決まった座標を白く塗る」

「塗った座標を配列に記録する」

「ヘビが長くなりすぎないよう後ろのマスを消す」

整理できたでしょうか。

実際、関数 updateSnakePosition() 内には、主に以下のような処理がかかれています。

POINT

・変数 buttonPressed の値に応じてかくマスのｘ座標、ｙ座標の値を変更する。
・ヘビが画面端に来るようなら、枠からはみ出さないように値を直す。
・決定した座標を中心に、3 × 3 の範囲を白く塗る。
・塗った座標を配列 x[head] と y[head] に記録する。
・ヘビが障害物（壁や自分の体）にぶつかっていないかチェックする。
・配列の添え字 head を１増やす。
・ヘビの体のうち、体長を超えた部分を消す。
・配列が上限に達したらリセットする。

これを踏まえたうえで、関数関数 updateSnakePosition() 内を確認していきます。
動作部分がかなり長いので、処理内容ごとに分けて見てみましょう。

プログラム「Snake」より抜粋

```
void updateSnakePosition(uint8_t buttonPressed){
    if (buttonPressed == BUTTON_UP){
        if(currentDirection != BUTTON_DOWN){
            pixelLocationYAxis -= 3;
            currentDirection    = BUTTON_UP;
        }
        else
            pixelLocationYAxis += 3;
    }

(中略)

    if (buttonPressed == BUTTON_RIGHT){
        if(currentDirection != BUTTON_LEFT){
            pixelLocationXAxis += 3;
            currentDirection    = BUTTON_RIGHT;
        }
        else
            pixelLocationXAxis -= 3;
    }
```

pixelLocationYAxis はヘビの描画位置のｙ座標を表します。上ボタンを１回ボタンを押すごとに 3 ずつ上に移動するということです。移動が１ずつでないのは、ヘビの体１マスが 3 × 3 ドットでできているためです。

やってみよう！

`if (buttonPressed == BUTTON_UP)` の分岐(ぶんき)内に `pixelLocationYAxis += 3;` という処理(しょり)が混じっているね。上ボタンを押しているのにヘビが下に移動するという意味だけど、どんなときにこの処理(しょり)が発生するのかな？プログラムから読みといて答えよう！

ヘビが下に移動している　　とき

ヒント

`A != B` は「AとBが等しくないとき」という意味だったね！

講 このゲーム内ではヘビが180度反対を向くことができない（頭と体が重なってしまうため）ので、反対方向を向くようなボタン操作は無効にしています。

プログラム「Snake」より抜粋(ばっすい)

```
if (pixelLocationYAxis < 10)
    pixelLocationYAxis = 10;
if (pixelLocationYAxis > tft.height() - 3)
    pixelLocationYAxis = tft.height() - 3;
if (pixelLocationXAxis < 1)
    pixelLocationXAxis = 1;
if (pixelLocationXAxis > tft.width() - 3)
    pixelLocationXAxis = tft.width() - 3;
```

画面端(はし)にヘビが来た時に、はみ出さないようにする処理(しょり)ですね。なお `height()` や `width()` は液晶ディスプレイの高さや幅、今回は128や160を返す関数です。

プログラム「Snake」より抜粋(ばっすい)

```
drawSnake(pixelLocationXAxis, pixelLocationYAxis, pixelColor);
x[head] = pixelLocationXAxis;
y[head] = pixelLocationYAxis;
collisionCheck();
head++;
```

drawSnake()という関数が、pixelLocationXAxis、pixelLocationYAxisの座標を中心に3×3マスを塗るという処理です。つまり、ここで実際に「ヘビを表示する」という動作を行っているというわけです。

続いて、配列に座標を記録しています。これは第5回で皆さんも製作した部分ですね。今回は添え字にiではなくheadという変数を使用しています。

ヘビが壁や自分の体にぶつかっていないかどうかは、collisionCheck()という関数で判定しています。
ちなみに、「collision」は「衝突」という意味です。

また、今回座標の記録に用いている配列x[]とy[]ですが、配列を宣言するときは添え字の範囲の指定が必要です。今回は以下のように指定されています。

プログラム「Snake」より抜粋

```
int head, timer, snakeSize, score, pixelLocationXAxis,
pixelLocationYAxis, x[300], y[300],appleX, appleY, yMultiplier,
selection = 100, difficulty, specialX, specialY, specialTime;
```

