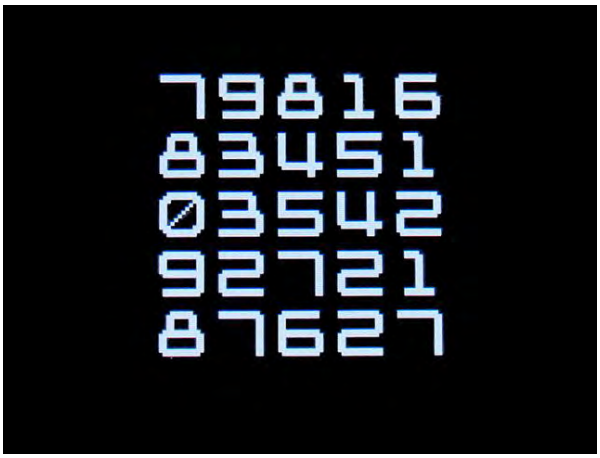
プチコンでパズルバトルゲームを作る（２）

●数字パネルの数字を変数に記憶する



前回までで、下画面に 5×5＝25 枚の数字パネルを表示するプログラムを作りました。
ただし、これまでのプログラムは、ただ数字を画面に並べて表示しただけです。
今後、タッチペンで数字パネルを操作するプログラムを作るのですが、その時には

どのパネルが何の数字なのか

を覚えておかないといけません。
そうしないと、タッチした場所のパネルの数字が何なのかわからず、パネルの数字の合計も計算できないからです。

そのため、タッチパネルの数字を変数に記憶しておいて、その変数の値を元に画面に表示するようにします。
ただ、パネルは 25 枚あるので、例えば 1 枚目→「A」、2 枚目→「B」…と変数を割り当てていっても、管理が面倒になります。
このように同じものがたくさんある時は、その値を記憶するのに**配列（はいれつ）変数**を使います。配列変数は、番号がついた変数です。
配列変数は、プログラムの最初で配列変数の宣言をしてから使います。

```
1 '*** PUZZLE3 ***
2
3 ACLS
4 XSCREEN 2
5
6 DIM P[5,5]
7
8 DISPLAY 1
```

タイトルを「PUZZLE3」にする

配列の宣言

配列の宣言は **DIM**（ディム）命令を使います。（※DIM は dimension〈次元〉の略です）

DIM P[5,5]
　配列変数と個数を指定。数字は 0 から始まる。

このように宣言すると、配列変数として、P[0,0]～P[4,4]まで、25 個の変数が使えます。
（数字は 0 から始まるので、最大値は個数－1までになります）

配列変数を数字パネルに当てはめると、
右の図のようになります。
一番左上のパネルの数字が P[0,0]、
その右のパネルの数字が P[0,1]、
　　：
一番右下のパネルの数字が P[4,4]
に記憶されます。

配列変数P[x,y]

↓y方向 / →x方向				
P[0,0]	P[1,0]	P[2,0]	P[3,0]	P[4,0]
P[0,1]	P[1,1]	P[2,1]	P[3,1]	P[4,1]
P[0,2]	P[1,2]	P[2,2]	P[3,2]	P[4,2]
P[0,3]	P[1,3]	P[2,3]	P[3,3]	P[4,3]
P[0,4]	P[1,4]	P[2,4]	P[3,4]	P[4,4]

プログラムを改造して、配列変数にパネルの数字を記憶させる部分と、配列変数を元に画面にパネルを表示する部分に分けます。わかりやすくするためにコメントも入れます。

```
 1 '*** PUZZLE3 ***
 2
 3 ACLS
 4 XSCREEN 2
 5
 6 DIM P[5,5]
 7
 8 '=== パネル セット ===
 9
10 FOR Y=0 TO 4
11  FOR X=0 TO 4
12   P[X,Y]=RND(10)
13  NEXT
14 NEXT
15
16 '=== パネル ひょうじ ===
17
18 DISPLAY 1
19 FOR Y=0 TO 4
20  FOR X=0 TO 4
21   N=P[X,Y]
22   SPSET X+Y*5,48+N
23   SPOFS X+Y*5,80+32*X,32+32*Y
24   SPSCALE X+Y*5,2,2
```

P[X,Y]に 0～9の乱数を設定

P[X,Y]から数字を取り出す

```
25    SPSHOW X+Y*5
26  NEXT
27 NEXT
28 WAIT 600
```

プログラムを実行して、数字パネルがちゃんと表示されるか確認してください。
（画面の見た目は以前と変わりません）

【参考情報】

配列変数の[ ]の中の数字は「添え字」（そえじ）と言います。
高校の数学では、「p_{00}」などの、添え字が付いた変数を習います。

今回は数字パネルのために、P[0,0]と添え字が2つある配列変数を使いました。
添え字が2つある配列は「2 次元配列」といいます。
プチコン3号では、1 次元～4 次元までの配列が使えます。

