講師用

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５回/第６回テキスト）

必ず、生徒に授業日と自分の名前を記入させるようご指導をお願いいたします。

第５回授業日 2024年　　月　　日

第６回授業日 2024年　　月　　日

名　前

Human ヒューマンアカデミー ジュニア STE"AMスクール

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

2024年12月授業分

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5回

ゲームプログラムをのぞいてみる

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。

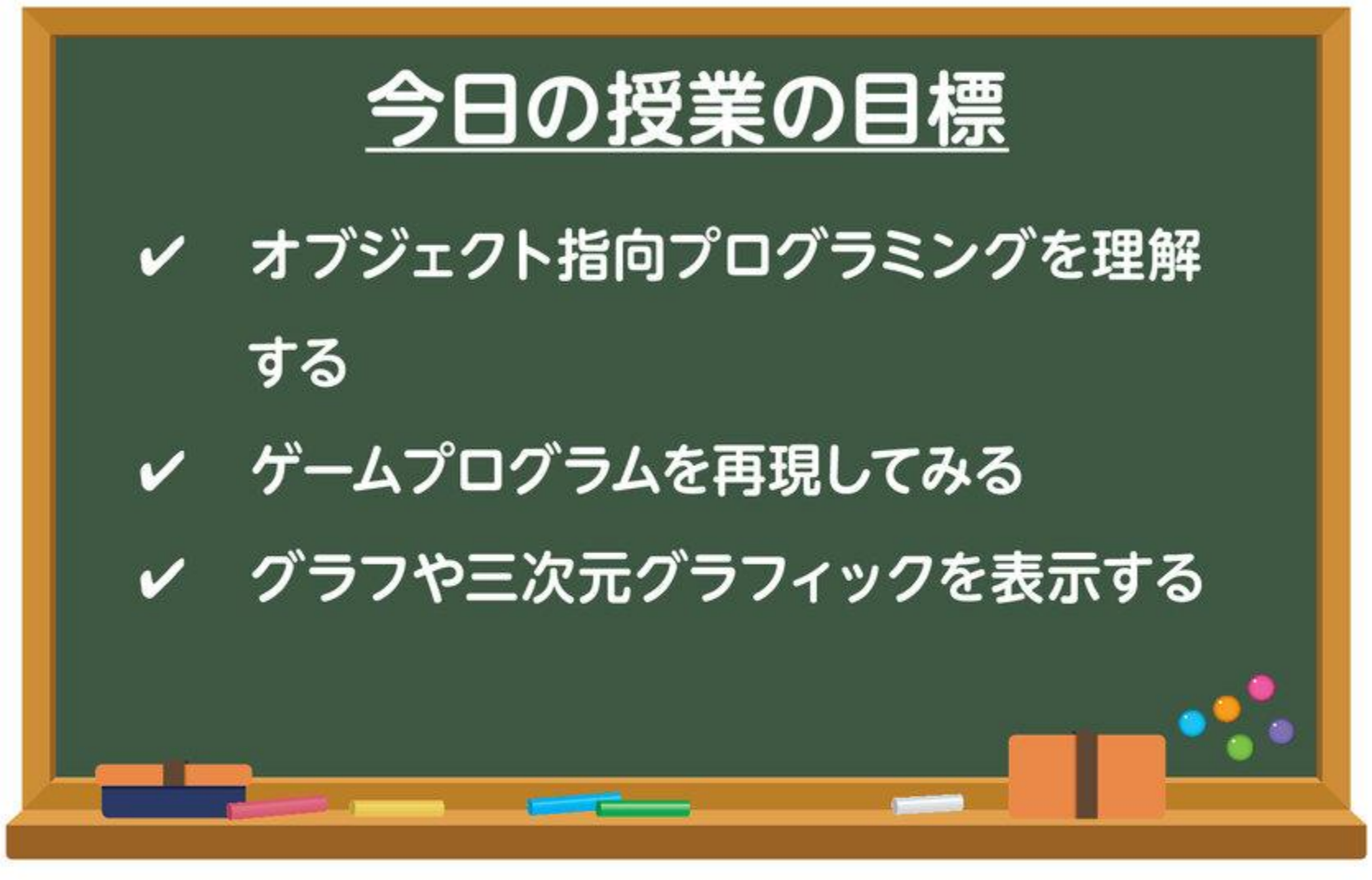
○　今回の目標をパネルで用意するか、黒板に予め書いておきます。

　（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のぞいてみる（目安５分）

0.0. 「ゲームプログラムをのぞいてみる」でやること



さて、「不思議アイテムⅢ-2」ではこれまでに、液晶ディスプレイにさまざまな図形やキャラクター画像を表示させる方法、赤緑青の数値を指定してさまざまな色を描画する方法、スイッチの状況によってキャラクターを左右に移動させる方法などを学んできました。
特に、スイッチを押すことで画面表示が変化し、その変化に応じて次の操作を考えて…という「インタラクティブ」な表示は、実際に世の中にあふれているさまざまな電子機器にも搭載されていますよね。
皆さんにとってわかりやすい「インタラクティブ」を利用した電子機器と言えば、やはりスマートホンやゲーム機などではないでしょうか。今回と次回では簡単なゲームプログラムを通じて、これまで学んできた構文や手法を振り返りましょう。
せっかくなので、実際にプロが行っているプログラミングについての知識にも触れつつ、プログラムについて更に深く学んでいきますよ！

0.1.　必要なもの

以下のパーツを準備しておきましょう。

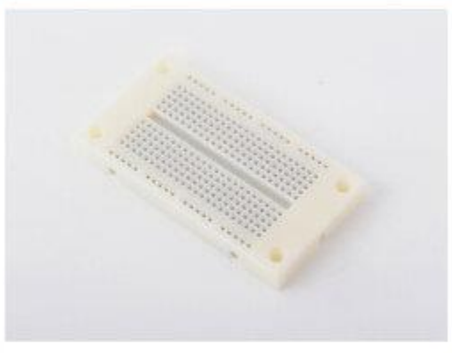
USB ケーブル 1	マイコンボード 1	ロボプロシールド 1	タッチセンサー 1
			
301 ブレッドボード 1	**ジャンパー線 65**	**タクトスイッチ 10**	**姿勢検出シールド 1**
			
液晶ディスプレイシールド 1			
			

図0-0　必要なもの

0.2.　マイコンユニットの準備

今回も、液晶ディスプレイシールドを組み込んだマイコンユニットを使います。

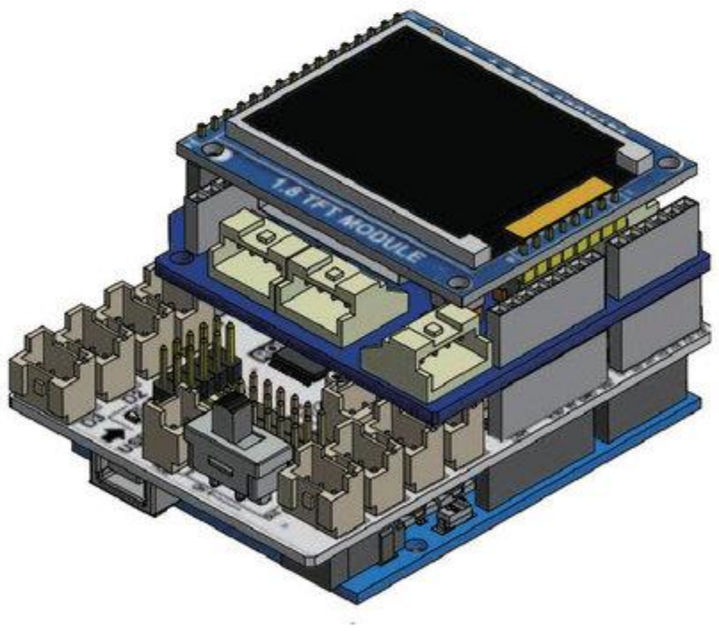


図0-1　マイコンユニット

1．オブジェクト指向プログラミングとは（目安15分）

1.0. 「オブジェクト指向」とは

「オブジェクト指向」、いきなり耳慣れない言葉が出てきましたね。
もともとプログラムというのは上から順番に動作手順をかいていけばとりあえずは機能するものです。しかし、コンピューターが普及し、時代とともに便利な機能をもったソフトウェアが開発されて、開発規模もどんどん大きくなってきました。
そうすると、プログラムを毎回イチから作成するのは手間も時間もかかってしまいます。プログラムの中で、使い回せる部分は使い回した方が便利ですよね。
たとえば、授業でもロボットをつくる過程で、モーターやセンサー、コントローラー、LEDなどのディスプレイなど同じ規格の製品を利用していますが、その都度使用するもののプログラムをイチから作成するのは効率的ではありません。あらかじめ、モーター、センサー、コントローラーなどのモノを機能させるプログラムを「ライブラリ」という形で部品化（オブジェクト化）することで再利用できるようにしていました。
この概念は世界で多くのプログラマーが使っている「C++」「C#」「java」「Ruby」「PHP」…といった多くのプログラム言語にも共通します。

