子どもＩＴ未来塾

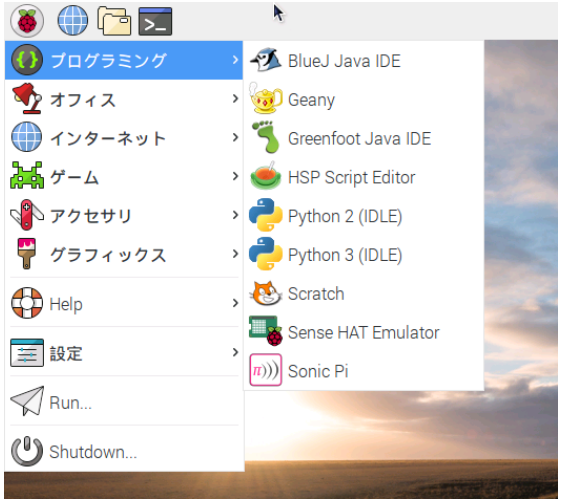
第 8 回(前半)　　「プログラム演習(2)」

8-1 HSP の復習

RaspberryPi を起動して、いままでに習ったプログラミングの復習をしてみましょう。
ゲームやハードウェア制御など、スクリプトを作ることで色々なことができます。
どのような手順で作ればいいか、もう一度確認してみましょう。

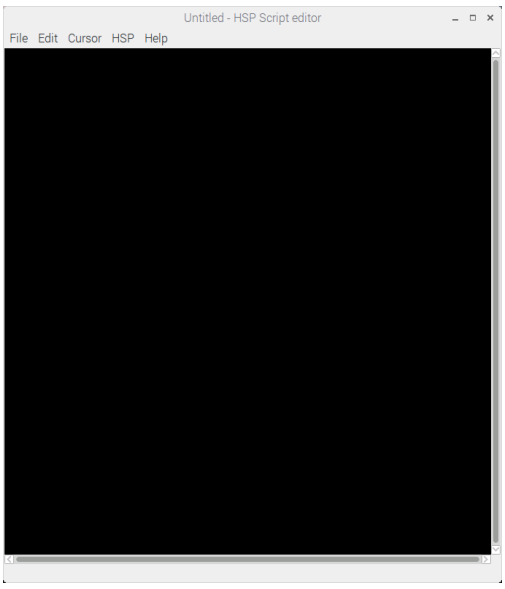
- **スクリプトエディタ起動**
- **スクリプトの読み込みと実行のしかた**
- **色の付け方**
- **文字の表示**
- **図形の描き方**

まず、プログラミングのツール HSP(Hot Soup Processor)を起動して、スクリプトを実行するための準備をします。左上の Raspberry Pi メニューの「プログラミング」項目にある、「HSP Script Editor」(HSP スクリプトエディタ)からエディタを起動してください。



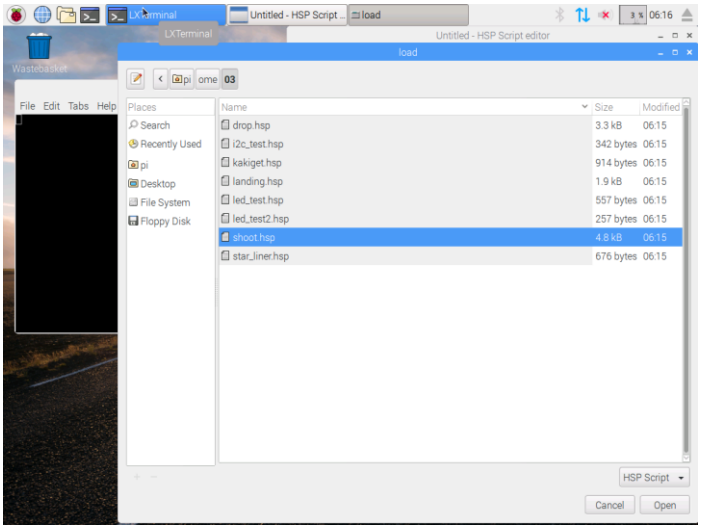
プログラミング項目のメニュー

HSP スクリプトエディタの画面が出てきます。
ここで色々なスクリプトを編集したり、読み込んで実行させることができます。



HSP スクリプトエディタを起動したところ

実際にできあがっているスクリプトを動かして試してみましょう。
ファイル→「開く」メニューから ome/08 フォルダの中にある「moji.hsp」を読み込んでください。
作業は、すべて「pi」から選択できる「ome/08」フォルダで行ないます。
見つからない場合は、まわりの友達か、近くの先生に聞いてみてください。



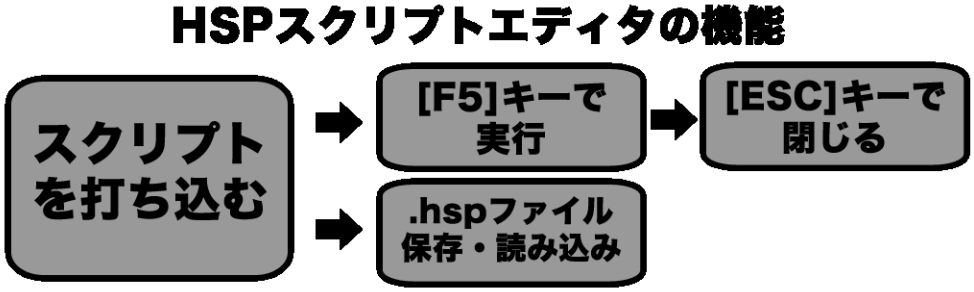
HSP フォルダから読み込む画面

スクリプトエディタで編集しているスクリプトを実行するには、「F5」キーを押すということを覚えていますか?