（例）
DIM A[10] ……A[0]～A[9]の 1 次元配列。変数 10 個。
DIM A[10,10] ……A[0,0]～A[9,9]の 2 次元配列。変数 100 個。
DIM A[10,10,10] ……A[0,0,0]～A[9,9,9]の 3 次元配列。変数 1,000 個。
DIM A[10,10,10,10] …A[0,0,0,0]～A[9,9,9,9]の 4 次元配列。変数 10,000 個。

●プレイヤーの体力を表示する

数字パネルの左上に、プレイヤーの体力（HP）を表示してみましょう。

```
 1 '*** PUZZLE3 ***
 2
 3 ACLS
 4 XSCREEN 2
 5
 6 DIM P[5,5]
 7
 8 HP=100
 9
10 '=== パネル セット ===

   （中略）

18 '=== パネル ひょうじ ===

   （中略）

28   NEXT
29 NEXT
30
31 LOCATE 0,0
32 COLOR 15
33 PRINT "プレイヤー"
34 PRINT "HP=";HP
35
36 WAIT 600
```

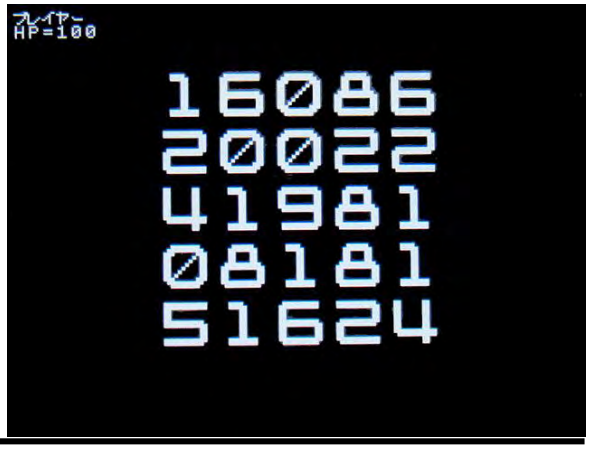
プレイヤーの HP を設定

カーソルを画面左上に移動

文字色を白にする

文字と HP の値を表示

プログラムを実行してみましょう。
数字パネル左上に、「プレイヤー」「HP=100」と表示されます。

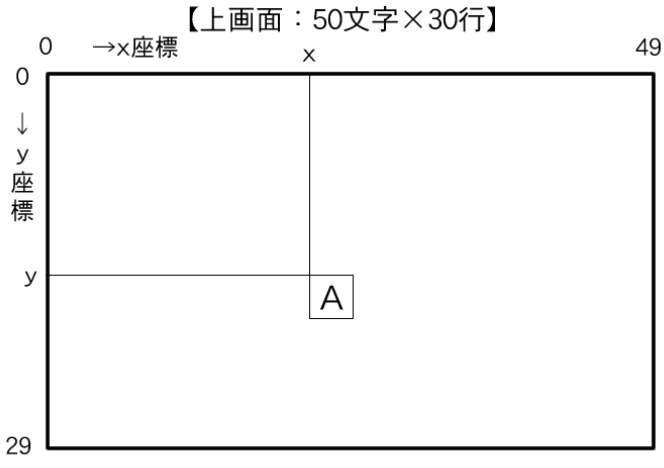


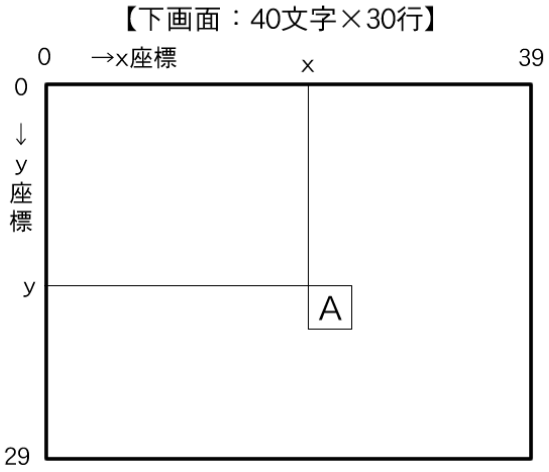
いくつか新しい命令が出てくるので、説明します。

★LOCATE（ローケイト）命令

```
LOCATE      0          ,0
          x 座標     y 座標
```

文字を表示するカーソルの位置を設定します。
文字を表示する画面（コンソール画面）のレイアウトは、右の図のようになっています。



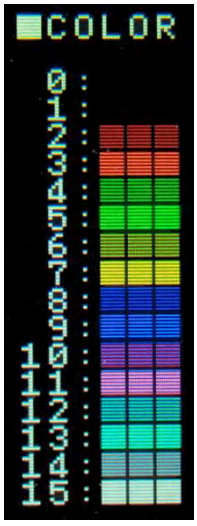


LOCATE 命令に続いて PRINT 命令を使うと、画面のいろいろな位置に文字を表示できます。
今回は「**LOCATE 0,0**」として、下画面の左上に文字を表示しています。