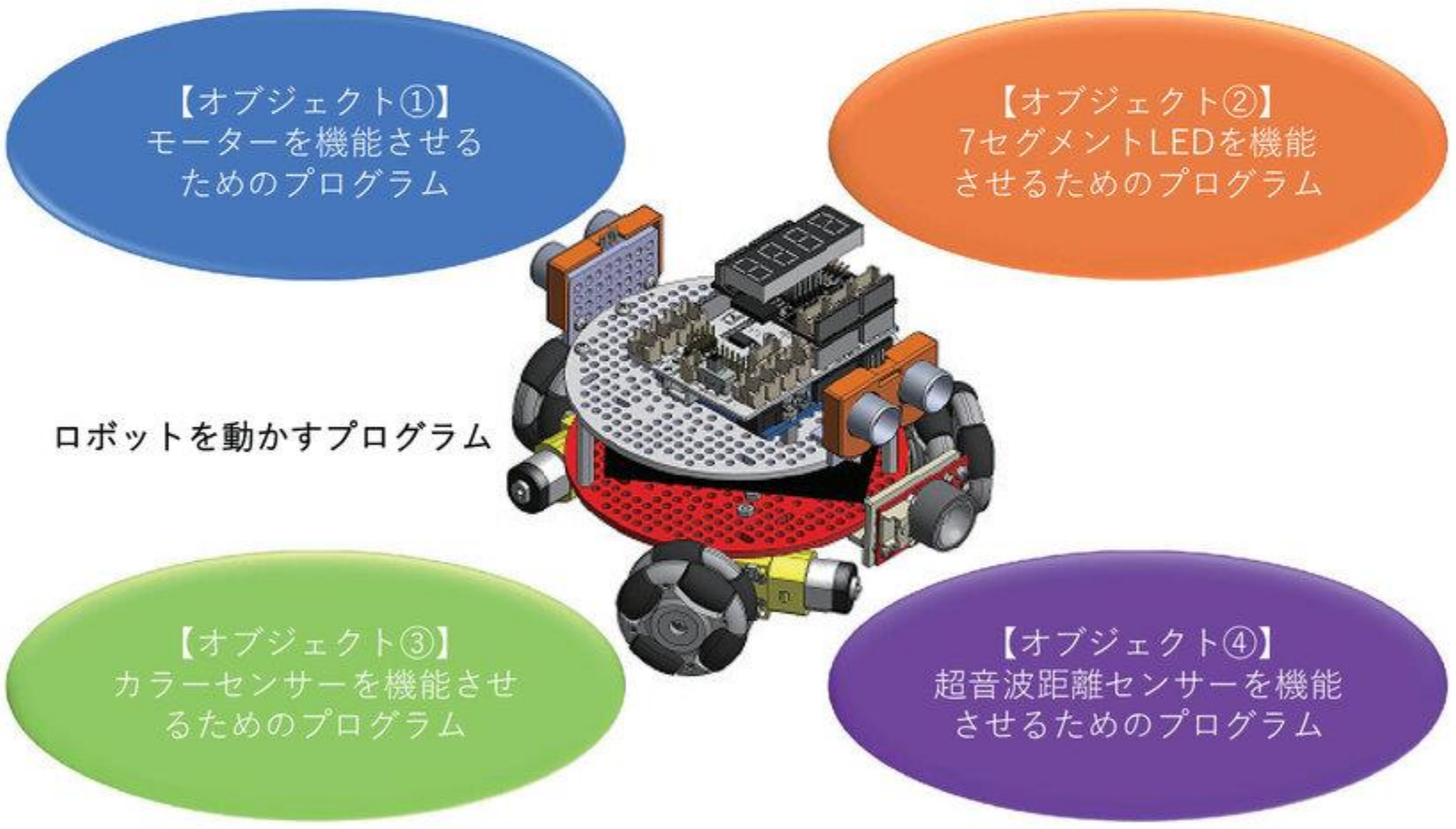


図1-0　オブジェクト指向イメージ（センサーロボットの例）

講　オブジェクトとはデータと処理の集まりのことです。漠然とした集まりではなく、ひとつのテーマ（センサーやモーターの制御）を持った集まりということをご説明ください。

コラム　「手続き型」と「オブジェクト指向」

もともとプログラミングというのは「手続き型」と呼ばれる方法が主流でした。
「手続き型」とは、記述された命令を上から順番に実行して、処理の結果に応じて変数を変化させていくプログラミング言語のことです。手続き型の考え方のメリットは「シンプルである」ことです。人間がソースコードを見たときにプログラムの処理の流れをつかみやすく、読み飛ばすなどのミスも起きにくいです。また、処理速度が速いというのもメリットです。反面、あらかじめ綿密にプログラムが実行される順番を決めておく必要があり、たくさんの機能をもたせる複雑なプログラムでは、のちのち変更や追記がしにくかったり、複数の人で作業を分業できないデメリットがあります。
一方、「オブジェクト指向」は、大規模なシステムのプログラムでは作業が分担できることや、エラーが出てもエラー箇所を見つけることや修正することがしやすいメリットがあります。このように、目的によってメリット、デメリットはありますがロボットのようにさまざまな機能を複合的に使用するものには、プログラムの再利用性の高いオブジェクト指向のメリットが特にありがたいのです。

1.1. 「オブジェクト指向」についてくわしく知る

さて、オブジェクト指向についてしっかり理解してもらうための解説をしていきます。

ロールプレイング･ゲームで､キャラクターを「魔法(まほう)使い」という職業（クラス）につかせ、使えるわざ（魔法(まほう)）を設定するという場面を考えてみましょう。