つまりx[299]、y[299]までの配列にしか座標を記録できないということです。変数headが300に到達してしまったら以下のような処理をしています。

プログラム「Snake」より抜粋

```
if (head >= 300){
    removeLastFromTail();
    resetArray();
    head = snakeSize + 1;
}
```

head = snakeSize + 1;の部分で、300に到達してしまった変数headを0付近に戻しているわけです。

snakeSizeという変数はヘビの体長＋1マス分の長さを示しています。snakeSizeが5のときは、headが300から6に置きかわることになります。

体長分を超えた体を消す処理と配列をリセットする処理は、それぞれremoveLastFromTail()とresetArray()という別の関数内にかかれていますので、あわせて確認してみます。

プログラム「Snake」より抜粋（ばっすい）

```
void removeLastFromTail(){
    drawSnake(x[head - snakeSize], y[head - snakeSize], ST7735_BLACK);
    x[head - snakeSize] = 0;
    y[head - snakeSize] = 0;
}
```

```
void resetArray(){
    for(int j = 1; j < snakeSize; j++){
        x[snakeSize - j] = x[head - j];
        x[head - j]      = 0;
        y[snakeSize - j] = y[head - j];
        y[head - j]      = 0;
    }
    x[0] = 0;
    y[0] = 0;
}
```

さまざまな変数が何度も登場していて、非常に紛（まぎ）らわしくなってしまっています。
このようなときに有効なのは、試しにそれぞれの変数に適当な値を代入して読んでみることです。

head は移動 1 回ごとに 1 ずつ増える変数で 0 ～ 299、snakeSize は今のヘビの長さを指定する変数で初期値は 5、その後はアイテムを取るたびに増減していきます。

今回、これらの関数は head が 300 になったときに使われていますから、試しに次のような状況を考えてみましょう。

ステップアップ

変数 head が 300、 snakeSize が 5 のとき、関数 removeLastFromTail()、resetArray() 内では具体的にどのような処理が行われているかかいてみよう！

removeLastFromTail()：

解答：(x[295], y[295]) の座標のマスを消し、x[295], y[295] に記録された座標は不要になったので 0 にする。

resetArray：

解答：x[296 ～ 299] に記録された座標を x[1 ～ 4] に、y[296 ～ 299] に記録された座標を y[1 ～ 4] にそれぞれ移し、x[296 ～ 299]、y[296 ～ 299] に記録された座標は不要になったので 0 にする。

これで添え字がリセットされ、再び [299] まで座標を記録していけるようになりました。「ボタンを押したらヘビが移動する」という実にシンプルな機能だけでも、これだけ複雑な処理が必要になってしまうわけです。

これ以外にもプログラム内にはさまざまな関数がありますね。どれも使われている構文そのものは今までに触れたことのあるものばかりですので、落ち着いて読んでみればどのような処理になっているか読み取れるものもあるはずです。ぜひ挑戦してみてください。

チャレンジ課題

プログラム「Snake」を読んで、各関数が具体的にどのような処理になっているか、可能な限り理解を深めてみよう！
ちなみに、各関数がそれぞれ何の処理を担当しているかは下の表にまとめておくから、参考にしてね。

関数名	担当している処理
showTitlScreen	起動直後のタイトル画面の表示
titleSelection	3段階の難易度を選択する画面の表示
drawBoard	ゲーム中の画面端の枠と、スコア部分の表示
updateScore	現在のスコアのリアルタイム表示
drawSnake	表示するヘビの形の指定
collisionCheck	ヘビが障害物に当たっているかどうかの判定
appleLogic	青りんごの場所の決定と表示
checkIfAppleGot	青りんごを取ったときの処理
specialApple	赤りんごの出現判定、場所の決定と表示
checkIfSpecialGot	赤りんごを取ったときの処理
specialSelection	赤りんごの効果をランダムに決定する処理
specialTimer	赤りんごが一定時間で消えるようにする処理
drawApple	表示するりんごの形の指定
displayEndingScreen	障害物に当たったときの処理

具体的な処理内容まで読み取れたら、りんごが取りやすくなるようにしたり、壁に何度かあたっても大丈夫なように「ヒットポイント」システムをつくったり、りんごを取ったときの得点をもっと高くしたりして、より自分に都合のいい「チート」を行っても面白いかもしれません！