★COLOR（カラー）命令

```
COLOR       15          ,0
          描画色      背景色
```

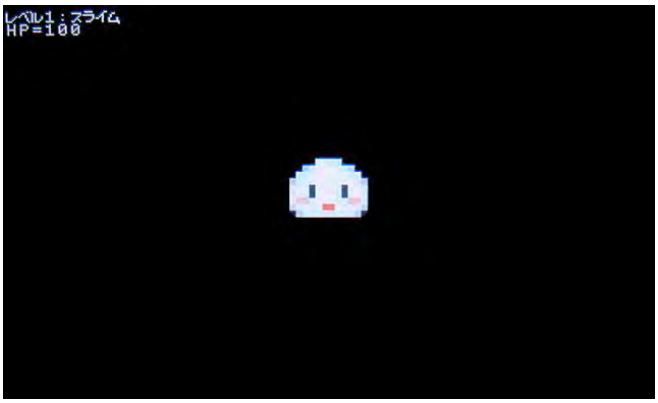
描画色	文字の色（0～15）。 0:透明　1:黒　2:暗い赤　3:赤 4:暗い緑　5:緑　6:暗い黄色　7:黄色 8:紺色　9:青　10:暗いマゼンタ　11:マゼンタ 12:暗いシアン　13:シアン　14:灰色　15:白
背景色	文字の背景の色。省略可。



画面に表示する文字の色を指定します。
色の番号は、SMILE ツールの上画面で確認できます。

プログラムを「**PUZZLE3**」の名前で保存（SAVE）しましょう。

●敵のモンスターを表示する



次は、上画面に敵のモンスターを表示しましょう。
数字パネルと同じように、モンスターの設定をするプログラムと、表示をするプログラムに分けます。

```
 1 '*** PUZZLE4 ***

   （中略）

31 LOCATE 0,0
32 COLOR 15
33 PRINT "プレイヤー"
34 PRINT "HP=";HP
35
36 '=== モンスター セット ===
37
38 MNAME$="スライム"
39 MS=1060
40 MHP=100
41
42 '=== モンスター ひょうじ ===
43
44 DISPLAY 0
45 SPSET 100,MS
46 SPSCALE 100,4,4
47 SPOFS 100,168,68
48 SPSHOW 100
49 LOCATE 0,0
50 COLOR 15
51 PRINT MNAME$
52 PRINT "HP=";MHP
53 WAIT 600
```

タイトルを「PUZZLE4」にする

モンスターの名前を**文字変数**に設定

モンスターのスプライト番号を設定

モンスターの HP を設定

表示を上画面に設定

スプライト管理番号 100 番をキャラ MS 番に

スプライトの表示倍率を 4 倍にする

スプライトの座標を画面中央に

スプライトを表示

モンスターの名前、HP を表示

プログラムを実行すると、上画面にモンスターが表示されます。

モンスターの設定をするプログラムで、モンスターの名前を記憶するのに、**文字変数**（もじへんすう）「MNAME$」を使っています。
変数の名前の後ろに「$」（ドル、ダラー）を付けると、文字を記憶する変数になります。

その他、モンスターのスプライト番号を変数 MS、モンスターの体力（HP）を変数 MHP に設定しています。
次のモンスターを表示するプログラムで、それらの変数の値を画面に表示しています。