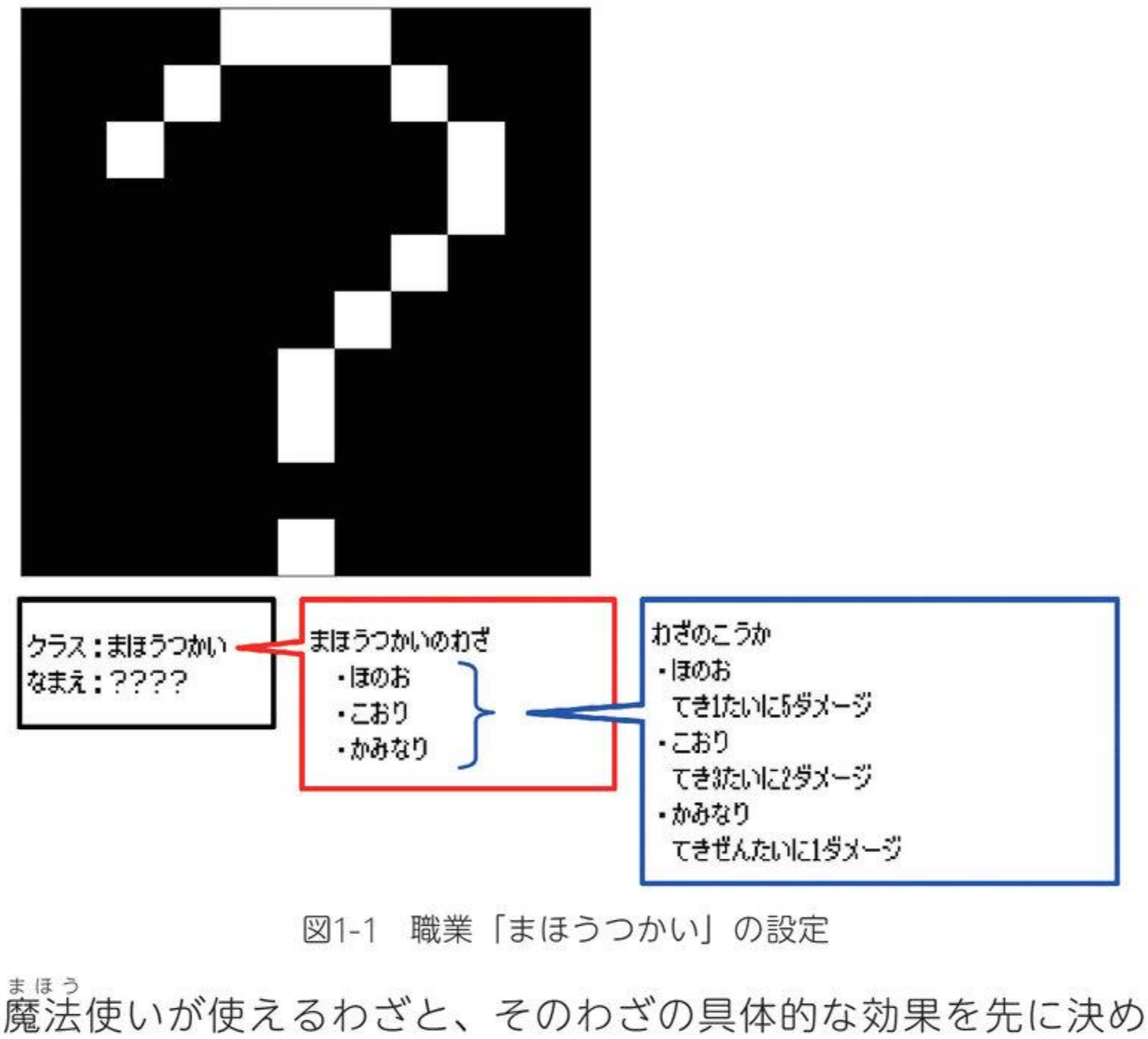


図1-1　職業「まほうつかい」の設定

このように、魔法(まほう)使いが使えるわざと、そのわざの具体的な効果を先に決めてしまえば、後が楽ですよね。

今後どのようなキャラクターが魔法(まほう)使いになったとしても、この設定を当てはめるだけで済みます。

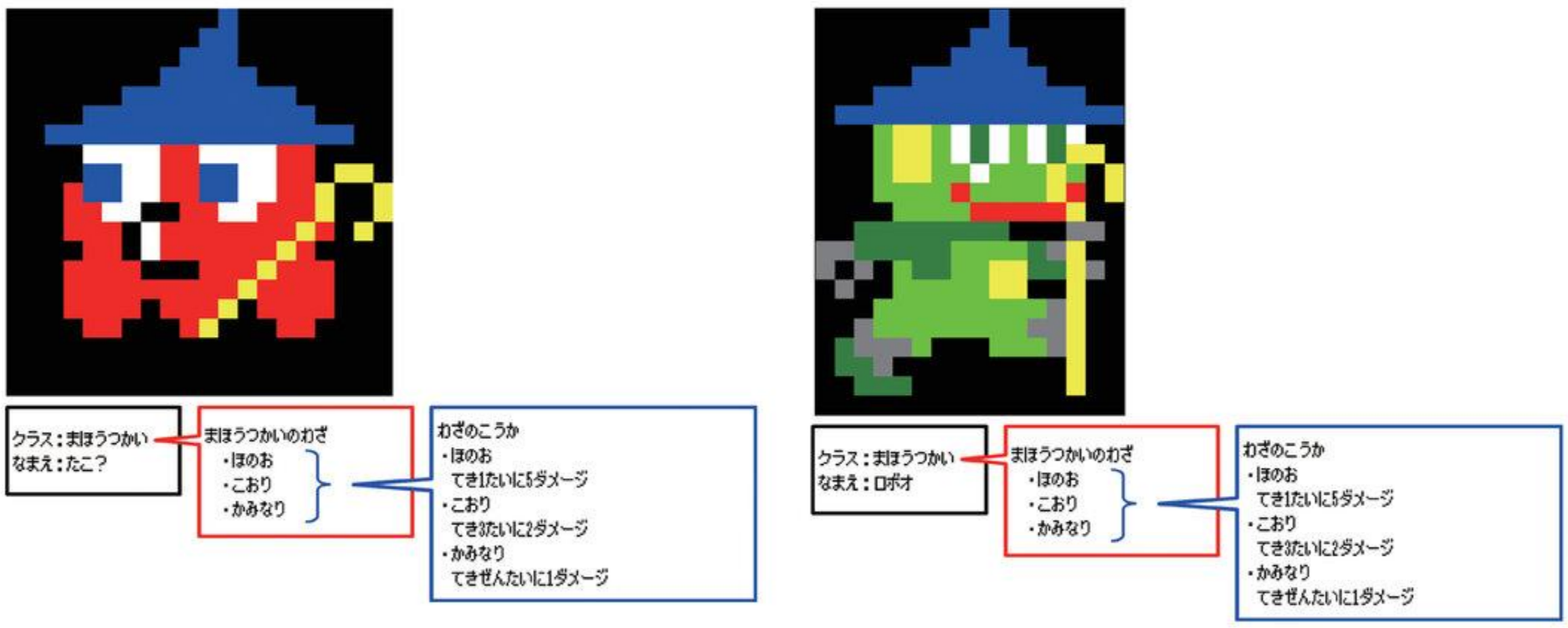


図1-2　誰が「まほうつかい」になってもわざは同じ

この考え方をプログラミングに応用したのが「オブジェクト指向」なのです。

ここで、ロボプロで扱(あつか)ってきたロボットを思い返してみましょう。

たとえば、オムニホイールロボットでは 3 つのギアドモーターを接続し、モーターを制御するために `mc1.rotate(100);` などといった命令をかいていました。`mc1` の部分は `mc0` や `mc2` などとかえていましたが、`rotate(100);` の部分はどのモーターを動かすときも同じように使えましたね。

モーターへの動作命令を出すプログラムは、必ずはじめにこのような宣言がありました。

プログラム「MotorTest」より抜粋

```
RPmotor mc1(MC1);          //MC1に繋がっているモーターを指定する
RPmotor mc2(MC2);          //MC2に繋がっているモーターを指定する
```

mc1やmc2の部分が各モーターにつけられた「なまえ」、つまり**図 1-2** でいえば「たこ？」や「ロボオ」の部分です。その前の部分でRPmotorとかかれていますが、これが**図 1-2** でいう「まほうつかい」に対応した部分です。

つまり、「『たこ？』と名付けたキャラクターは魔法使いの職業の設定を使いますよ」と宣言するように、「mc1やmc2と名付けたもの（オブジェクト）はRPmotorというクラスの設定を使いますよ」という宣言をしているのです。

魔法使いというクラスにはいくつかの「わざ」があり、それぞれのわざが特有の効果を持っていましたね。同様に、RPmotorというクラスにもいくつかの「わざ」、つまり関数があります。

実際に、RPmotorのクラスの設定がかかれている部分を見てみます。

ヘッダーファイル「RPlib.h」より抜粋

```
class RPmotor
{
  public:
      RPmotor(int PWMA, int AIN1, int AIN2);
      RPmotor(int ch);
    void rotate(int velo, int time);
      void cw(int velo, int time);
      void ccw(int velo, int time);

      void rotate(int velo);

       (中略)

};
```

これが、RPmotorというクラスがもっている「わざ」、つまり関数の一覧です。たとえばvoid rotate(int velo);とありますね。モーターをRPmotorクラスに指定すれば、rotate();という関数が使えるようになりますよ、ということです。

rotate();という関数が具体的にどのような動作でできているかも、ライブラリファイルにかかれています。確認してみましょう。

ライブラリファイル「RPlib.cpp」より抜粋

```
void RPmotor::rotate(int velo)
{
	if(velo>0){
	digitalWrite(_AIN1, HIGH);
	digitalWrite(_AIN2, LOW);
		SETPWM;
	}
	if(velo<=0){
	digitalWrite(_AIN1, LOW);
	digitalWrite(_AIN2, HIGH);
	velo=-1*velo;
		SETPWM;
	}
}
```

大まかに説明すれば、引数がプラスのときはモーターを正回転、マイナスのときは逆回転させるという命令です。ただ「モーターを回す」というだけの命令にも、実はこれだけの行数が必要だったのです。

このようなつくりにしておくと、どのように便利か考えてみましょう。
まず、どれだけモーターを接続しても、すべて同じ関数で制御できるのは便利ですね。
「他のキャラクターも魔法使いにしたい！」と思ったとき、いちいち新たにわざの設定をかかなくて済むのと同じです。

また、モーターに何度も動作命令を出す時はどうでしょうか。もし「オブジェクト指向」の考え方を使わないとしたら、モーターを回すたびに上にあったような命令をかかなければいけません。しかし、`RPmotor`というクラスをつくったことで、`rotate(100);`と一文かくだけでよくなりました。わざを使うたびに「ほのおを使います。ほのおの効果として、敵１体を選択し、その敵に５ダメージを与えてください」と命令していたところを、「わざの効果」を先に設定しておけば「ほのおを使います」と命令するだけであとは自動的に処理してくれるようなものです。

さらに言えば、これまでみなさんは`rotate()`関数の中身を具体的に知らなくても、適切に命令を出してたくみにロボットをあやつっていました。誰かがいったんクラスをつくってしまえば、他の人はその中身をいちいち読みとかなくても関数を使うことができるわけです。

現代のプログラミングには、「大規模なプログラムをグループで分担してかいていく」という場面がたびたびありますから、こういったオブジェクト指向の便利さというのが何かと合っているわけです。

図 1-2 の例とロボプロのプログラムがどのように対応しているのか、表にまとめます。

表1-0　オブジェクト指向における用語

図1-2の例	RPmotorの例	プログラミングの世界での呼び名
「魔法使い」という職業	RPmotorというクラス	クラス
「魔法使い」になった「たこ？」というキャラ	RPmotorクラスを割り当てられたmc1というモーター	インスタンス
各「わざ」の具体的な効果	各関数の具体的な動作	メソッド
何らかの職業になったキャラクター自体	何らかのクラスを割り当てられたもの自体	オブジェクト

これまで何度も触れてきたRPmotorクラスにも、まだ使われたことのない関数がたくさんあります。
機会があればライブラリファイルを読んでみると、新たな発見があるかもしれませんよ！

2．ゲームプログラムの動作を確認する（目安60分）

2.0.　ゲームを実行してみる

いきなり「ゲームをつくりましょう！」と言われても困ってしまうと思います。まずは、すでに完成しているゲームを実際に実行してみましょう。これをなるべく自分で再現できるようにしたいので、具体的にどのような動作をしているかよく見ておいてください。

なお、A0、A1、A4、A5にタクトスイッチを接続します。配線は下の図を参考にしてください。

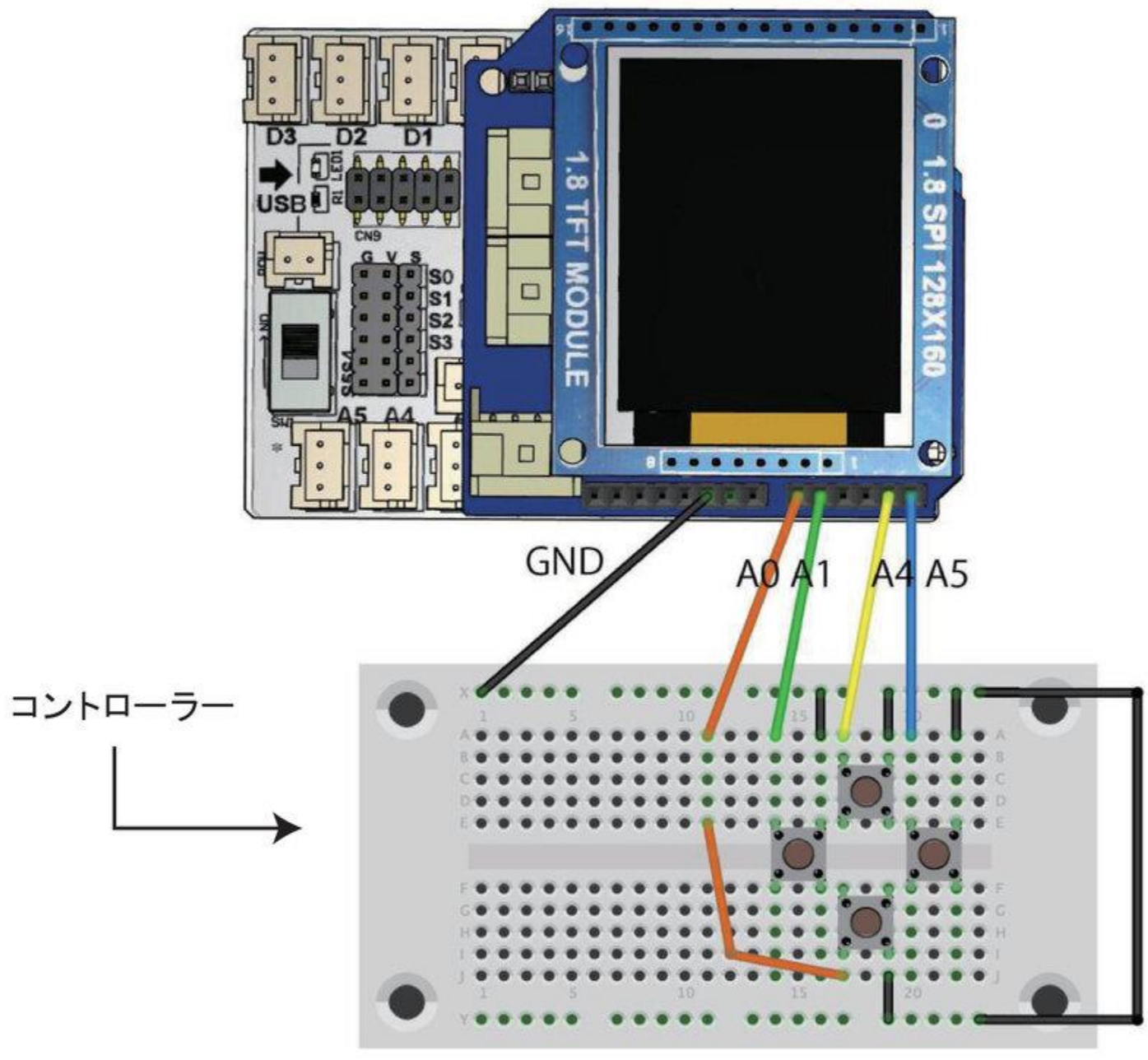


図2-0　タクトスイッチ接続の回路

続いて、プログラムを実行してみましょう。

プログラムの書き込み

RoboticsProfessorCourse3 > MagicItemLCD6 > Snake

POINT

タイトル画面では上下スイッチで難易度を選び、右スイッチを押したらゲーム開始です！

いかがでしょうか。
「ヘビゲーム」などとよばれる、「レトロゲーム」のなかでは非常に有名なものの一つです。
色々な人にアレンジされていてオリジナル・ルールが追加されることも多いですが、今回のルールは以下の通りです。