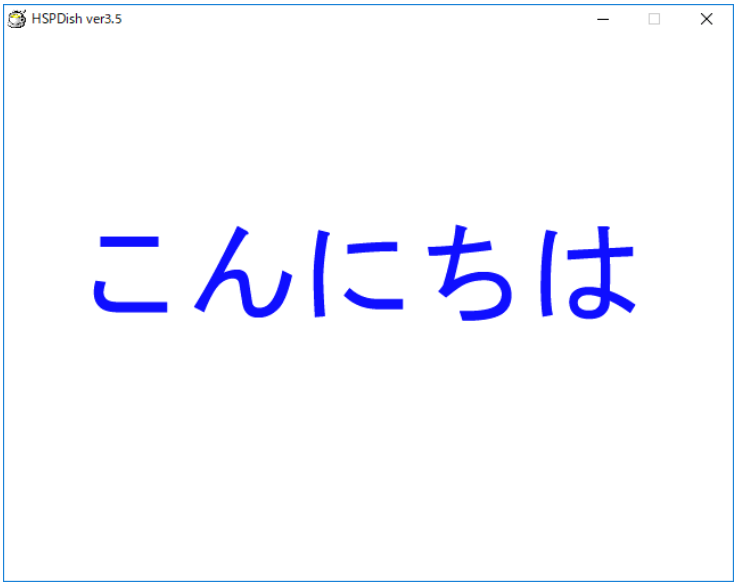
```
#include "hsp3dish.as"

redraw 0
font "",100
color 0,0,255
pos 60,160
mes "こんにちは"
redraw 1
```

スクリプトを編集して実行することが、プログラミングの基本になるのでよく覚えておきましょう。



mes 命令が文字を表示するための命令になります。
font や color、pos 命令の機能を覚えていますか? すべてを覚えておく必要はありませんが、繰り返し使っていくと、どのような命令があるのか、思い出せるようになってくるはずです。



[F5]キーで実行したところ

命令を追加して、背景に図形を出してみましょう。
color 命令で色を指定して、boxf 命令で四角形を出すことができます。
たとえば、水色で画面の一部に四角形を出すとこんな感じになります。

背景に四角形を出したところ

スクリプトのどこに追加すれば、背景に水色の四角が出るようになるでしょうか?
水色を出すためには、color 命令を使います。

```
color 0,255,255
```

四角形を出すためには、boxf 命令を使います。
左上の X,Y(横、縦の位置)、右下の X,Y をパラメーターとして書く必要があります。

```
boxf 左上 X, 左上 Y, 右下 X, 右下 Y
```

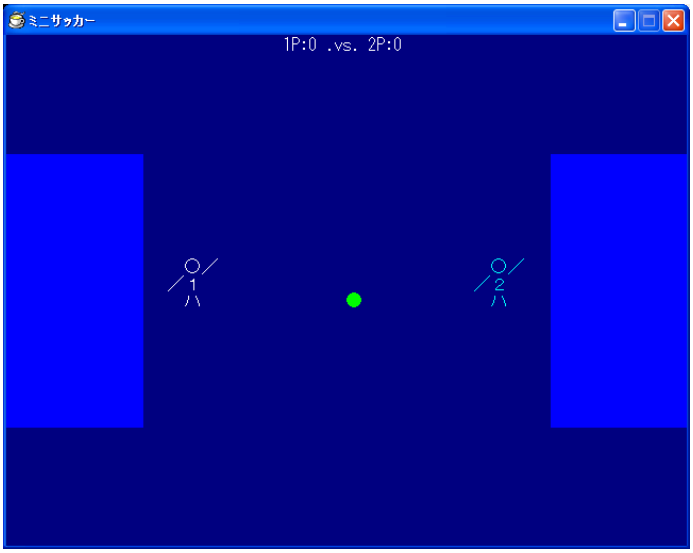
この 2 つを実際にスクリプトエディタで入力してみましょう。

```
color 0,255,255
boxf 100,100,540,380
```

入力できたら、[F5]を押して、ちゃんと背景に四角が表示されるか確かめてみましょう。

8-2 ミニサッカーゲームで遊んでみよう

文字と簡単な図形を使うだけでもゲームを作ることができます。実際にできあがっているプログラムを動かして試してみましょう。
スクリプトエディタの、ファイル→「開く」メニューから「kick.hsp」を読み込みましょう。
ミニサッカーゲームが動きます。



ミニサッカーゲームの画面

このゲームは二人で対戦します。
隣の席にいる人と、２人１組になって一緒に遊んでみてください。
左側の「１」の人をキーボードの左側で操作します。

　　[W]
[A]　　[D]　　[S] = ゴール前にもどる
　　[X]