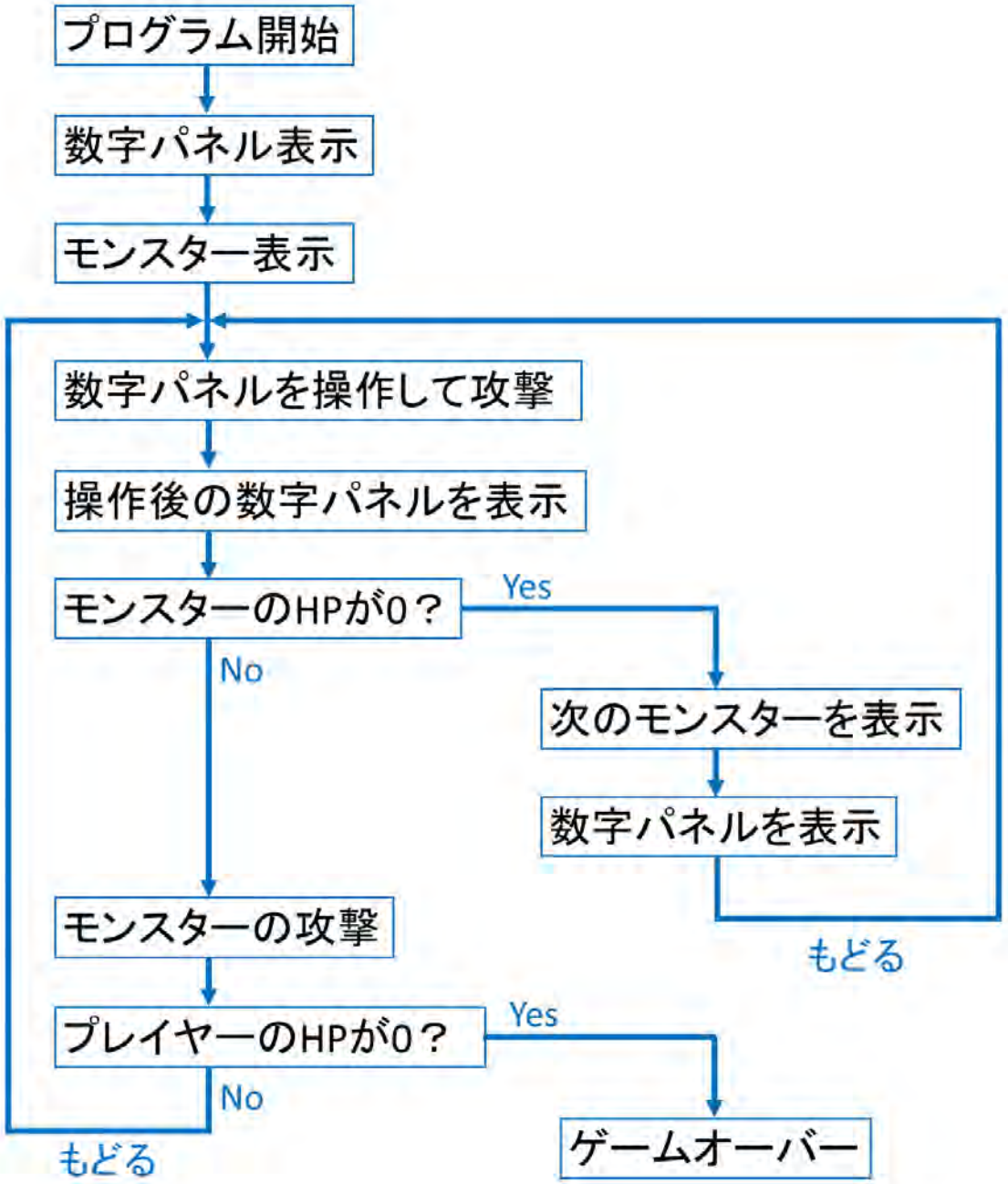
プログラムを「PUZZLE4」の名前で保存（SAVE）しましょう。

●プログラムの流れ・構成を考える

今後、

- 数字パネルを操作して、モンスターを攻撃する。
- もしモンスターの体力（HP）が 0 になったら勝ち。次のモンスターを表示。
- モンスターからも攻撃を受ける。
- もし自分の体力（HP）が 0 になったら負け。ゲームオーバー。
- 上にもどってくり返し

…というゲームのプログラムを作っていきます。
今後のために、プログラム全体の流れ（**フローチャート**）を考えてみます。



このようにプログラム全体の流れを考えると、「数字パネルを表示」「モンスターを表示」の処理は、何度も行われることがわかります。
同じプログラムを何度も打つのはムダなので、**サブルーチン**にして呼び出すようにします。

```
 1 '*** PUZZLE5 ***
 2
 3 ACLS
 4 XSCREEN 2
 5
 6 DIM P[5,5]
 7
 8 HP=100
 9
10 DISPLAY 1
11 LOCATE 0,0
12 COLOR 15
13 PRINT "プレイヤー"
14 PRINT "HP=";HP
15
16 GOSUB @SETPANEL
17 GOSUB @PRTPANEL
18 GOSUB @SETMONSTER
19 GOSUB @PRTMONSTER
20
21 '=== メイン ループ ===
22 @MAINLOOP
23
24 GOTO @MAINLOOP
25
26 '=== パネル セット ===
27 @SETPANEL
28
29 FOR Y=0 TO 4
30  FOR X=0 TO 4
31   P[X,Y]=RND(10)
32  NEXT
33 NEXT
34 RETURN
35
```

タイトルを「PUZZLE5」にする

表示画面を下画面に設定

プレイヤーの体力表示の行をコピーして、ここへ貼り付ける。元の行は削除する。

パネルを設定するサブルーチンを呼ぶ

パネルを表示するサブルーチンを呼ぶ

モンスターを設定するサブルーチンを呼ぶ

モンスターを表示するサブルーチンを呼ぶ

メインループ。今は何もせずにくり返し。プレイヤーのタッチ操作などの処理を、これから入れる。

サブルーチンを呼ぶためのラベルを追加

サブルーチンからもどる

（↓つづく↓）

```
36 '=== パネル ひょうじ ===
37 @PRTPANEL
38
39 DISPLAY 1
40 FOR Y=0 TO 4
41  FOR X=0 TO 4
42   N=P[X,Y]
43   SPSET X+Y*5,48+N
44   SPOFS X+Y*5,80+32*X,32+32*Y
45   SPSCALE X+Y*5,2,2
46   SPSHOW X+Y*5
47  NEXT
48 NEXT
49 RETURN
50
51 '=== モンスター セット ===
52 @SETMONSTER
53
54 MNAME$="スライム"
55 MS=1060
56 MHP=100
57 RETURN
58
59 '=== モンスター ひょうじ ===
60 @PRTMONSTER
61
62 DISPLAY 0
63 SPSET 100,MS
64 SPSCALE 100,4,4
65 SPOFS 100,168,68
66 SPSHOW 100
67 LOCATE 0,0
68 COLOR 15
69 PRINT MNAME$
70 PRINT "HP=";MHP
71 RETURN
```