POINT

① ヘビを上下左右に動かし、画面上のアイテム（青りんご）をなるべく多く取らせることが目的。
② ヘビは４つのボタンで操作できるが、ボタンが押されていないときは直前と同じ向きに進む。
③ ヘビはりんごを食べるたびに、尻尾が１マスずつ伸びていく。
④ たまにスペシャルアイテム（赤りんご）が出現する。取ると大量にポイントが貰(もら)えるうえに、ヘビの尻尾が少し縮む。
⑤ ヘビの頭が画面端(はし)の壁(かべ)、またはヘビの胴体(どうたい)にぶつかったらゲームオーバー。

ルール自体は非常にシンプルですが、不思議と遊んでしまいますよね。
このゲームプログラムは、関数の機能を駆使(くし)してメインループを非常に短くまとめているのが特徴(とくちょう)です。

プログラム「Snake」より抜粋

```
//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);
}
```

複雑な動作の割には、void loop()内の行数はかなり少なく済んでいますね。黄色の部分がこのプログラム内で宣言されている関数です。それぞれの関数に、このゲームに必要な各種機能がまとまっています。
関数名を見てみると、それぞれどのような機能かなんとなくわかるのではないでしょうか？

関数の中身を確認してもいいのですが、複雑な処理が多くいきなり見てもわからない部分があちこちにあると思います。
そこでまず、自分で「ヘビゲーム」の動作をある程度再現してみることで、プログラムのつくりを大まかにつかむところから始めましょう。

2.1. 動く「点」をつくる

今回は、ヘビの移動をこれまでに学んできた構文で再現してみましょう。

やってみよう！

ヘビの形をつくる前に、まずはタクトスイッチを押すとただの「点」が移動するプログラムをつくってみよう。
第 3 回で使ったプログラム「Line1」をベースにしてみてね。

ヒント

「Line1」では「指定した座標に点をえがく」という構文を使ったね。
`TFTscreen.point([x 座標 ], [y 座標 ]);`だよ。
また、タクトスイッチでキャラクターを動かすプログラムは、第4回でも「Chara5A」などで触れたね！
タクトスイッチのピン宣言（せんげん）など、わからなくなったらこれらのプログラムを参考にしてみよう！

講　解答例は巻末に記載します。

できましたか？
今回のように点を動かすだけなら、必要なのは以下のような処理（しょり）だけですね。

POINT

① 4つのタクトスイッチを`INPUT_PULLUP`で接続する。
② 点をえがくx座標、y座標を入れておくため、変数を2つ宣言（せんげん）する（仮に`px`、`py`とする）。
③ 左右スイッチ、上下スイッチが押されたら、`px`か`py`を1ずつ増減させる。
④ `px`や`py`を増減し終えたら、`point();`命令を使って(px, py)に点をえがく。
⑤ しばらく（0.1秒程度）点を表示したら、今度は黒い点を同じ座標にえがく（次の点をえがいたとき、前の点が残っているのを防ぐため）。

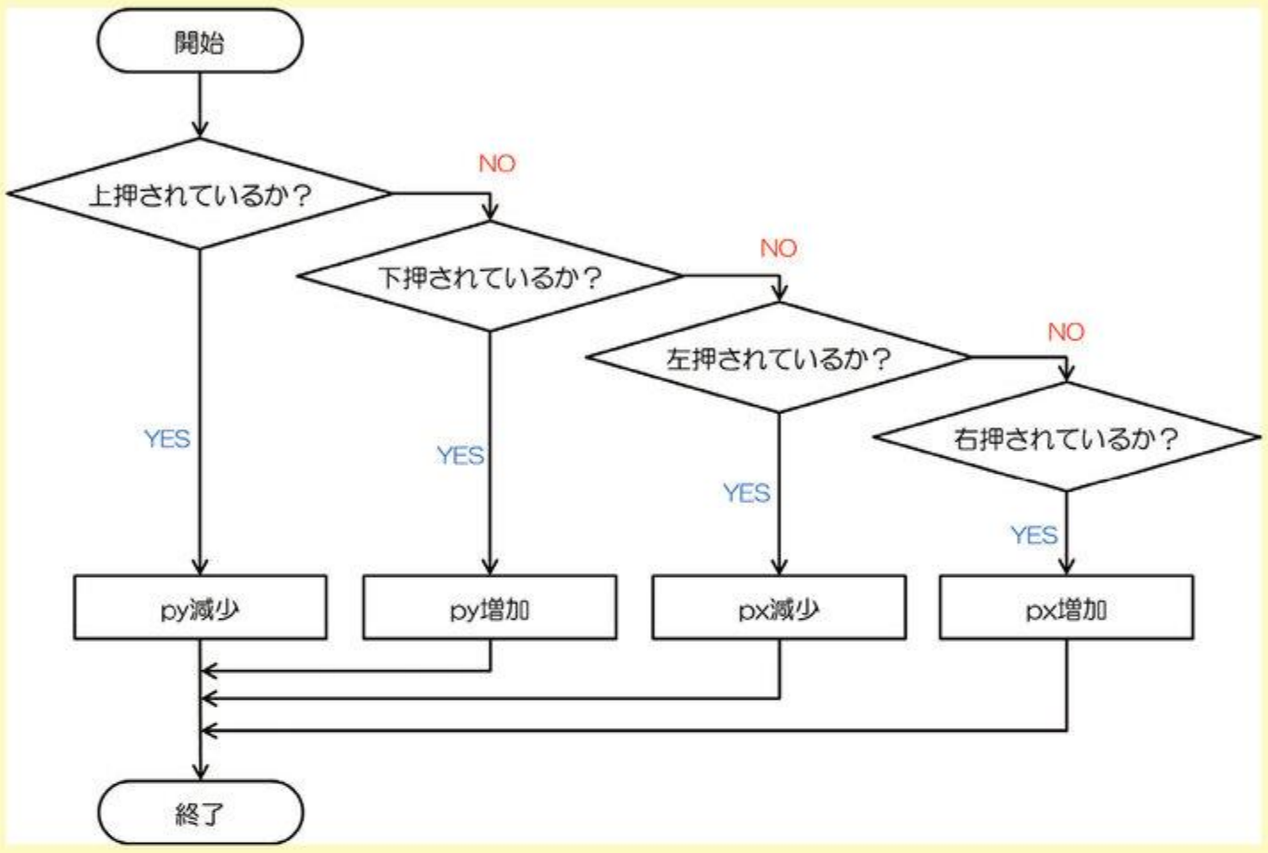
ここから、ヘビゲームの動作を再現していくため、いくつか機能を追加していきましょう。

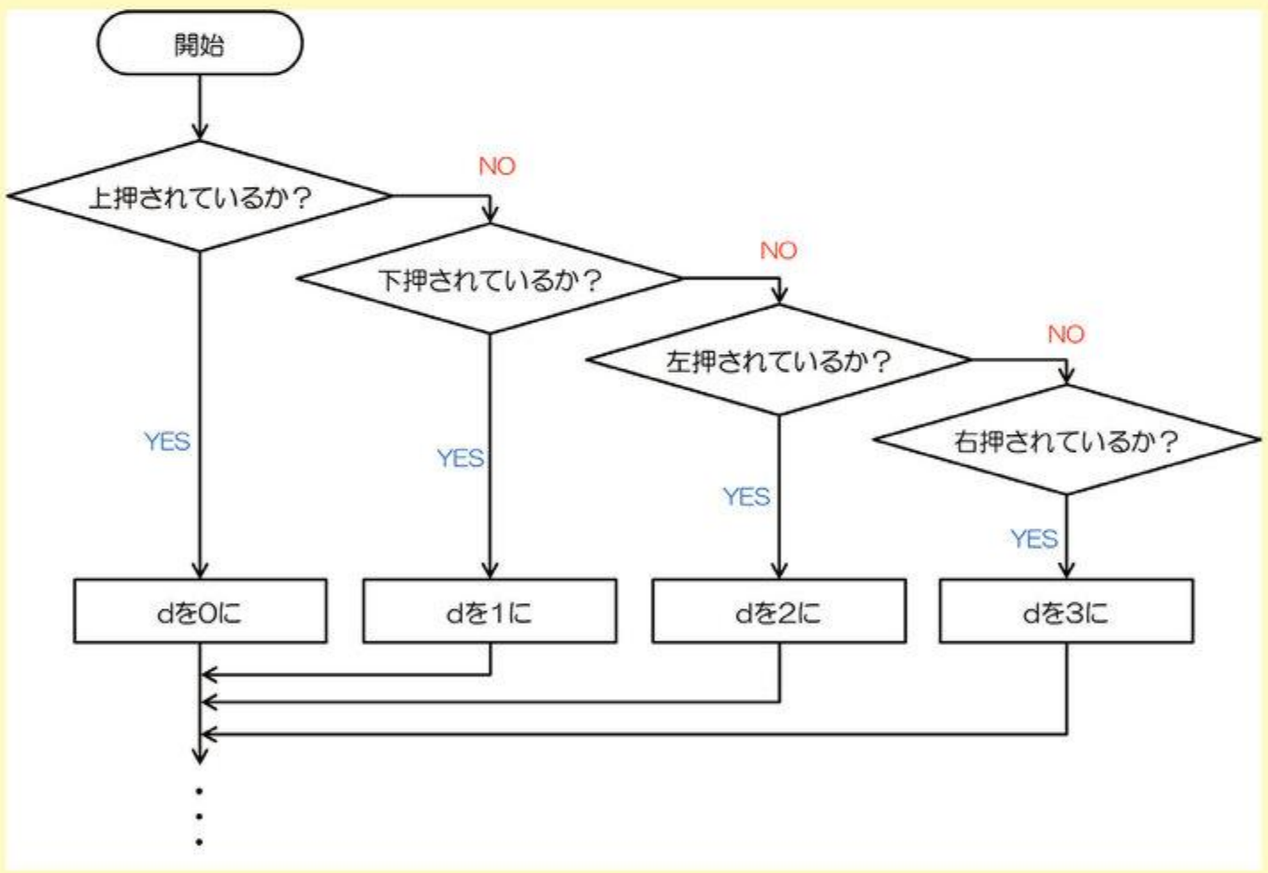
やってみよう！

先ほどつくったプログラムを更にかきかえ、以下のような動作を加えてみよう！
① ボタンが押されていないときは、直前と同じ向きに移動し続ける。
② 上下左右の画面端（はし）に各2マスずつ程度の「壁（かべ）」を設定し、それ以上外側へは移動できないようにする。

