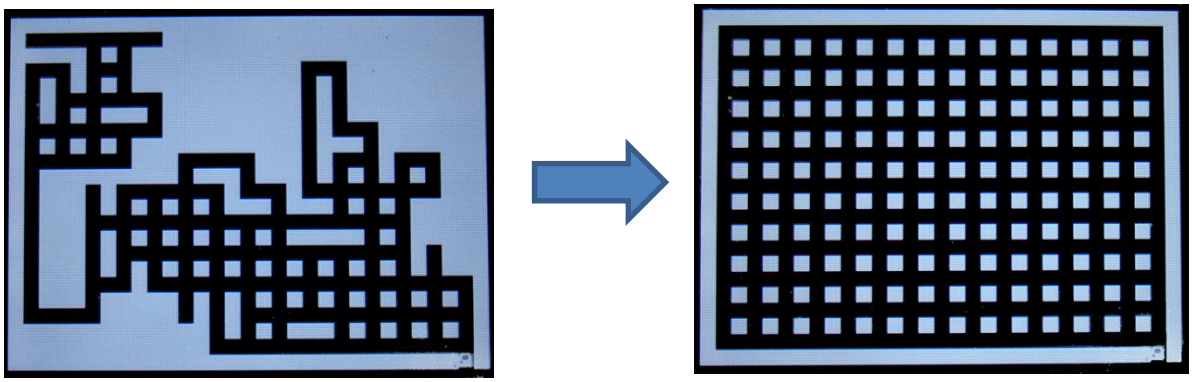
プチコンでダンジョンゲームを作る（２）

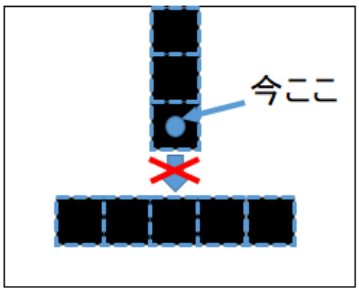
●他の道とつながらないように迷路を掘る

前回は、碁盤の目のように道を掘る所まで、プログラムを作りました。



このプログラムは、道を掘るときに何も考えずにどんどん掘ってしまうので、道が全部つながって碁盤の目になってしまいます。

これを防ぐには、「これから掘る先がかべなのか？道なのか？」を判断しなければいけません。
例えば右の図のように、上から道を掘ってきていて、その下にすでに道があったとします。
ここから下へ道を掘ってしまうと、下の道とつながってしまいますから、そちらへ掘ってはいけません。
つまり、掘る前に、掘る先の迷路の状態をチェックして、かべなのか？道なのか？を判断する必要があります。



この先、下へ道を掘ると
下の道とつながってしまう
→そちらへは掘らない

迷路の状態をチェックするにはいくつか方法が考えられますが、ここでは迷路全体の情報を**マップ配列変数**に記憶させて、それをチェックすることにします。

まず、最初のプログラムタイトルを「DANJON2」にしましょう。

```
1 '*** DANJON2 ***
2
```

プログラムの最初の方で、迷路全体を記憶するマップ配列変数 M を設定します。

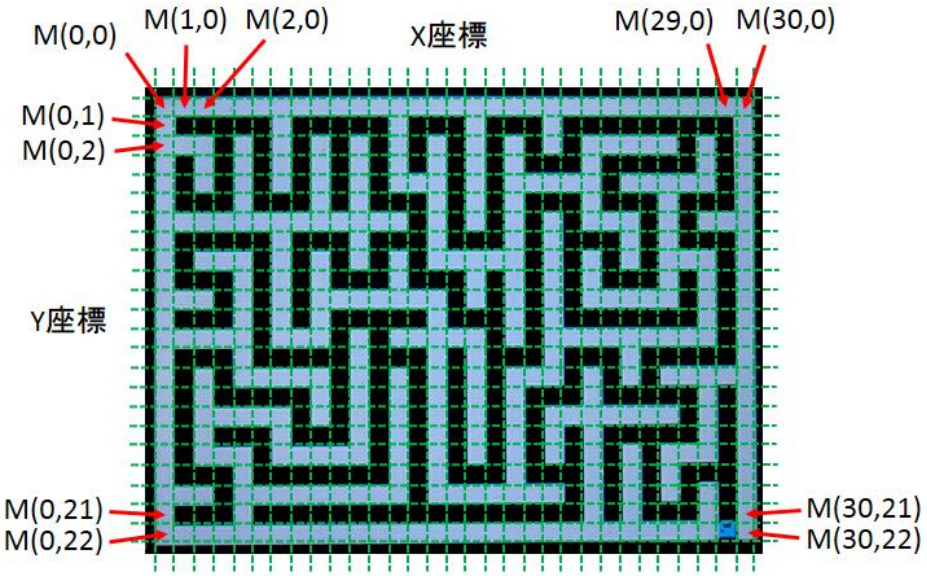
```
14 '--- ﾒｲﾛ ﾂｸﾘ ---
15
16 DIM M(30,22)
17 PNLTYPE "OFF"
18 FOR MY=0 TO 22
19 PNLSTR 0,MY,"■"*31
20 NEXT MY
```