右側の「２」の人をキーボードの右側で操作します。

　　[↑]
[←]　　[→]　　[Enter] = ゴール前にもどる
　　[↓]

「１」 の人は右のゴールへ
「２」 の人は左のゴールへボールを入れたら点が入ります。

３点先に取った人の勝ちです。

8-3 ミニサッカーゲームを改造してみよう

・改造できる部分

でたらめに書き直してもエラーが出るだけです。
命令にあたる部分は変えないようにしましょう。
前に覚えた「mes」や「color」などの命令があるはずです。
自分のキャラクターや色を自由に変えてみて動かしましょう。
さらに、「kickx = 1.8」の数字を少しだけ大きくしてみましょう。

(大きくしすぎないようにしてください)
「kicky = 2.0」なども変えてみると、どうなるか確認してみましょう。

8-4 休憩

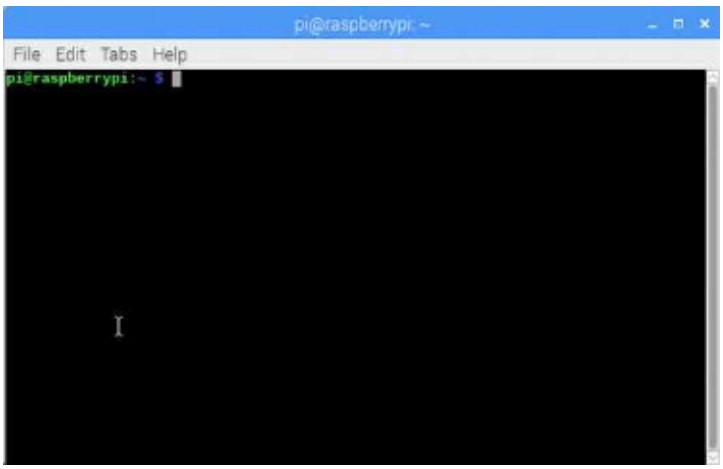
・自由時間です ・次の時間が始まる前に教室の席に戻るようにしましょう ・質問や疑問などがあれば先生のところまで聞きに行きましょう

8-5 文字だけのコンピューター

文字だけしか出ない画面は、あまり華やかではありません。
昔のコンピューターは、文字だけしか表示できませんでした。実は、現在のコンピューターも多くは文字だけでプログラミングされ、管理されています。

古いコンピューターの端末(DEC VT100)

皆さんが使っている RaspberryPi も、その伝統を引き継いでいます。
大量の情報を、できるだけ簡潔に、手軽に使えるように工夫されているのがターミナルの画面です。
コマンドを打ちこむために開いているターミナル(コンソール)は、Unix(または Linux)と呼ばれるシステムで、世界中に利用者がいる歴史のある OS です。



ターミナル画面

プログラミングで使っている HSP も、文字だけのバージョンと、絵の出るバージョンの 2 つを選ぶことができます。
よく使ってきた、絵がでる全画面で動く HSP は、HSP3Dish と呼ばれています。
スクリプトの最初に、このような行があるはずです。

```
#include “hsp3dish.as”
```

この行がある時は、絵を出すことのできる HSP。つまり、HSP3Dish を選んだことになります。
HSP3Dish は、全画面で実行されて、それまでの画面は見えなくなります。
[ESC]キーを押すと、実行を中断して元の画面に戻ってきます。

文字だけしか出ない、コンソール画面の上で動く HSP は、HSP3CL と呼ばれています。
スクリプトの最初に、このような行を書くことで選ばれます。

```
#include “hsp3cl.as”
```

ハードウェア制御や、データを扱う時など、文字だけしか使わない場合もあります。
その場合は、文字だけが表示される HSP3CL も使われることがありますので覚えておきましょう。

HSP3Dish や HSP3CL のどちらからも、ターミナルの画面で打ち込むコマンドを利用することができます。
ターミナルのコマンドは、コンピューターで使用している設定やファイルの操作など、あらゆることができるようになっています。
皆さんも、ファイルの一覧を見る時に「ls」というコマンドを使ったり、ファイルのコピーをするために、「cp」というコマンドを使ったりしています。
このコマンドを HSP から利用するためには、exec という命令を使います。
HSP では、「exec “ターミナルのコマンド”」という形式で命令の 1 つとして実行します。
たとえば、

```
exec “ls”
```

と書いた場合には、「ls」というコマンドが実行されます。
HSP3CL で実行した場合は、ファイルの一覧が出てくることになります。
HSP には、画面の描画や、音の再生など豊富な機能を持っていますが、ターミナルのコマンドを呼び出すことで、さらに色々な機能を呼び出すことができるようになっています。

8-6 絵を出してみよう

とは言うものの、ゲームを作るような場合には絵を出したいですよね。
ここでは、HSP で文字だけでなく、絵を出す方法について学んでいきましょう。

まずは、簡単なスクリプトから見てみましょう。
スクリプトエディタの、ファイル→「開く」メニューから ome/08 フォルダの中にある「celput.hsp」を読み込んでください。
終わったら、さっそく[F5]キーを押してスクリプトを実行してみましょう。

celput.hsp の実行画面

今度は、背景に絵が出ました。
この絵は、あらかじめ「sozai1.jpg」というファイルで用意されているものです。

実際に絵を出すためには、画像ファイルと呼ばれる絵のデータが必要になります。
それが、「sozai1.jpg」になります。(画像ファイルは、GIMP ツールや、ブラウザなどで開くことができます。)