ヒント

①も②も基本的にはif文をうまく追加していくだけでできるから、じっくり考えてみよう。





①は上図のようなプログラムだと、スイッチが押されている間だけ点が移動することになるよね。
ここで適当な変数（今回はdとするよ）を宣言（せんげん）して、下図のように直してみるとどうなるかな？

講　解答例は巻末に記載します。
なお、「ヒント」のフローチャート通りに変更すると、他のスイッチが押されるまでｄの値が変更されずにそのまま残るので、スイッチが押されていないとき直前の動きを続けるようになります。

2.2. 動く「点」を「ヘビ」にする

続いて、「点」をえがいていただけのプログラムを少し「ヘビ」に近づけてみましょう！

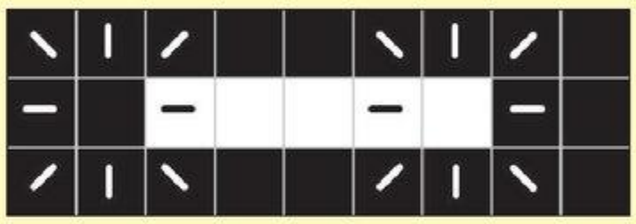
ステップアップ

プログラムを更にかきかえ、以下のような動作にしてみよう！

POINT

・「点」ではなく、全長5マス分の「ヘビ」をえがく。
・移動はこれまでと同じように上下左右のスイッチで行う。操作するのはヘビの頭（先端の1マス）。ヘビの体（頭以外の4マス）は、頭が通ったルートをそのまま追いかける。

ヒント



これまではえがいたマスをすぐ消していたけど、5つ前のマスを消すようにかえることでヘビになるよね。
この「5つ前のマスを消す」という処理が難しいんだ。前にえがいたマスの位置も、どうにかして記憶しておかなければならないんだね。これまでに学んできた機能を色々と思い出してみよう。

さて、ヘビをえがくことはできましたか？
方法はたくさんありますが、今回はこれまでにも何度か扱ってきた「配列」を使うとうまくいきますね。

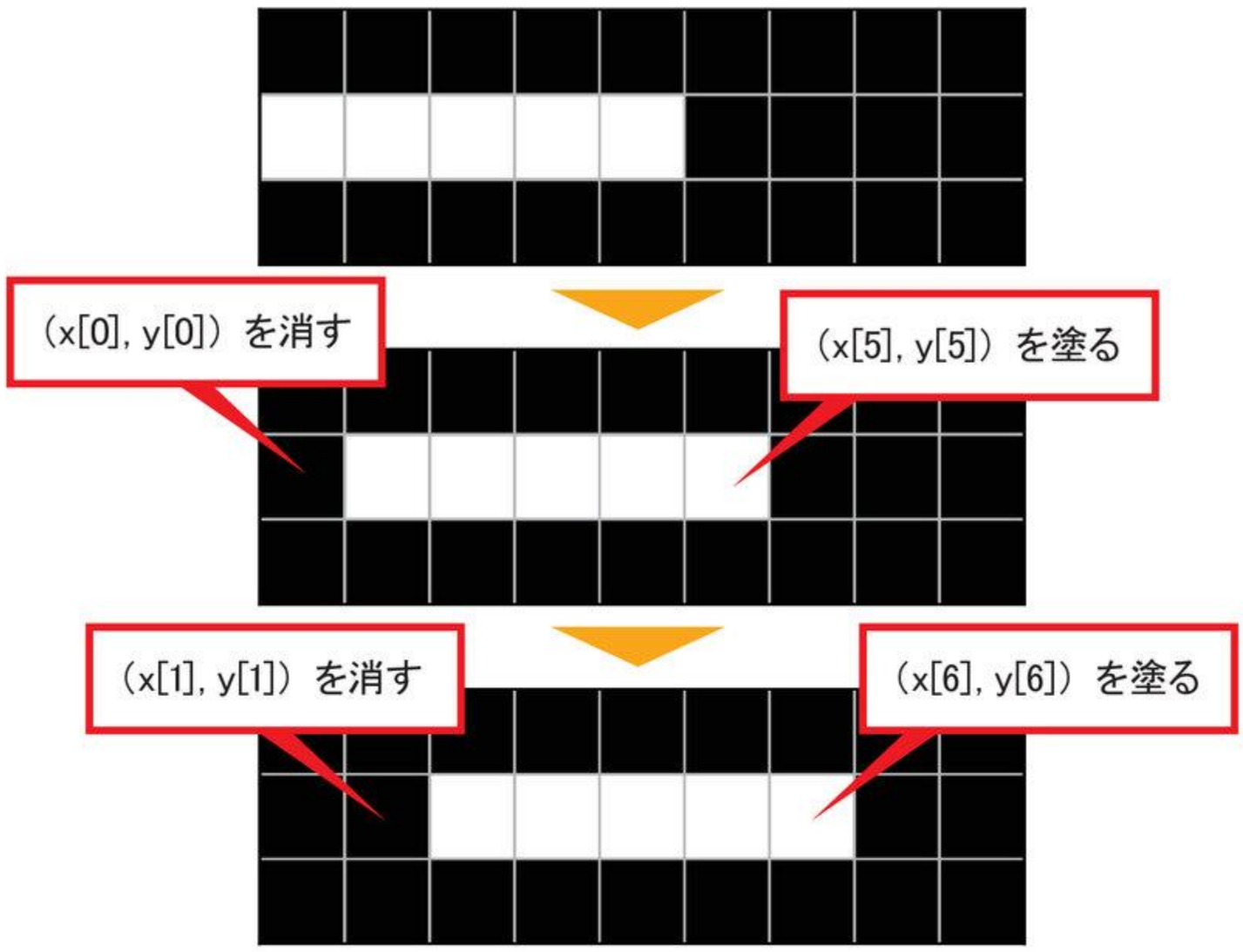


図2-1　配列で座標を管理する

1ループ毎に変数 px 、 py の値を x[i] 、 y[i] などという配列に記録すれば、上の図のように「新たなマスを塗る」「5つ前のマスを消す」という2つの処理が同時に行えるようになります。

前のページでわからなかったという人は、このヒントを参考にもう一度チャレンジしてみましょう！

講　解答例は巻末に記載します。

これで、基本的なヘビの動きは完成しましたね。
あとは、自分のできる限りで構わないので、必要な機能をどんどん実装してみましょう。

チャレンジ課題

ヘビの動きのプログラムにかき足していく形で、プログラム「Snake」の動作をなるべく再現できるよう機能を追加していきましょう。たとえば以下のような機能があるとそれらしくなるね。

POINT

・ヘビの体1マスを1×1の「点」ではなく、3×3の「正方形」で表示する。
・ヘビがいずれかの向きに移動しているとき、正反対にはターンできないようにする（たとえば右に移動しているときに左ボタンを押しても、移動方向が変化しないようにする）。
・ヘビの頭が画面端（はし）に当たってしまったらゲームオーバーになる（プログラム「Snake」では、ヘビの動きを止めるのと同時に頭のマスを赤く塗（ぬ）っているね）。

「Snake」にあった機能以外でも、「こういう動きができればゲームとして面白いかもしれないな」というアイデアがあればプログラムに反映させてみるのも面白いね！

講

ヘビ1マスを3×3にする場合、pxおよびpyが1変化するごとに描画位置を3マスずらす必要があります（または、ボタン操作1回ごとにpxやpyが3ずつ変化するようにかきかえても構いません）。
ターンできないようにする処理は第6回プログラム「Snake」の関数 updateSnakePosition() 内にもかかれていますので、こちらを参照させるのもよいでしょう。この関数が今回製作したヘビの移動を司る部分です。

3．液晶ディスプレイを活用する（目安30分）

3.0. プロットでグラフをかく

第3回から今回にかけて、point(x, y);という命令が何度も登場しました。基本的には液晶ディスプレイにキャラクターを表示させるときなどに使っていましたが、1つずつ点を打つ（プロットする）という処理にはほかにも使い道があります。たとえば、数学のグラフをえがくのにも、point文が役立ちます。

皆さんが学校で算数や数学を学ぶとき、グラフは「線」でえがくのが基本だったと思います。「比例」や「一次関数」は直線のグラフ、「反比例」や「二次関数」は曲線のグラフでした。もっと先の学年では「三次関数」「三角関数」「指数関数」などなど…さらに多くの関数を学ぶ事になりますが、やはりどれも基本的には「線」でグラフをえがくことになります。

しかし、もっと正確にグラフというものを見てみましょう。たとえば、y = 4 / x、つまり反比例のグラフを考えてみます。
普段であれば「x = 2のときy = 2」のようなキリの良い点だけいくつかプロットして、点どうしをそれらしい曲線で結ぶ、というえがき方をするかもしれませんが、このグラフはもっと拡大してみると「xが1.999999999999のとき」や「xが2.000000000001のとき」の点も存在するはずです。つまり、教科書に載っている「反比例のグラフ」というのは、より厳密に言えば「無数の点が連なって線のように見えているもの」ということになりますね。