マップ配列変数 M を設定

迷路の大きさが、横（x 座標）が 0～30、縦（y 座標）が 0～22 まであるので、その大きさの **2 次元配列変数 M(30,22)**を設定します。
M(0,0)から M(30,22)まで、31×23＝713 個の配列変数ということになります。

変数の値は、
- 0＝かべ
- 1＝道

という設定にしましょう。
配列変数は、設定した直後は値が全て「0」なので、最初は全部「かべ」ということになります。



※プチコンでは、配列変数は 1 次元「A(x)」、2 次元「A(x,y)」が使えます。

まず、スタート地点（1,1）に道を掘る所で、スタート地点のマップ配列変数 M(1,1)を 1（道）にします。

```
22 'ｽﾀｰﾄ ﾁﾃﾝ
23 PNLSTR 1,1,"  "
24 M(1,1)=1
25
```

スタート地点の M(1,1)を 1（道）にする

そして、道を掘るプログラムで、道になった部分のマップ配列変数を「1」（道）にします。

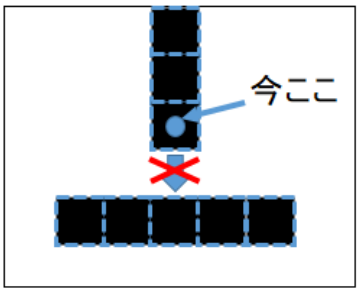
```
39 'アナホリ
40 @ANAHORI
   …（中略）…
47 X=X+DX(D)
48 Y=Y+DY(D)
49 PNLSTR X,Y,"  "
50 M(X,Y)=1
51 X=X+DX(D)
52 Y=Y+DY(D)
53 PNLSTR X,Y,"  "
54 M(X,Y)=1
55
56 GOTO @ANAHORI
```

道を掘った所の M(X,Y)を 1（道）にする

道を掘った所の M(X,Y)を 1（道）にする

これで、現在の迷路全体の「かべ」か「道」かの状態を M(X,Y)に記憶させることができます。

さて、このマップ配列変数 M(X,Y)を使って、道を掘る時のチェックをします。
これから道を掘る先のマップ配列変数をチェックして、もし道（1）だったら、そちらへは道を掘らないようにします。



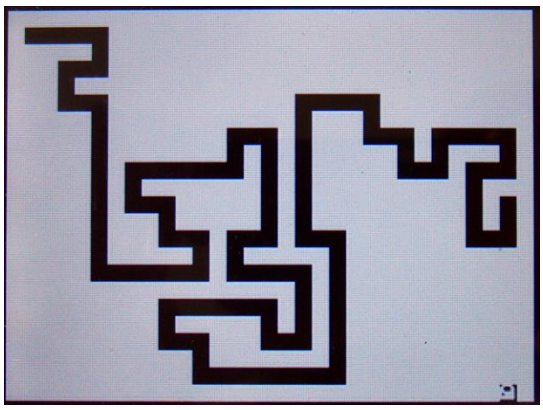
```
39 'ｱﾅﾎﾘ
40 @ANAHORI
41 D=RND(4)+1
42 IF X==1 AND D==3 THEN @ANAHORI
43 IF X==29 AND D==4 THEN @ANAHORI
44 IF Y==1 AND D==1 THEN @ANAHORI
45 IF Y==21 AND D==2 THEN @ANAHORI
46
47 X2=X+DX(D)*2
48 Y2=Y+DY(D)*2
49 IF M(X2,Y2)==1 THEN @ANAHORI
50
51 X=X+DX(D)
52 Y=Y+DY(D)
53 PNLSTR X,Y,"  "
54 M(X,Y)=1
```

2つ掘った先の座標(X2,Y2)を計算

マップ配列変数 M(X2,Y2)が 1（道）だったらそちらへは道を掘れないので、@ANAHORI へもどって、方向の乱数 D を出し直す

道を掘る先（2 つ先）の座標を、新しい変数 X2,Y2 で計算して、その座標のマップ配列変数 M(X2,Y2)をチェックします。「1」（道）だったらそちらへは掘れないので、元へもどして違う方向へ掘るようにします。