絵を出すためには、2 つの命令を使う必要があります。 1 つ目の celload は、最初に必要な絵の素材を知らせるための命令です。これにより、指定された素材を表示する準備をします。

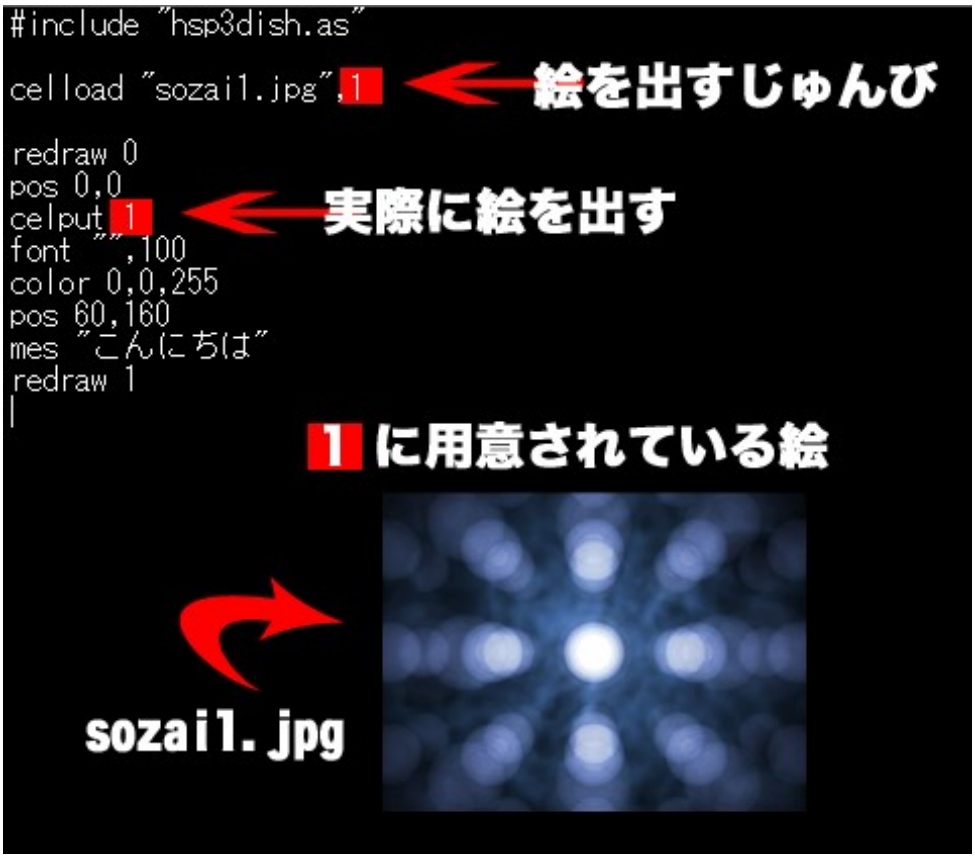
celload “画像ファイル名”, 絵の番号　　← 絵を出す準備をする

一度、celload 命令によって登録された絵は、celput 命令によって表示させることができます。
celload 命令は、最初の 1 回だけ実行すれば良いです。以降は、celput 命令によって、mes 命令や boxf 命令と同じように絵を出すことができるようになります。

celput 命令は、以下のように書くことができます。

celput 絵の番号

絵の番号というのは、celload 命令で指定していた番号のことです。
出すための絵を登録する時に、1,2,3,4…というように、別々な番号を割り振っておいて、同時に色々な絵を出せるようにするための仕組みです。
最初は、番号 1 を使っておきましょう。



つまり、

celload “sozai1.jpg”,1

は、番号 1 で「sozai1.jpg」の絵を表示するための準備をするという意味になります。
celput で指定した番号に準備された絵を表示します。
表示する位置は、mes 命令などと同じように pos 命令で指定された位置になります。

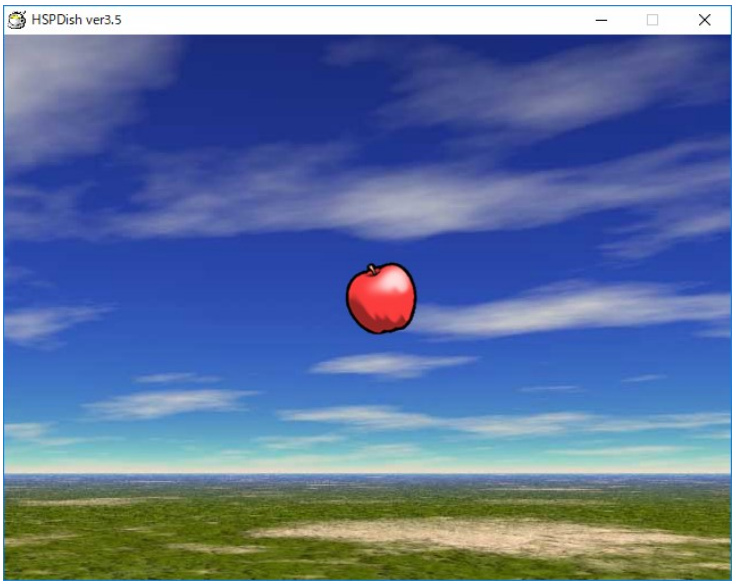
準備する画像のファイルは、スクリプトがある場所と同じフォルダに入れておいてください。
ですから、