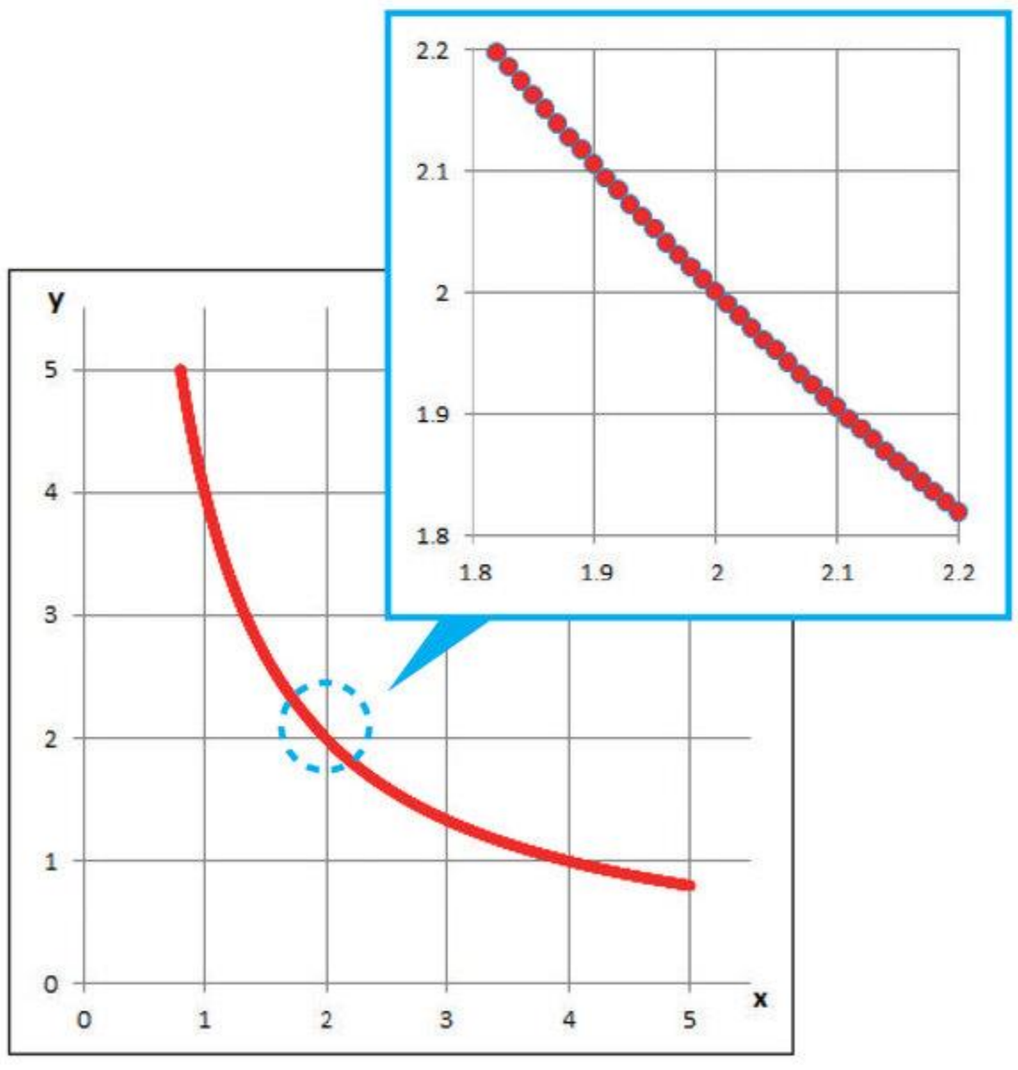


図3-0　グラフは「点の集合」

「そんなにたくさんの点をいちいち手がきしていられないよ！」という事で普段は線でつなぐだけで済ませているのですが、point文さえあればこの作業をマイコンボードに行わせることができますよね。