サブルーチンを呼ぶためのラベルを追加

サブルーチンからもどる

サブルーチンを呼ぶためのラベルを追加

サブルーチンからもどる

サブルーチンを呼ぶためのラベルを追加

サブルーチンからもどる

プログラムを実行して、動作を確認してください。

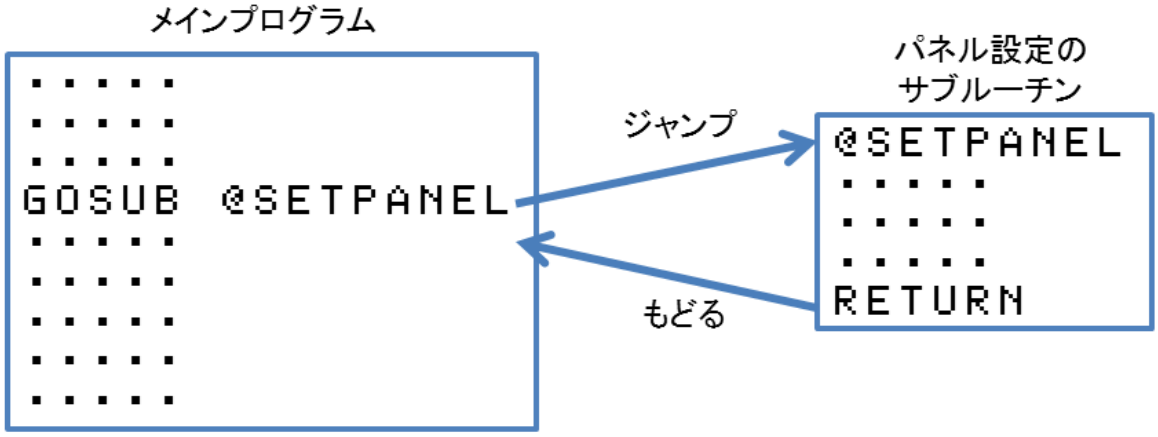
見た目は全く変わりませんが、最後の WAIT 命令が無くなったので、プログラムを止める時は 3DS の SELECT キーを押してください。

メインループでくり返しをするために、GOTO（ゴートゥー）命令を使います。

```
GOTO        @MAINLOOP
            ジャンプ先ラベル
```

ジャンプする先は「**ラベル**」で指定します。
ラベルは、「@」（アットマーク）に続けてアルファベットや数字で指定します。

メインプログラムで GOSUB（ゴーサブ）命令を使うと、プログラムの実行がサブルーチンへジャンプします。
サブルーチンの最後で RETURN（リターン）命令を使うと、メインプログラムにもどります。
メインプログラムでは、次の行から実行が再開されます。



プログラムの中で何度も同じ処理をする場合、その部分をサブルーチンにして、使いたい時に呼び出すようにすると、同じプログラムを何度も書かなくてすみます。

サブルーチンを呼び出すのは「GOSUB」（ゴーサブ）命令を使います。
GOSUB 命令の文法は以下のとおりです。

```
GOSUB       @SETPANEL
            ジャンプ先ラベル
```

GOTO 命令と同じように、ジャンプする先は「ラベル」で指定します。

サブルーチンからメインプログラムへ戻るには、RETURN（リターン）命令を使います。

プログラムを「PUZZLE5」の名前で保存（SAVE）しましょう。

●数字パネルのタッチ操作

ペンで数字パネルを操作するプログラムを、メインループの中に作っていきましょう。

プチコンでは、下画面でペンでタッチした場所の座標を、TOUCH（タッチ）命令で読み取ることができます。
メインループに、以下のプログラムを追加してみましょう。

```
 1 '*** PUZZLE6 ***

   （中略）

21 '=== メイン ループ ===
22 @MAINLOOP
23
24 TOUCH OUT STTM,TX,TY
25 LOCATE 0,20
26 PRINT "TX,TY=";TX;",";TY;"    "
27
28 GOTO @MAINLOOP
```