/home/pi/ome/08/celput.hsp

というスクリプトの中で準備する画像ファイル、「sozai1.jpg」は、「/home/pi/ome/08/sozai1.jpg」にあるはずです。今回は、あらかじめ画像ファイルを用意していましたが、皆さんが自分で画像を作ったり、コピーして使用することもできます。

celload 命令と celput 命令、絵を出す番号と画像ファイルの関係をよく覚えておきましょう。

もう少し別な絵を表示してみましょう。
スクリプトエディタの、ファイル→「開く」メニューから ome/08 フォルダの中にある「apple.hsp」を読み込んで実行してみてください。

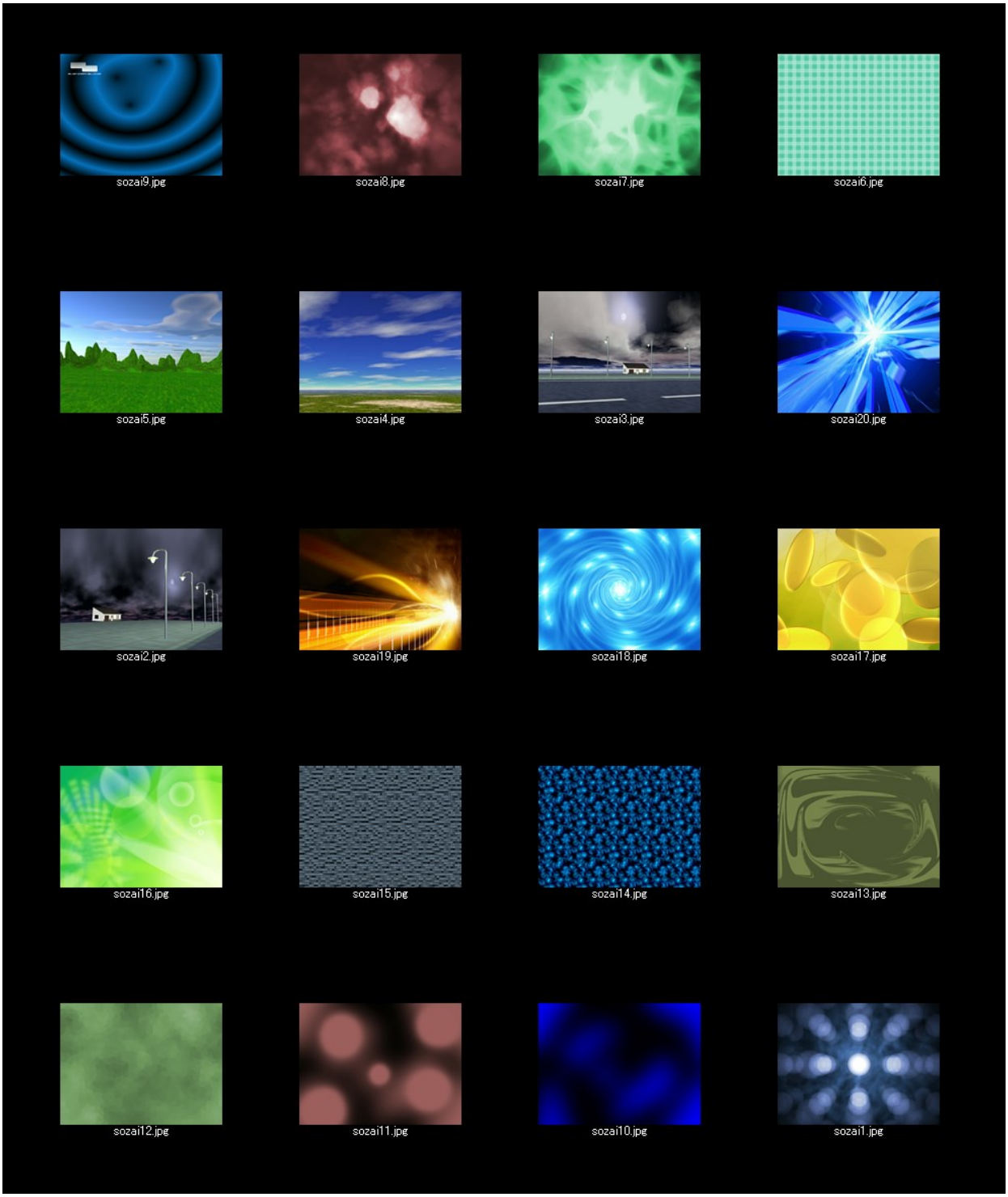


apple.hsp の実行画面

今度は、別な背景の絵になりました。
実は、「/home/pi/ome/08」には、自由に使える素材として「sozai1.jpg」から「sozai20.jpg」まで 20 種類の画像ファイルが含まれています。