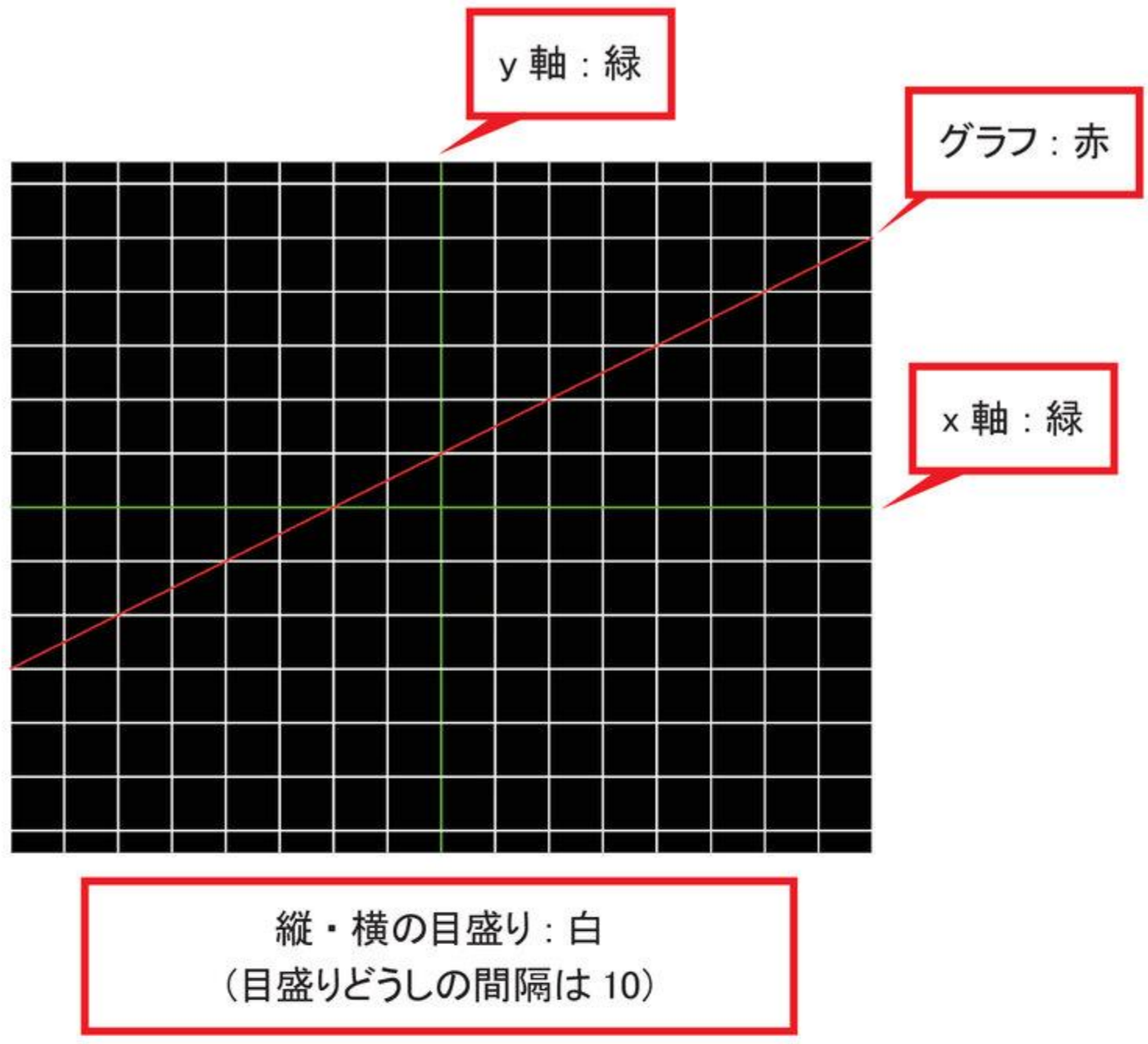


図3-1　液晶ディスプレイにグラフをかく

まずは上の図のような見た目で軸や目盛りを表示させておけるよう、プログラムの下地をつくっておきましょう。

プログラムの書き込み

RoboticsProfessorCourse3 > MagicItemLCD5 > Graph_cline

実行結果：**図3-2**のような画面が表示されます。
このプログラムに手を加えてグラフを完成させてみましょう。

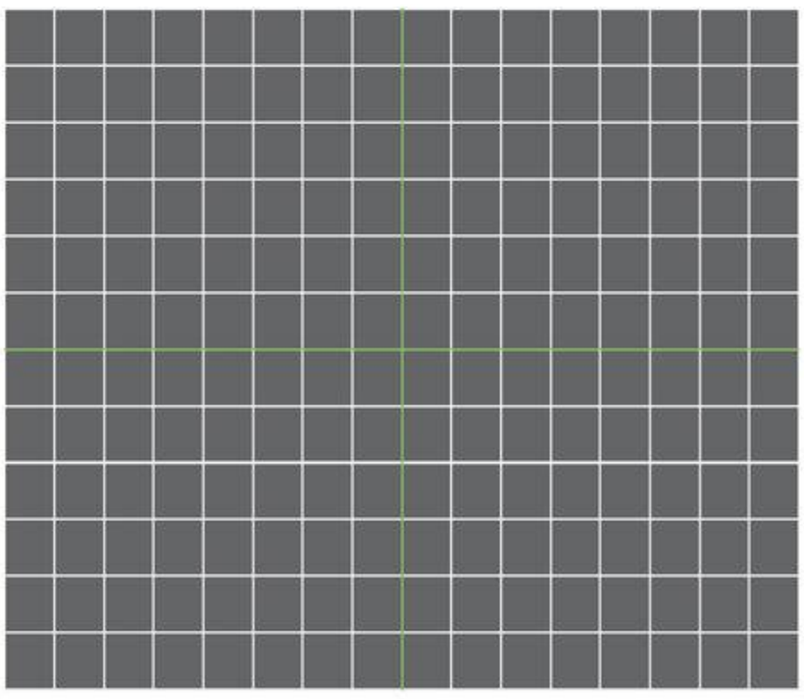
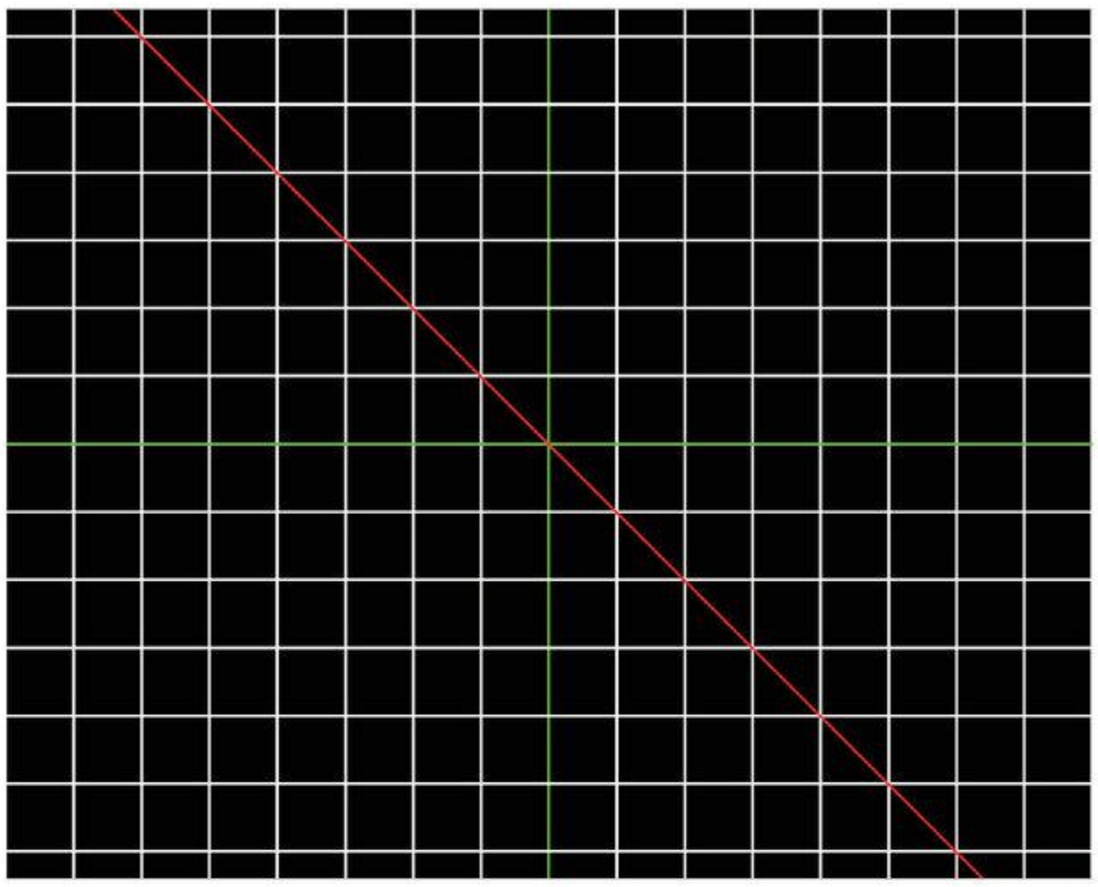


図3-2　センターライン

やってみよう！

プログラム「Graph_cline」をかきかえ、下の図のようにグラフを表示させてみよう！
算数・数学で扱うグラフとはy軸の向きが逆なので紛(まぎ)らわしいけど、「y = x」のグラフだよ！



ヒント

for文を使って、「xが●のときのyの値」を求めてその座標に赤い点を打つ、という処理(しょり)を、x = -80からx = 80までくり返せばいいんだね！
ちなみに、今回は最初にグラフをえがいてしまえばいいだけだから命令文は void setup() 内にかくといいよ！

y = xは比例の関係なので直線のグラフになるはずですが、えがけたでしょうか？
以下のような処理(しょり)を追加すればよいですね。

```
TFTscreen.stroke(0, 0, 255);


int x, y;
for(x = -80; x <= 80; x++){
    y = x;
    TFTscreen.point(x + 80, y + 64);
}
```

`y = x;`の部分で、プロットする点のy座標を求めていますね。今後、違うグラフをえがきたいときはこの式をかきかえましょう。
ところで、「グラフは一応えがけたけれど、点線になってしまう」と悩んでいる人も多いのではないでしょうか。実際に「x += 3」などとすると点線が目立ちます。このままだと、どうしても点と点の間隔が広くなってしまいますね。そこでこのように改良してみましょう。

```
TFTscreen.stroke(0, 0, 255);

float x, y;
for(x = -80; x <= 80; x += 0.01){
    y = x;
    TFTscreen.point(x + 80, y + 64);
}
```

変数`x`、`y`をfloat型にすることで、小数点以下を含む値を指定できるようになりました。ディスプレイの性能にもよりますが、これなら、点の間隔を理論上は狭められます。
これまでにも何度か説明はありましたが、`int`は「整数型」といい、-32,768から+32,767までの整数しか扱えません。もし小数を扱いたければ浮動小数点型である`float`を、より大きい数字を扱いたいときは長整数型である`long`を使いましょう。

豆知識

変数を`long`で使用すると、-2,147,483,648 から +2,147,483,647 までの数を扱えます。ただし`int`と同様、整数しか扱えません。
とはいえ、今回のグラフは x, y 座標とも`int`の範囲で十分収まりきるので、`long`の出番はありませんね。

あとは自由に式をかきかえ、色々なグラフの形を見てみましょう。
例として、いくつかグラフのサンプルを用意しました。興味のある人は書き込んでみてください。

プログラムの書き込み

RoboticsProfessorCourse3 > MagicItemLCD5 > Graph_xx

実行結果：$y=x^2$のグラフが表示される。
まだ習っていない学年の人もいるかもしれませんが、中学校の数学で学ぶグラフです。

これ以降のグラフは、全て高校数学レベルの数式になっています。
「習っていないからどんなグラフになるかわからないよ！」というときでも、数式を入力すれば自動的にグラフをかいてくれるのでとても便利ですね！

表3-0　グラフのサンプルプログラム

プログラム名	グラフの式
Graph1	$y = \frac{(10-x)(x+20)}{10}$
Graph2	$y = \frac{(10-x)^2}{100}$
Graph3	$y = \frac{(x-10)^2}{1000}$
Graph4	$y = \frac{x(x+20)(x-30)}{1000}$

さらに、高校数学で学ぶ「三角関数」を駆使したグラフに、以下のようなものもあります。

プログラムの書き込み

RoboticsProfessorCourse3　＞　MagicItemLCD5　＞　GraphPole

①～⑥の //（コメントアウト）の位置を順番に変更していきましょう。

```
/********* コメントアウトを変更しましょう！ *********/
    //楕円
      x=MAG*cos(p);
      y=MAG/2*sin(p);

    //心臓型　ケルスマのカーディオイド
①   //x=20*(2*cos(p)-cos(2*p));
    //y=20*(2*sin(p)-sin(2*p));

    //三葉線
②   //x=MAG*sin(3*p)*cos(p);
    //y=MAG*sin(3*p)*sin(p);

    //四葉線
③   //x=MAG*sin(2*p)*cos(p);
    //y=MAG*sin(2*p)*sin(p);

    //八葉線
④   //x=MAG*sin(4*p)*cos(p);
    //y=MAG*sin(4*p)*sin(p);


    //アルキメデス曲線
⑤   //x=MAG/(p*5)*cos(p*5);
    //y=MAG/(p*5)*sin(p*5);

    //星芒形
⑥   //x=MAG*cos(p)*cos(p)*cos(p);
    //y=MAG*sin(p)*sin(p)*sin(p);
/*************ここまで**************************/
```

①を実行し、表示を確認したら
①のx・yに再び//（コメントアウト）
を入れて②の//（コメントアウト）
を削除してください。

同じ手順で②～⑥も進めてください。

3.1. 三次元表示にチャレンジ

二次元はかくことができました！ さらに三次元に拡張します！
本来、三次元表示とは非常に大きい計算処理とメモリー容量が必要になるため、パソコンを使わないと難しいのですが、我々のマイコンくんにも頑張ってもらいましょう。

ここではタッチセンサーを使います。D3 にタッチセンサーを接続してください。

まずは実行してみましょう！ プログラムは以下になります。

プログラムの書き込み

RoboticsProfessorCourse3 ＞ MagicItemLCD5 ＞ tiny_3d_MPU

実行結果：三次元のものが表示され、D3 に接続したタッチセンサーを押すと、表示が変わる。

やってみよう！

コメントアウトの位置をかえてみよう。
コメントアウトの位置をかえることによって表示されるものも変わるよ！

プログラム「tiny_3d_MPU」より抜粋（93 ～ 103行目）

```
// ----------------------------------------
// meshes   変更してみよう
// ----------------------------------------
/*****************************************/
// uncomment 1 header to automatically load mesh
#include "mesh_cube.h"
//#include "mesh_cone.h"
//#include "mesh_sphere.h"
//#include "mesh_torus.h"
//#include "mesh_monkey.h"
/*****************************************/
```

実行が終わったものは、再び //（コメントアウト）してください。

さて、三次元表示ができましたが、せっかくなので姿勢検出シールドの機能を使ってみましょう。まずは姿勢検出シールドのジャイロ機能をテストしてみましょう。
以下のプログラムを実行してください。

プログラムの書き込み

RoboticsProfessorCourse3 ＞ MagicItemLCD5 ＞ MPU9250DMP_TFT

実行結果：液晶画面上にセンサーの数字が表示される。
「センサーを動かしたときに各値が変化するかどうか確認してみましょう。「倒立振子ロボット」で使った機能ですね。

やってみよう！

マイコンユニットを動かして、三次元表示を動かしてみよう。プログラム「tiny_3d_MPU」を実行して以下のようにコメントアウトを外して実行してみよう。

プログラム「tiny_3d_MPU」に姿勢検出シールドの機能を加えます。黄色部分のコメントアウトを外して実行してみましょう。プログラム実行後、タッチセンサーを押してください。

プログラムの書き込み

RoboticsProfessorCourse3 ＞ MagicItemLCD5 ＞ tiny_3d_MPU

プログラム「tiny_3d_MPU」より抜粋（87 ～ 92 行目）

```
// ----------------------------------------
// mode 変更してみよう
// ----------------------------------------
/****************************************/
//#define  USE_MPU9250
/****************************************/
```