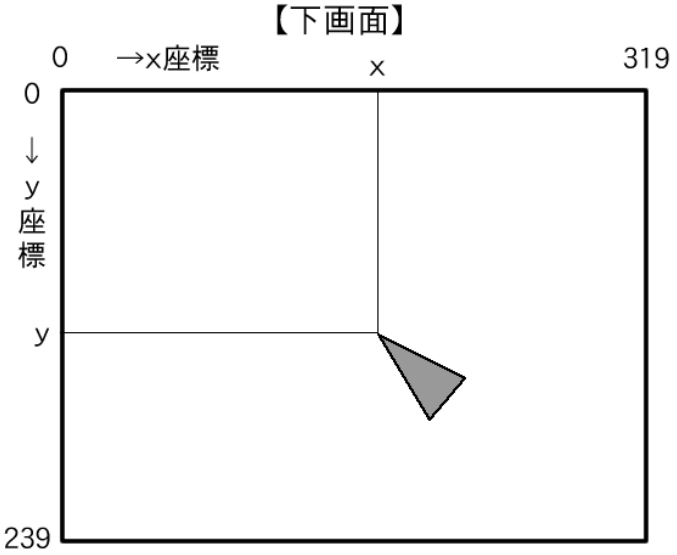
タイトルを「PUZZLE6」にする

タッチされた座標をTX,TY に読み取る

読み取ったTX,TY を画面に表示

プログラムを実行して、下画面にペンでタッチしてみましょう。
上画面に、タッチした場所の座標が表示されます。

以前も出した図ですが、下画面のグラフィック座標は、右の図のようになっています。
いろいろタッチして、座標の数字を確認しましょう。

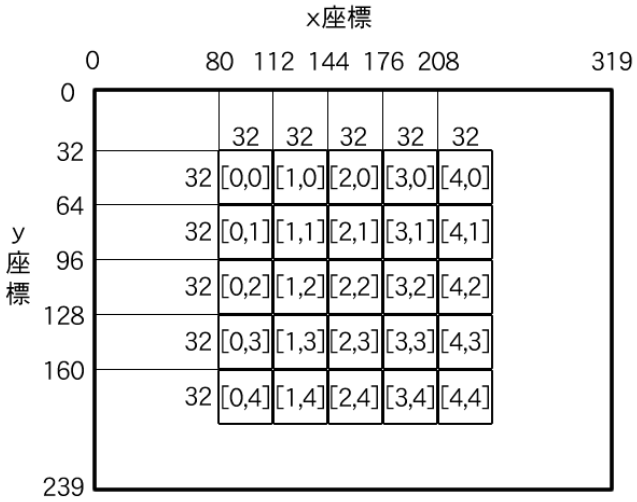


TOUCH 命令の文法は、以下のとおりです。

```
TOUCH OUT      STTM        ,TX       ,TY
              タッチ時間     x 座標     y 座標
```

タッチ時間	タッチされた時間を受け取る変数。タッチされていなければ 0。
x座標	タッチされた場所のx座標を受け取る変数（5～314）。 ※画面の端は検出されない
y座標	タッチされた場所のy座標を受け取る変数（5～234）。 ※画面の端は検出されない

このタッチ場所のグラフィック座標(TX,TY)から、どの数字パネル[0,0]～[4,4]がタッチされたのか、計算しないといけません。
数字パネルの番号とグラフィック座標は、右の図のようになっています。



まず、一番右上の[0,0]のパネルのグラフィック座標（左上の角）は(80,32)です。
そこから x 方向を考えると、パネルの x 方向の番号が 1 増えるごとに、グラフィック座標は 32 ずつ増えます。
同じように、y 方向を考えると、パネルの y 方向の番号が 1 増えるごとに、グラフィック座標は 32 ずつ増えます。
逆に考えれば、グラフィック座標を 32 で割れば、パネルの番号になるはずです。
実際はそこから(80,32)だけズレているので、その分を引き算してから 32 で割ります。

実際にパネル番号を計算して、画面に表示してみましょう。

```
'=== メイン ループ ===
@MAINLOOP

TOUCH OUT STTM,TX,TY
LOCATE 0,20
PRINT "TX,TY=";TX;",";TY;"    "
PX=(TX-80)/32
PY=(TY-32)/32
PRINT "PX,PY=";PX;",";PY;"    "

GOTO @MAINLOOP
```

タッチ座標(TX,TY)から
パネル番号(PX,PY)を計算

(PX,PY)を画面に表示

プログラムを実行してみましょう。
PX,PY の値が表示されます。
32 で割り切れないので、小数になってしまってよくわかりません。

PX,PY の値の小数点以下を切り捨てて、整数にします。
切り捨てには FLOOR（フロア）関数を使います。

```
'=== メイン ループ ===
@MAINLOOP

TOUCH OUT STTM,TX,TY
LOCATE 0,20
PRINT "TX,TY=";TX;",";TY;"    "
PX=(TX-80)/32
PY=(TY-32)/32
PX=FLOOR(PX)
PY=FLOOR(PY)
PRINT "PX,PY=";PX;",";PY;"    "

GOTO @MAINLOOP
```

PX,PY の小数点以下を切り捨て

プログラムを実行してみましょう。
パネル番号(PX,PY)が表示されます。
数字のパネルをいろいろタッチして、
ちゃんと[0,0]～[4,4]のパネル番号が
表示されるか、確認してください。

FLOOR 関数は、小数点以下の値を切り捨てる関数です。

```
B=FLOOR(      A)
```
元の値

（例）
A が 0.1 → B は 0
A が 0.9 → B は 0
A が 1.0 → B は 1
A が 1.1 → B は 1

※正確には「その値を超えない最大の整数を得る関数」なので、
A が-0.1 → B は-1 になります。
正の数で使っている時は、「切り捨て」と考えていいです。

●タッチされたパネルの表示を変える

ここまでの計算方法で、どの数字パネルがタッチされたか判断できます。
タッチされた数字パネルの表示を変えて、色を着けてタッチされたことがわかるようにしてみましょう。