2．フラクタルを体験する（目安30分）

2.0. フラクタルとは？

さて、ゲームからは話がそれますが、前回も扱ったpoint命令の活用についてもう1つお話ししておきます。

今回は「point命令を役立てる」というよりは、純粋に数学の世界の変わった考え方について紹介することが目的です。こういう考え方もあるのか、という事をなんとなくでもいいので感じてみてください。

図形の考え方の一つに「フラクタル」というものがあります。このフラクタルを、第5回と同様point命令を駆使して液晶ディスプレイに表示させてみましょう。

2.1. シェルピンスキーのギャスケット

「フラクタル」とは何か、簡単に説明すると、基本的には「自己相似（じこそうじ）」の性質をもつ図形のことです。

…と言われても、「自己相似（じこそうじ）」とは何かがわからないと何の解決にもなりませんね。

たとえば、黒い正三角形を想像してください。3つの辺の中点を結ぶと、半分に縮小された逆さまの正三角形ができますね。この正三角形の中を白く塗（ぬ）りつぶします。

図2-0　正三角形の中に正三角形をつくる

こうすることで、黒い正三角形が3つにわかれたともいえます。この3つの黒い正三角形にも、先ほどと同じ処理（しょり）をします。すると黒い三角形がさらに小さく分裂するので、そこにさらに同じ処理（しょり）を…とくり返していきます。

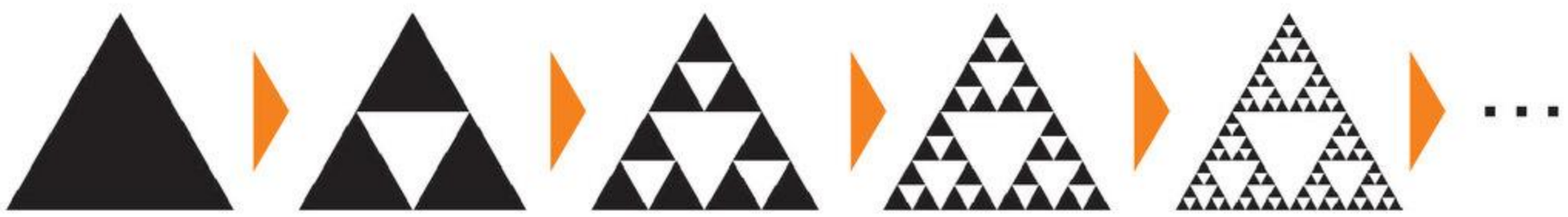


図2-1　同じ処理（しょり）を何度もくり返していく

この処理を「無限回」くり返したものがあるとしましょう。この図形は全体を見ても、分裂した小さな黒い三角形の1つだけを拡大して見ても、まったく同じ形のはずですね。このような性質のことを「自己相似」とよび、こういった性質をもつ図形をフラクタル図形と呼ぶのです。

ちなみに、この手順でつくられたフラクタル図形は「シェルピンスキーのギャスケット」といいます。

このフラクタル図形をプログラム上でつくると、以下のようになります。

プログラムの書き込み

RoboticsProfessorCourse3 > MagicItemLCD6 > Gasket1
RoboticsProfessorCourse3 > MagicItemLCD6 > Gasket2

複雑な見た目の図形をかく割には、プログラムはなかなかシンプルにまとまっています。そしてよく見ると、どちらのプログラムも実際に液晶ディスプレイに表示させるときは今までと同様 `point(x, y);` を使っているだけです。
フラクタル図形はほかにも有名なものがたくさんありますので、いくつかプログラムで紹介します。