次のページで、実際に使用できるファイルと画像の一覧があるので参考にしてください。

[資料] 背景として使える画面の一例
（sozai1.jpg ～ sozai20.jpg）



試しに、celload 命令で指定している画像ファイル名を他のものに変更して、実行してみましょう。
このように自由な絵を使って画面を作っていくことができます。
ところで、画面の中央に表示されている「りんご」だけは変わっていないことに気付いたでしょうか。

実は、このスクリプトでは背景と「りんご」の 2 つの画像を使っています。
背景の上に「りんご」を表示しているのです。

リンゴの画像(apple.png)

```
#include "hsp3dish.as"

celload "sozai4.jpg",1  ; 背景
celload "apple.png",2   ; りんご

x=300:y=200

*main
        redraw 0
        pos 0,0
        celput 1        ; 背景を表示
        gmode 2         ; 重ね合わせる
        pos x,y         ; りんごを表示
        celput 2
        redraw 1
        await 1000/30
        goto *main
```

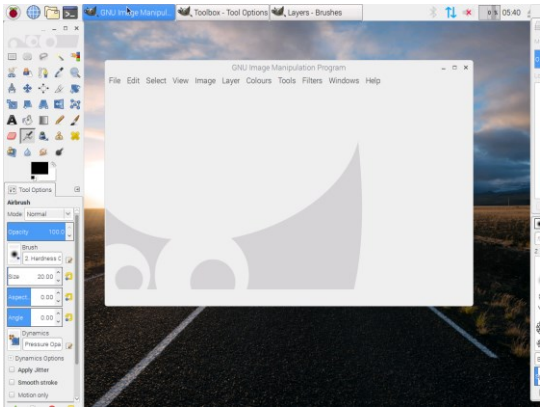
このように複数の番号を登録して、celput 命令を使うことで絵を重ねて表示することができます。
そのために、gmode という新しい命令が使われています。

(HSP のルール)

画像を重ねる設定をするには gmode 命令を使います
gmode の後、スペースに続けてコピーモードを示す数字を指定します
コピーモードは、以下のような意味があります

0 のとき	：画像のまま表示
1 か 2 のとき	：透明な部分は背景を残して表示
3 のとき	：透明度を変えて表示

「gmode 2」を指定することで、リンゴの透明な部分は背景が表示されて、きれいに重なっているように見えています。試しに、「gmode 0」にしてリンゴの絵がどのように表示されるか確認してみましょう。
リンゴのように透明な部分がある絵は、以前にも使った GIMP ツールで作ったり、確認することができます。GIMP は、Raspberry Pi メニューからグラフィックス→「GNU Image Manipulation Program」をクリックすることで起動させることができます。



Gimp を起動したところ

celput 命令には、実はもっと多くのパラメーターがあります。

celput 絵の番号, 分割画像 No. , 横方向の表示倍率, 縦方向の表示倍率, 回転角度

横方向の表示倍率=0.0～(1.0)： 横方向の表示倍率(実数)
縦方向の表示倍率=0.0～(1.0)： 縦方向の表示倍率(実数)
回転角度=0.0～(0.0)： 回転角度(実数)

絵を自由な大きさ、角度で出すことができるようになっています。
通常は、倍率 1、角度 0 の状態で表示されます。
分割画像 No.は、1 つの画像を複数の小さなブロックに分けて表示するための機能です。
(これについては、また別の機会に説明します。)

試しに、表示倍率や角度を変えてみて、どのように表示が変わるか確認してみましょう。
また、余裕がある人は、文字を重ねてさらに複雑な画面を作ることに挑戦してください。

8-7 絵を動かすには

今度は、表示した絵を動かしてみることにしましょう。
絵が動くことで、ゲームやアニメーションなどさらに応用が広がります。
そのために覚えておかなければならないのが、変数です。

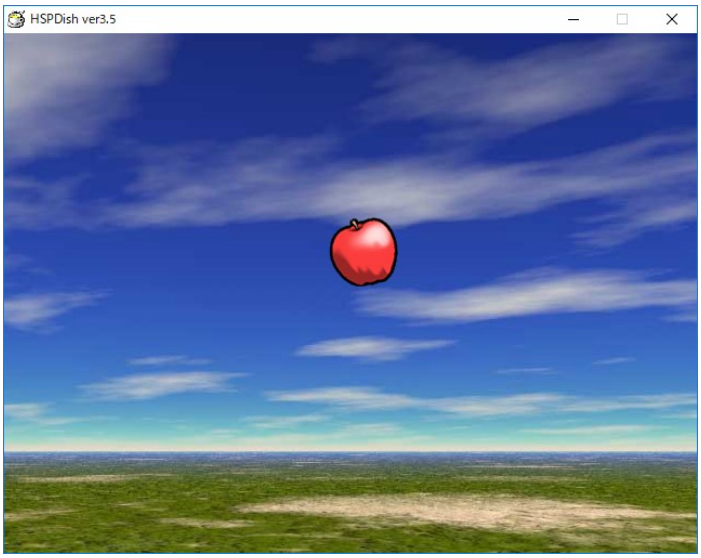
変数について学んだことを覚えていますか?