実行結果：姿勢検出シールドの機能が加わり、マイコンユニットを傾けるとグラフィックが動く。

さてどうでしょうか？ 動かした方向に画像が動くだけでも不思議な感覚になったかと思います。動かした方向の部位が見えるようになっています。くるくると回して確認してみてください。

やってみよう！

いろいろな物体を回してみよう。プログラム「tiny_3d_MPU」のコメントアウトの位置を変更して、他の図形も動かしてみよう。

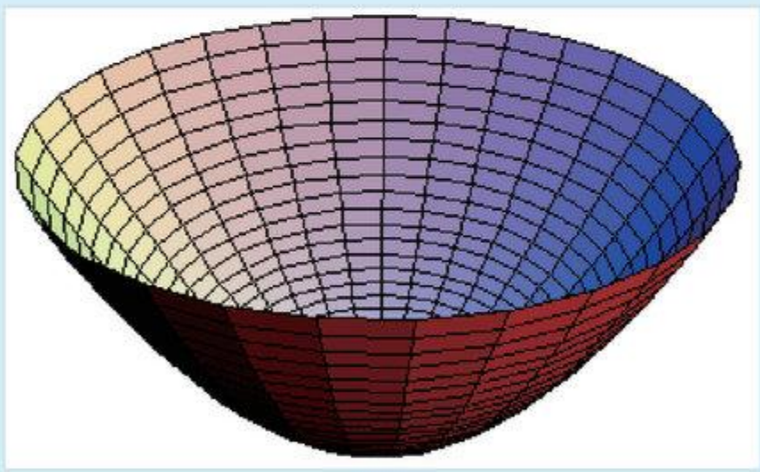
```
// ----------------------------------------
// meshes   変更してみよう
// ----------------------------------------
/**************************************/
//  uncomment 1 header to automatically load mesh
#include "mesh_cube.h"
//#include "mesh_cone.h"
//#include "mesh_sphere.h"
//#include "mesh_torus.h"
//#include "mesh_monkey.h"
/**************************************/
```

コラム　ポリゴンとは？

ポリゴンとは、多角形という意味の言葉ですが、三次元コンピュータグラフィックスにおいては、曲面などの立体物を細かい三角、四角等の多角形に分割し、擬似(ぎじ)表示する方法を指します。

一次元情報のデジタル化と似ていると思います。環境の情報をアナログセンサのA/D変換という方法によってマイコン内に取り込(こ)んだ時と同じイメージです。実際の三次元空間の物体をコンピュータ上で表現したり、取り込(こ)んだりするときに非常に重宝します。

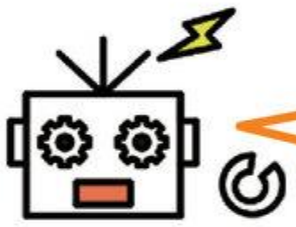
この多面体の数が多ければよりリアルな物体に見えますが、その分、データ量は増えて、処理(しょり)は重くなります。最近のゲームでは非常に多くのデータを取り扱えるので、遊んでいる人はほとんど気がつくことはないと思いますが、細かく見ていくと必ず、このポリゴンを使用して3DCGのキャラクターはかかれています。



“ポリゴン”『ウィキペディア (Wikipedia): フリー百科事典』
更新日時：(2017年11月11日)　URL: http://ja.wikipedia.org/wiki/ポリゴン

4. まとめ（目安５分）

今回は「オブジェクト指向」という言葉を学びました。
聞きなれない言葉ですが、実際にプログラムをかくときに、クラス、メソッドの使い方を理解していればよいと思います。
さらにグラフィックスを深く学んでみました。数学の使い方を含めていろいろなことが体験できたかと思います。
最後に三次元に関して体験をしてみました。三次元グラフィックスを取り扱うにはどうしても「高校レベルの数学」の知識が必要になりますので、勉強した後にまた振り返っておくとよいでしょう。
次回は、ゲームに関してもう少しくわしく学んでいきましょう！



次回は、不思議なグラフィックやゲームをさらに体験シヨウ！

講

〇以下の理解度を確認してください。
・オブジェクト指向プログラミングを理解する
・ゲームプログラムを再現してみる
・グラフや三次元グラフィックを表示する
〇次回は、「ゲームプログラムを読み解こう」を行います。

《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。

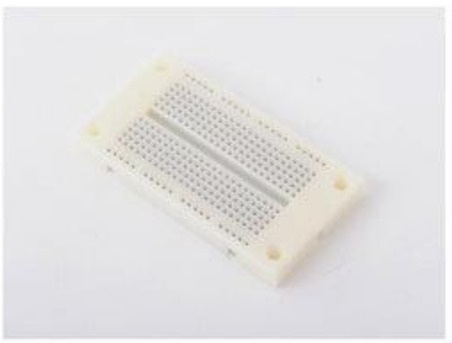
USB ケーブル	1	マイコンボード	1	ロボプロシールド	1	301 ブレッドボード	1
							
ジャンパー線	65	タクトスイッチ	10	姿勢検出シールド	1	液晶ディスプレイシールド	1
							

図4-0　次回必要なもの

P.12　やってみよう！　解答例

```
#include <TFT.h>
#include <SPI.h>

TFT TFTscreen = TFT(A2, 1, 9);

void setup(void){
    TFTscreen.begin();
    TFTscreen.background(0, 0, 0);
    pinMode(A0, INPUT_PULLUP);
    pinMode(A1, INPUT_PULLUP);
    pinMode(A4, INPUT_PULLUP);
    pinMode(A5, INPUT_PULLUP);
}

int px = 80;
int py = 64;
void loop(){
    if(digitalRead(A0) == LOW){
        py++;
    }
    else if(digitalRead(A1) == LOW){
        px--;
    }
    else if(digitalRead(A4) == LOW){
        py--;
    }
    else if(digitalRead(A5) == LOW){
        px++;
    }


    TFTscreen.stroke(255, 255, 255);
    TFTscreen.point(px, py);
    delay(100);
    TFTscreen.stroke(0, 0, 0);
    TFTscreen.point(px, py);
}
```

P.13　やってみよう！①　解答例

```
#include <TFT.h>
#include <SPI.h>
#define up 0
#define down 1
#define left 2
#define right 3

TFT TFTscreen = TFT(A2, 1, 9);

void setup(void){
    TFTscreen.begin();
    TFTscreen.background(0, 0, 0);
    pinMode(A0, INPUT_PULLUP);
    pinMode(A1, INPUT_PULLUP);
    pinMode(A4, INPUT_PULLUP);
    pinMode(A5, INPUT_PULLUP);
}

int d;
int px = 80;
int py = 64;
void loop(){
    if(digitalRead(A0) == LOW){
        d = down;
    }
    else if(digitalRead(A1) == LOW){
        d = left;
    }
    else if(digitalRead(A4) == LOW){
        d = up;
    }
    else if(digitalRead(A5) == LOW){
        d = right;
    }

    if(d == up){
        py--;
    }
    else if(d == down){
        py++;
    }
```

```
    else if(d == left){
        px--;
    }
    else if(d == right){
        px++;
    }

    TFTscreen.stroke(255, 255, 255);
    TFTscreen.point(px, py);
    delay(100);
    TFTscreen.stroke(0, 0, 0);
    TFTscreen.point(px, py);
}
```

P.13　やってみよう！②　解答例

```
#include <TFT.h>
#include <SPI.h>
#define up 0
#define down 1
#define left 2
#define right 3

TFT TFTscreen = TFT(A2, 1, 9);

void setup(void){
    TFTscreen.begin();
    TFTscreen.background(0, 0, 0);
    pinMode(A0, INPUT_PULLUP);
    pinMode(A1, INPUT_PULLUP);
    pinMode(A4, INPUT_PULLUP);
    pinMode(A5, INPUT_PULLUP);
}

int d;
int px = 80;
int py = 64;
void loop(){
    if(digitalRead(A0) == LOW){
        d = down;
    }
    else if(digitalRead(A1) == LOW){
        d = left;
    }
    else if(digitalRead(A4) == LOW){
        d = up;
    }
    else if(digitalRead(A5) == LOW){
        d = right;
    }

    if(d == up){
        py--;
    }
    else if(d == down){
        py++;
    }
```

```
    else if(d == left){
        px--;
    }
    else if(d == right){
        px++;
    }

    if(px < 2){
        px = 2;
    }
    if(px > 158){
        px = 158;
    }
    if(py < 2){
        py = 2;
    }
    if(py > 126){
        py = 126;
    }

    TFTscreen.stroke(255, 255, 255);
    TFTscreen.point(px, py);
    delay(100);
    TFTscreen.stroke(0, 0, 0);
    TFTscreen.point(px, py);
}
```

P.14　ステップアップ　解答例

```
#include <TFT.h>
#include <SPI.h>
#define up 0
#define down 1
#define left 2
#define right 3

TFT TFTscreen = TFT(A2, 1, 9);          //液晶ディスプレイを使うときのオマジナイ

void setup(void){
    TFTscreen.begin();                  //液晶ディスプレイ表示
    TFTscreen.background(0, 0, 0);      //背景色は黒
    pinMode(A0, INPUT_PULLUP);
    pinMode(A1, INPUT_PULLUP);
    pinMode(A4, INPUT_PULLUP);
    pinMode(A5, INPUT_PULLUP);
}

int x[300], y[300], i, d;
int px = 80;
int py = 64;
void loop(){
    if(digitalRead(A0) == LOW){
        d = down;
    }
    else if(digitalRead(A1) == LOW){
        d = left;
    }
    else if(digitalRead(A4) == LOW){
        d = up;
    }
    else if(digitalRead(A5) == LOW){
        d = right;
    }

    if(d == up){
        py--;
    }
    else if(d == down){
        py++;
    }
```

```
    else if(d == left){
        px--;
    }
    else if(d == right){
        px++;
    }

    if(px < 2){
        px = 2;
    }
    if(px > 158){
        px = 158;
    }
    if(py < 2){
        py = 2;
    }
    if(py > 126){
        py = 126;
    }

    x[i] = px;
    y[i] = py;

    TFTscreen.stroke(255, 255, 255);
    TFTscreen.point(x[i], y[i]);
    delay(100);
    TFTscreen.stroke(0, 0, 0);
    TFTscreen.point(x[i - 4], y[i - 4]);

    i++;
}
```