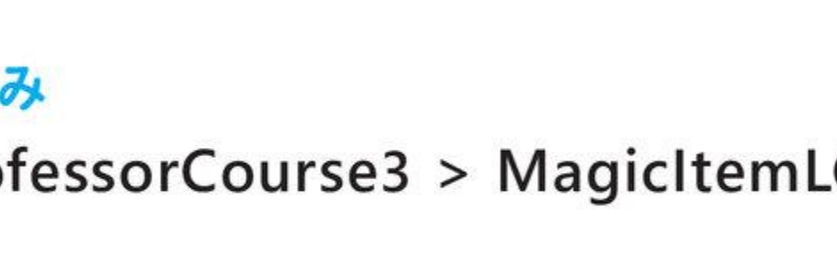
2.2.　いろいろなフラクタル図形

１）コッホ曲線

①　まず、横に一本線を引く。

②　線を三等分して真ん中に山をつくる。

③　できた四つの線分（辺）に対して同じ処理（しょり）を行う。



④　さらに各線分（辺）に同じ処理（しょり）を行う。これを無限回くり返す。

図2-2　コッホ曲線

プログラムの書き込み

RoboticsProfessorCourse3 > MagicItemLCD6 > koch

2）木

①　縦線をかく。
②　縦線の70％の長さの枝を縦線から35度ずつ左右に広がるようにかく。
③　できた枝に対して②の処理を行う。これを無限回くり返す。

これだけです。これをくり返して行うと木のようになります。以下のプログラムを実行してみてください。

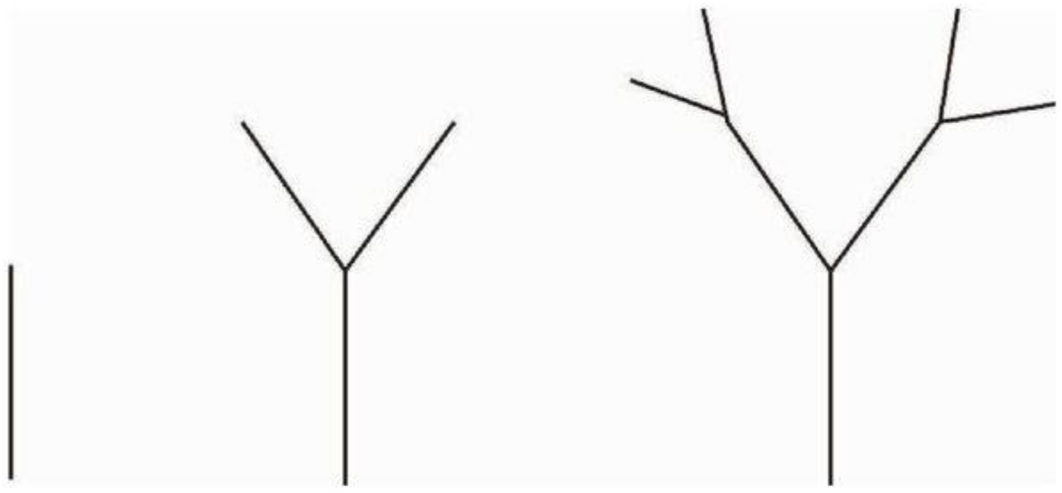


図2-3　木のフラクタル

プログラムの書き込み

RoboticsProfessorCourse3　>　MagicItemLCD6　>　drawTree1

プログラム「drawTree1」では、木としては不自然です。そこでランダムに枝を増やすようにアレンジすると次のプログラム「drawTree2」のようになります。

プログラムの書き込み

RoboticsProfessorCourse3　>　MagicItemLCD6　>　drawTree2

3）シダ

①　一本の線をかく。
②　幾何学的に枝をかく。
③　枝に対してさらに②の処理(しょり)を行う。これを無限回くり返す。
これだけです。実行すると驚くほどに自然のシダに似ていることがわかります。