(HSP のルール)

「変数の名前=数値」で変数に値を入れることができる(代入)
変数に数値を入れておくとパラメーターとして使える
mes 命令の中で「+変数」を書くことで変数の中身を表示できる

「a=100」と置くと、変数 a に 100 という数字が記憶されます。
これを使って､いままで数字を書いていた部分に変数を代わりに書いて数字を変化させることができます。

スクリプトエディタの、ファイル→「開く」メニューから ome/08 フォルダの中にある「fall.hsp」を読み込んで実行してみてください。



fall.hsp の実行画面

今度はリンゴが動いているのがわかると思います。

```
#include "hsp3dish.as"

celload "sozai4.jpg",1  ; 背景
celload "apple.png",2   ; りんご

        x=300:y=0
*main
        redraw 0
        pos 0,0
        celput 1        ; 背景を表示
        gmode 2         ; 重ね合わせる
        pos x,y         ; りんごを表示
        celput 2
        redraw 1
        await 1000/30

        ;       りんごを動かす
        y=y+2
        goto *main
```

スクリプトを見てみましょう。
変数 x と変数 y が、リンゴの横、縦の位置を記憶しています。
最初に「x=300:y=0」の行で、変数 x に 300 を、変数 y に 0 を代入しています。
「pos x,y」がありますので、celput する時の位置を変数 x,y で変えられるようになっています。

実際にリンゴの位置を変えているのが、「y=y+2」の部分です。
ここを繰り返し実行することで、変数 y の値が 2 ずつ増えていきます。

(HSP のルール)

数式とは、数値と変数、またはそれらを計算式でつなげて書くこと
「+」は足し算、「-」は引き算 、「*」は掛け算(×)、「/」は割り算(÷)

「y=y+2」は変数 y に記憶されている数字に 2 を足すという意味だと覚えておきましょう。
これで、縦の位置が少しずつ変化するようになります。
足す値を 2 ではなく、他の値にしてどうなるか試してみましょう。

8-8 条件判断で自動化する

「fall.hsp」のスクリプトは、一度落ちてしまったリンゴはもう表示されなくなります。
縦の座標に指定している値が大きくなり、画面に入らない位置まで行ってしまうためです。

そこで、画面の下まで行ったら、また上に戻すように変更してみましょう。
これは、条件判断を使うことで実現できます。条件判断により、色々なことが自動化できます。
スクリプトエディタの、ファイル→「開く」メニューから ome/08 フォルダの中にある「rndfall.hsp」を読み込んで実行してみてください。

今回は、下まで行ったリンゴがまた上から出てくるようになりました。
また、横の位置も乱数を使ってばらばらに位置に出るようになっています。
条件判断がどのように使われているか確認してみましょう。

```
#include "hsp3dish.as"

celload "sozai4.jpg",1  ; 背景
celload "apple.png",2   ; りんご

randomize

*main
        x=rnd(600)      ; 横の位置を乱数で決める
        y=-80           ; 縦の位置を画面上に
*main2
        redraw 0
        pos 0,0
        celput 1        ; 背景を表示
        gmode 2         ; 重ね合わせる
        pos x,y         ; りんごを表示
        celput 2
        redraw 1
        await 1000/30

        ;       りんごを動かす
        y=y+2
        if y>480 : goto *main
        goto *main2
```

リンゴの縦の位置を記憶している変数 y の内容が、一定の値を超えたら、また上に戻すというスクリプトになっています。

if y>480：goto *main

上の部分がどのような意味か考えてみましょう。
if 命令の使い方、ルールを覚えていますか?

(HSP のルール)

if 命令により条件を判断することができる
if の後にスペースに続けて条件式を指定します
その後で「:」に続けて条件が正しい時に実行される命令を書きます

(条件式はいくつか書き方があります)

条件式	意味
変数名 = 数値	変数の内容と数値が同じである
変数名 ! 数値	変数の内容と数値が同じではない
変数名 < 数値	変数の内容より数値の方が大きい
変数名 > 数値	変数の内容より数値の方が小さい

最初は、条件判断の仕組みがわかりにくいかもしれませんが、スクリプトの１行目から、コンピューターが実行することを想像しながら、１つ１つ確認することが大切です。
やがて、スクリプトを見て、どのように動くのか想像ができるようになります。
最初はちょっと難しく感じるかもしれませんが、慣れれば誰でも理解できるようになります。
あせらず、わからない所はまわりの先生や友達に聞きながら、進んでいきましょう。

条件判断によって、色々な入力を調べることができます。

スクリプトエディタの、ファイル→「開く」メニューから ome/08 フォルダの中にある「move.hsp」を読み込んで実行してみてください。

```
#include "hsp3dish.as"

celload "sozai4.jpg",1  ; 背景
celload "apple.png",2   ; りんご

x=300:y=200

*main
        redraw 0
        pos 0,0
        celput 1        ; 背景を表示
        gmode 2         ; 重ね合わせる
        pos x,y         ; りんごを表示
        celput 2
        redraw 1
        await 1000/30

        getkey a,'A'
        if a=1 : x=x-4
        getkey a,'W'
        if a=1 : y=y-4
        getkey a,'D'
        if a=1 : x=x+4
        getkey a,'X'
        if a=1 : y=y+4

        goto *main
```