今は数字パネルの数字を、配列変数 P[x,y]に記憶させています。
同じように、パネルがタッチされているかどうかを、配列変数 T[x,y]に記憶させます。

- T(x,y)=0 → タッチされていない
- T(x,y)=1 → タッチ中

として、その値をチェックしてパネルの表示を変えることにします。

```
1  '*** PUZZLE6 ***
2
3  ACLS
4  XSCREEN 2
5
6  DIM P[5,5],T[5,5]

   (中略)

21 '=== メイン ループ ===
22 @MAINLOOP
23
24 TOUCH OUT STTM,TX,TY
25 IF STTM==0 THEN @MAINLOOP
26 PX=(TX-80)/32
27 PY=(TY-32)/32
28 PX=FLOOR(PX)
29 PY=FLOOR(PY)
30 IF PX<0 OR PX>4 THEN @MAINLOOP
31 IF PY<0 OR PY>4 THEN @MAINLOOP
32 T[PX,PY]=1
33 GOSUB @PRTPANEL
34
35 GOTO @MAINLOOP
```

タッチ検出の配列 T[5,5]を追加

タッチされていなかったらもどる

パネル以外の場所をタッチしていたらもどる

T[]の値をタッチ中に変える

パネル表示サブルーチンを呼ぶ

新しく**IF**（イフ）～**THEN**（ゼン）命令が出てきました。**条件判断**をする命令です。

```
IF  STTM==0  THEN  @MAINLOOP  ELSE    ～    ENDIF
     条件式          処理1           処理 2
```

条件式	条件を判断する式 A==B　…A と B が等しい（==と 2 つ続ける） A>B　…A が B より大きい A<B　…A が B より小さい A>=B　…A が B 以上（等しい時もふくむ） A<=B　…A が B 以下（等しい時もふくむ） A!=B　…A と B が等しくない
処理 1	条件が成り立つ時に実行する処理
処理 2	条件が成り立たない時に実行する処理

※ELSE 以下は省略可能。

※ENDIF は処理 1・処理 2 が複数行ある時に使う。1 行だけの時は省略可能。

IF STTM==0 THEN @MAINLOOP

は、STTM が 0 と等しい、つまりタッチされた時間が 0（タッチされていない）だったら、@MAINLOOP へもどる、ということです。全然タッチされていなかったら、その後の処理をしてもしょうがないからです。

タッチされていた時、つまり STTM が 0 でない時は、次の行へ実行が移ります。

IF PX<0 OR PX>4 THEN @MAINLOOP

IF PY<0 OR PY>4 THEN @MAINLOOP

は、パネルをタッチしているかどうかを判断しています。

PX と PY を計算した結果、0 より小さかったり、4 より大きかったりしたら、そもそもタッチした場所がパネルを外れているので、@MAINLOOP へもどります。

「**OR**」は「**～または～**」で、2 つの条件のどちらかが成り立てばいい、という条件式です。

もう一つ、「**AND**」は「**～かつ～**」で、2 つの条件の両方が成り立つ、という条件式です。

【参考情報】

「～かつ～」「～または～」は、中学の数学の「集合」で習います。

そしてパネル表示サブルーチンで、タッチされたパネルの表示を変えます。

```
47 '=== パネル ひょうじ ===
48 @PRTPANEL
49
50 DISPLAY 1
51  FOR Y=0 TO 4
52   FOR X=0 TO 4
53    N=P[X,Y]
54    S=X+Y*5
55    GX=80+32*X
56    GY=32+32*Y
57    SPSET S,48+N
58    SPOFS S,GX,GY
59    SPSCALE S,2,2
60    SPSHOW S
61    IF T[X,Y]==1 THEN GFILL GX,GY,
GX+31,GY+31,RGB(128,128,255)
62   NEXT
63 NEXT
64 RETURN
```