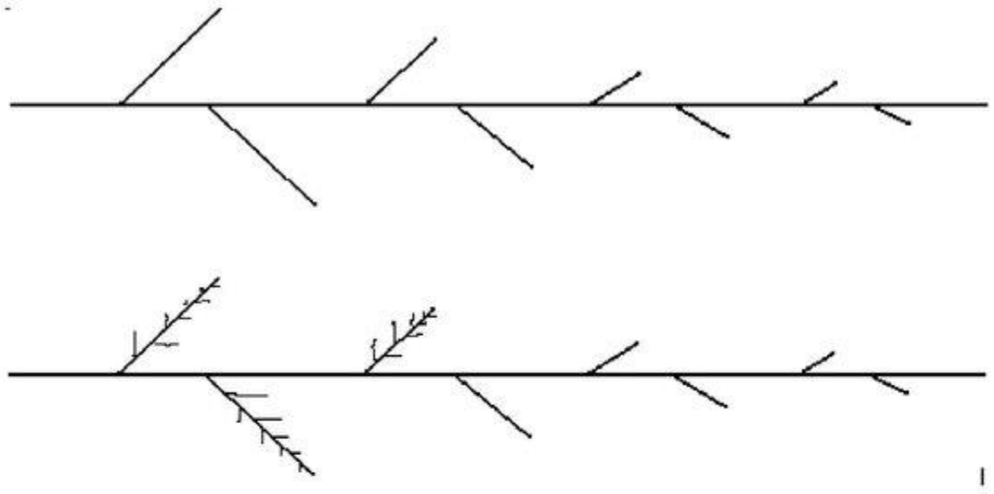


図2-4　シダ

プログラムの書き込み

RoboticsProfessorCourse3　>　MagicItemLCD6　>　Fern

4）マンデルブロ集合

プログラムの書き込み

RoboticsProfessorCourse3 > MagicItemLCD6 > Mandelbrot

「マンデルブロ集合」という、数学の世界の式の一種をグラフ化したものです。フラクタルについての話では、非常によく話題に出される図形です。

ちなみにマンデルブロ集合を研究・発見したポーランド出身の数学者ブノワ・マンデルブロは、フラクタルという考え方自体を発見し、「フラクタル」と名付けた張本人でもあります。

3．まとめ（目安5分）

液晶ディスプレイを使ったプログラミングを、6回分学び終えました。
ゲームプログラム、グラフの描画、フラクタル図形など、非常に複雑(ふくざつ)なプログラム・数学の世界に触れて目が回ってしまった人もいるのではないでしょうか？
しかしここで思い返してみると、どれもプログラムの中身は今まで学んできた構文を巧みに組み合わせていただけだと思いませんか？

「非常にシンプルな処理(しょり)をうまく組み立てて、非常に複雑(ふくざつ)な動作を創りあげていく」ことが、プログラミングの基本にして極意(ごくい)であるともいえます。
つまり「限られた知識・構文を、いかにうまく活用できるか」がプログラマーの腕のみせどころなわけです。

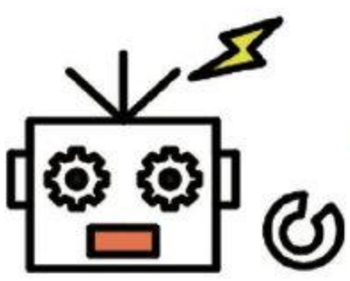
この「知識の活用」はなにも、プログラミングに限った話ではありませんよね。
テストで見たこともない問題を出されたら、今までに学んだ内容の中で活かせるものを探します。友人から悩みを相談されたら、自分の知らないことでもこれまでの経験と照らし合わせてアドバイスを考えるでしょう。あるいは大人になって他の人と仕事をするようになれば、自分が持っている判断材料だけで他人の考えを「推(お)し量(はか)る」場面もぐっと増えますよね。こういったときにも、プログラミングで鍛えた「知識の活用」の力は大いに役立つはずです。
プログラミングを学んでいくとロボットをカシコクできるだけでなく、皆さんの人生をより豊かにしてくれるような力が身につくというわけですね！

このテーマでプログラミングへの興味が増した、という人は、サンプルプログラムを色々とかきかえてみたり、参考にして新たなプログラムをつくってみたりと、さらに力を伸ばしてみてください。

次回は「いかにもロボット！」な感じの、2つの脚(あし)で地面を歩くロボットを製作します！
人間のようにさまざまなパーツを同時に動かす、複雑(ふくざつ)なロボットの制御を学びましょう！
お楽しみに！

講

○以下の理解度を確認してください。
・ゲームプログラムを学ぶ
・フラクタルを体験する
○次回から、「二足歩行ロボット」の製作を行います。次のページのパーツを準備させてください。