今度はリンゴを「A」「W」「D」「X」のキーを押して動かすことができます。何だかゲームみたいですね。
キーボードが押されているかどうかを条件判断して座標を変化させています。
新しい命令、getkey の使い方を覚えておきましょう。

getkey 変数 , ‘キーの文字’

と書くことで、指定した変数にキーが押されたかどうかが数値として代入されます。
たとえば、

getkey a, ‘X’

を実行すると、「X」キーの状態で変数 a の値が 0 か 1 になります。
「X」のキーが押されていない時は、変数 a は 0 になります。
「X」のキーが押されている時は、変数 a は 1 になります。

キーの文字は、必ず大文字で’A’のように書く必要があるので注意してください。
文字の代わりにキーコードと呼ばれる数字を指定することで、特殊なキーの状態を知ることもできます。

キーコード	：実際のキー
1	：マウスの左ボタン
2	：マウスの右ボタン
8	：[BACKSPACE]
9	：[TAB]
13	：[ENTER]

16　：[SHIFT]
17　：[CTRL]
18　：[ALT]
32　：スペースキー
37　：カーソルキー[←]
38　：カーソルキー[↑]
39　：カーソルキー[→]
40　：カーソルキー[↓]
48～57　：[0]～[9]（メインキーボード）
65～90　：[A]～[Z]
96～105　：[0]～[9]（テンキー）
112～121　：ファンクションキー [F1]～[F10]

こうして代入された 0 か 1 の値を条件判断によって、やりたい内容を書くことで色々なことに応用ができます。
余裕のある人は、自分で色々な条件判断を使ってリンゴだけでなく、色々なものを動かすことにも挑戦してみましょう。

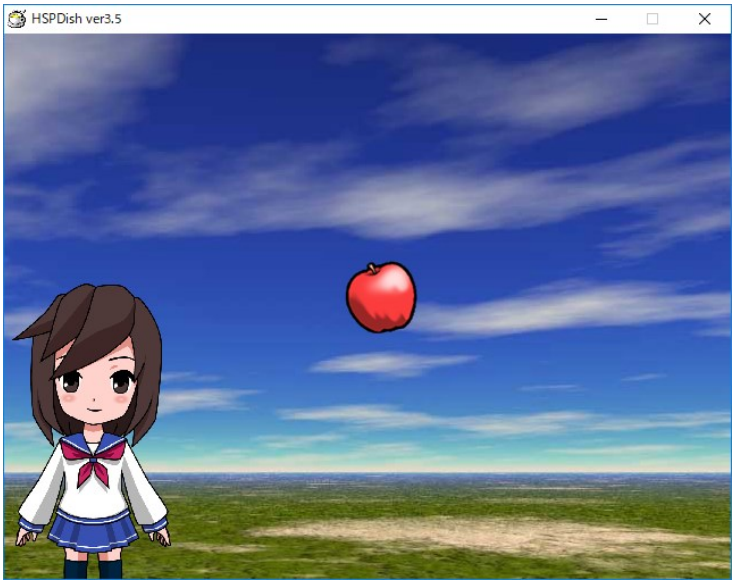
8-9 光センサーの値で絵を動かす

アイデア次第で、キーボードだけでなく、センサー類を使ってゲームのようにすることができます。
光センサーを使ったゲームを動かしてみましょう。

スクリプトエディタの、ファイル→「開く」メニューから ome/08 フォルダの中にある「catch.hsp」を読み込んで実行してみてください。

今度は、センサーボードの光センサーから得た値を使って絵を動かしています。光センサーを手で覆ったり、離したりしてどうなるか確認してみましょう。

光センサーが暗くなった時には、キャラクターが左に移動します。光センサーが明るくなった時には、キャラクターが右に移動します。



catch.hsp の実行画面

うまくキャラクターを移動させて、上から落ちてくるリンゴをキャッチしてみましょう。

これは、いままでの応用ですが、センサーと動きのある画面を組み合わせるだけでも、面白いものが作れるはずです。
センサーだけでなくキーボードで動かしたり、別なセンサーを使ったりしてもいいでしょう。
前回使った gpio 命令で、LED を光らせたり、スイッチを読み取ったりということと、変数・条件判断を組み合わせれば、さらに応用範囲が広がります。

大切なことは、自分で何でも試してみることです。わからないことがあっても、実際に実行すれば結果がわかります。立ち止まらずに、どうなるか想像しながらスクリプトを書いて実行することが、早く上達するコツになります。
それでもわからない時は、先生に聞いてみましょう。

8-10 わからないことがあったら…

「子ども IT 未来塾」の参加者が利用できる質問フォームが開設されています。
授業の中でわからないことがあった時､確認したいことなどがあればインターネットブラウザから以下の URL を開いて、直接先生に質問をすることができます。

子ども IT 未来塾 質問フォーム
https://bit.ly/2NHiVgi
(大文字小文字を間違えないように入力してください)

質問の返事はメールで返ってきますので、必ずメールアドレスを入力してください。
時間がかかる場合がありますので、返事が来るまでに 2～3 日待ってみてください。

8-11 休憩・退出

・今日の教室は終わりです
・質問や疑問などがあれば先生のところまで聞きに行きましょう
・時間までに家に帰るようにしましょう