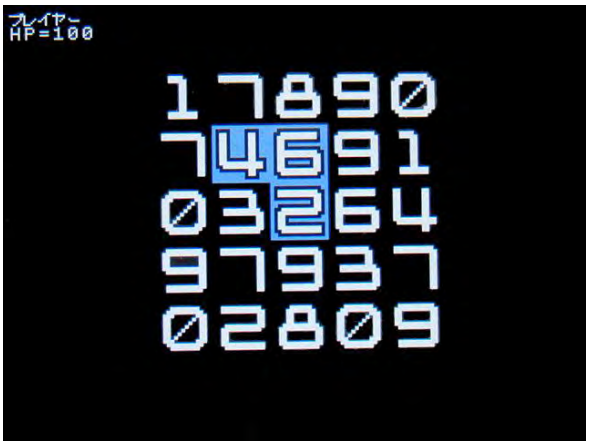
スプライト番号を S に設定

パネルのグラフィック座標を GX,GY に設定

変数を使った指定に変更

タッチ中のパネルは、背景に水色の四角を描く

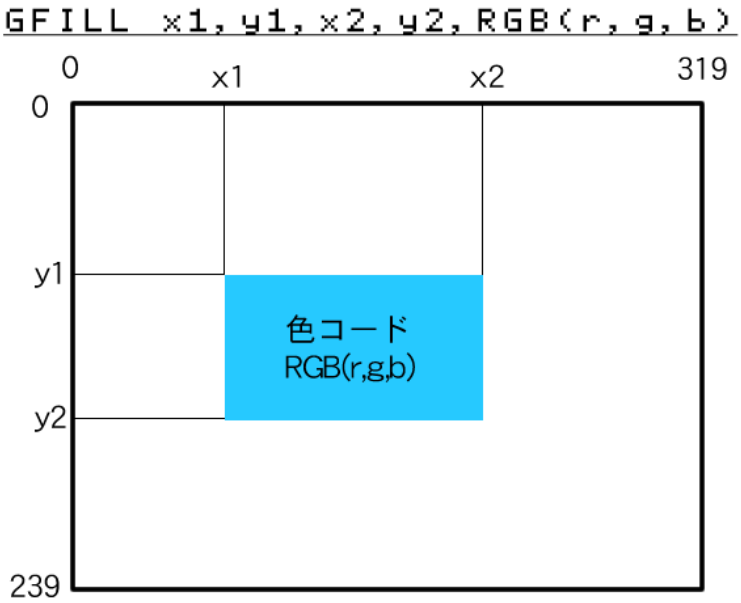
プログラムを実行してみましょう。
パネルをタッチすると、水色の色がつきます。



パネルの色つけは、GFILL（ジーフィル）命令でグラフィックの四角を描いています。

```
GFILL   GX      ,GY      ,GX+31    ,GY+31    RGB(128,128,255)
        x 座標 1  y 座標 1  x 座標 2  y 座標 2  色コード
```

RGB 関数の中の色コードは、赤（red）、緑（green）、青（blue）（光の 3 原色）の数値を、それぞれ 0～255 の値で指定します。
それぞれの数字を変えると、色が変わります。いろいろ試してみてください。



このままだと、ペンを画面からはなしても、パネルに色がついたままです。
ペンをはなしたら色が消えて元にもどるようにしてみましょう。

```
21 '=== メイン ループ ===
22 @MAINLOOP
23
24 TOUCH OUT STTM,TX,TY
25 IF STTM==0 THEN @MAINLOOP
26
27 'タッチ ちゅう
28 WHILE STTM>0
29  PX=(TX-80)/32
30  PY=(TY-32)/32
31  PX=FLOOR(PX)
32  PY=FLOOR(PY)
33  IF PX<0 OR PX>4 THEN BREAK
34  IF PY<0 OR PY>4 THEN BREAK
35  T[PX,PY]=1
36  GOSUB @PRTPANEL
37  TOUCH OUT STTM,TX,TY
38 WEND
```

STTM が 0 以上（タッチ中）の間はくり返し

PX,PY がパネルの範囲を外れていたら、くり返しをやめる

パネルをタッチ状態にする

パネルを再表示

タッチされているか確認

くり返し

```
39
40 'タッチ クリア
41  FOR Y=0 TO 4
42   FOR X=0 TO 4
43    T[X,Y]=0
44   NEXT
45 NEXT
46 GOSUB @PRTPANEL
47
48 GOTO @MAINLOOP

   （中略）

60 '=== パネル ひょうじ ===
61 @PRTPANEL

   （中略）

72    SPSCALE S,2,2
73    SPSHOW S
74    IF T[X,Y]==0 THEN
75     GFILL GX,GY,GX+31,GY+31,RGB(0,
0,0)
76    ELSE
77     GFILL GX,GY,GX+31,GY+31,RGB(1
28,128,255)
78    ENDIF
79   NEXT
80 NEXT
81 RETURN
```

タッチが終了したので、パネルをタッチしていない状態にもどして再表示

タッチされていないパネルは、背景を黒表示

タッチされているパネルは、背景を水色表示

プログラムを実行してみましょう。
ペンで数字パネルにタッチすると色がつき、ペンをはなすと色が消えて元にもどります。

★WHILE～WEND

新しく「WHILE」（ホワイル）～「WEND」（ダブリューエンド）命令が出てきました。
これは、「**ある条件が成り立っている間だけくり返す**」という命令です。

```
WHILE 条件式
 （プログラム）
WEND
```

で、条件式が成り立っている間だけ、WHILE と WEND の間のプログラムをくり返します。
今回のプログラムは、条件式を「STTM>0」として、TOUCH 命令で読み取ったタッチ時間が 0 以上（＝タッチされている）の間はくり返す、というプログラムになっています。

【参考情報】

くり返し命令は、これまで使って来た「FOR」～「NEXT」もあります。
「FOR」～「NEXT」は、「**くり返しの回数があらかじめ決まっている場合**」、
「WHILE」～「WEND」は、「**何回くり返すかわからない場合**」に使います。

★IF～THEN～ELSE～ENDIF

パネル表示のサブルーチンでは、タッチ中のパネルとタッチしていないパネルを、IF 命令で場合分けして表示しています。
IF 命令では、条件式が成り立つ時・成り立たない時に実行するプログラムを、それぞれ何行も書くことができます。

```
IF 条件式 THEN
 （条件式が成り立つ時に実行するプログラム）
    :
ELSE
 （条件式が成り立たない時に実行するプログラム）
    :
ENDIF
```

プログラムを「PUZZLE6」の名前で保存（SAVE）しましょう。