次回からは、二足歩行ロボットだヨ！

《次回必要なもの》

次回は以下のパーツを持ってきてください。

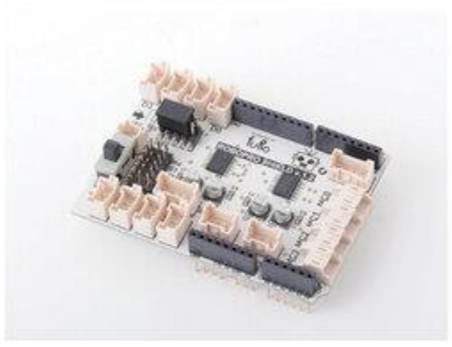
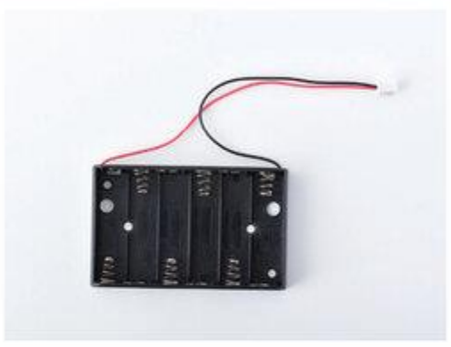
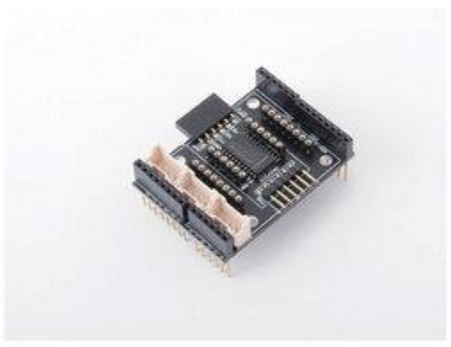
ラジオペンチ 1	ドライバー 1	USB ケーブル 1	マイコンボード 1
			
ロボプロシールド 1	電池ボックス 1	リボンケーブル 1	コントローラー 1
			
無線受信モジュール 1	マトリクス LED シールド 1	マトリクス LED 1	スピーカー 1
			
超音波距離センサー 1	センサーカバー 1	センサーケーブル 1	M3L6 タッピングネジ (B) 2
			
MG995 サーボモーター 3	サーボモーター付属パーツ 3	SG90 マイクロサーボモーターセット 1	AC アダプター 1
			
A-1 (二足歩行ロボットパーツ) 1	A-2 (二足歩行ロボットパーツ) 1	A-3 (二足歩行ロボットパーツ) 1	B-1 (二足歩行ロボットパーツ) 1
			

図 3-0　次回必要なもの①

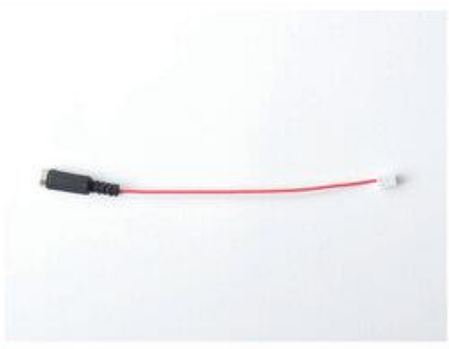
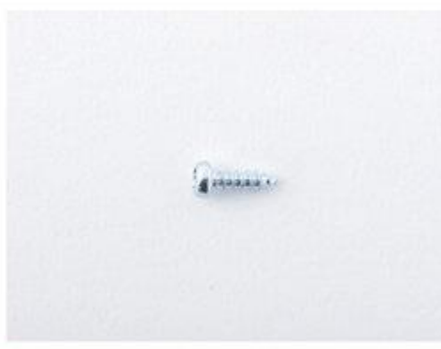
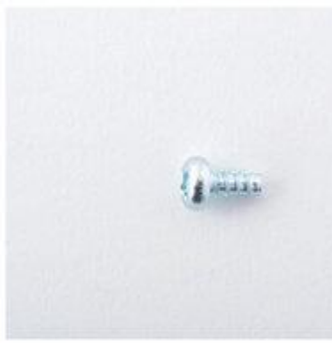
B-2 (二足歩行ロボットパーツ) 1	B-3 (二足歩行ロボットパーツ) 1	B-4 (二足歩行ロボットパーツ) 2	C-1 (二足歩行ロボットパーツ) 2
			
C-2 (二足歩行ロボットパーツ) 2	C-3 (二足歩行ロボットパーツ) 2	C-4 (二足歩行ロボットパーツ) 2	D-1 (二足歩行ロボットパーツ) 2
			
サーボホーン 3	SG90 マイクロサーボモーターセット 1	AC 変換ケーブル 1	M2L6 タッピングネジ (A) 8
			
M3L6 タッピングネジ (B) 4	M2L12 ナベワッシャータッピングネジ (A) 12	針金フォーミング加工 2	M2L8 タッピングネジ (B) 12
			
M3L10 タッピングネジ (B) 2	M3L6 フラットヘッドビス 13	オイレスブッシュ (二足歩行ロボット用) 32	
			

図3-1　次回必要なもの②