プログラムを実行してみましょう。
他の道とはつながらないように、ちゃんと迷路を掘っていきます。
ただし、どちらにも道を掘れなくて行き止まりになると、そこでプログラムが止まってしまいます。



これを防ぐには、どこかで**分かれ道**を作らないといけません。つまり、

- 道を掘りながら、分かれ道が作れるポイントを記録しておく。
- 行き止まりになったら、記録していたポイントへ戻って、分かれ道を掘る。

という手順が必要になります。

ここで、「DANJON2」の名前で、プログラムを保存（SAVE）してください。

●道が掘れる方向数を記録する

分かれ道を作るために、道を掘りながら、そのポイントで何方向に道が作れるかを記録しておくようにします。
まず、プログラムのタイトルを「DANJON3」に変更します。

```
1 '*** DANJON3 ***
2
```

分かれ道ポイントはたくさんできるので、配列変数を使って記録することにします。
迷路作りの最初で、記録用の変数を設定します。

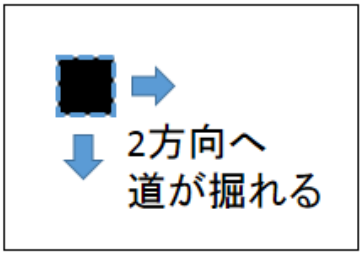
```
14 '--- メイロ ツクリ ---
15
16 DIM M(30,22)
17 DIM C(200),CX(200),CY(200)
18 CP=0
19 PNLTYPE "OFF"
```

分かれ道ポイント用の配列変数を設定

分かれ道ポイントの数を設定

- C(200)……分かれ道ポイントの、各ポイントで掘れる方向の数（0～3）を記録。
- CX(200)…分かれ道ポイントの x 座標。
- CY(200)…分かれ道ポイントの y 座標。
- CP……分かれ道ポイントの数。最初は 0。

例えば、最初のスタート地点(1,1)を考えると、道が掘れる方向が下と右の 2 方向あるので、分かれ道ポイントの 1 個目として数えられます。
なので、以下のようにプログラムに書きます。




```
24 'スタート チテン
25 PNLSTR 1,1,"  "
26 M(1,1)=1
27 CP=1
28 C(1)=2
29 CX(1)=1
30 CY(1)=1
31 PNLSTR 1,1,"2"
32
33 'ゴール チテン
```

分かれ道ポイントの数を 1 にする

1 個目の分かれ道ポイント配列の値を 2（方向）にする

1 個目の分かれ道ポイントの座標を(1,1)にする

確認用に、分かれ道ポイントの値を表示する

27 行目で、分かれ道ポイントの数 CP を 1 にします。
28 行目で、分かれ道ポイント配列の 1 個目 C(1)に、道を掘れる方向の数「2」を記録します。
29～30 行目で、1 個目の分かれ道ポイントの x 座標，y 座標を CX(1)，CY(1)に記録します。
31 行目は、プログラムの動作を確認するために、分かれ道ポイントの値（ここでは「2」）を画面に表示します。（この行はあとで消します）

プログラムを実行してみましょう。
スタート地点に「2」が表示されます。

次に、道を掘っていくに従って、分かれ道ポイントをどんどん更新しないといけません。
例えばスタート地点の分かれ道ポイントの値は最初は「2」ですが、下か右のどちらかに道を掘った後は、1 減らして「1」にしないといけません。（残りは 1 方向しか掘れないので）
道を掘る部分のプログラムで、掘る前の位置の分かれ道ポイントの値 C(CP)を 1 減らします。

```
54 X2=X+DX(D)*2
55 Y2=Y+DY(D)*2
56 IF  M(X2,Y2)==1  THEN  @ANAHORI
57
58 C(CP)=C(CP)-1
59 PNLSTR  X,Y,STR$(C(CP))
60 X=X+DX(D)
61 Y=Y+DY(D)
62 PNLSTR  X,Y,"  "
63 M(X,Y)=1
```

分かれ道ポイントの値を 1 減らす

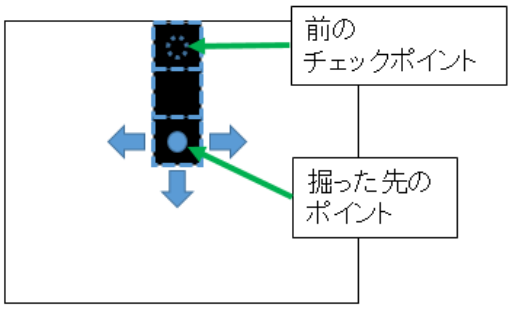
確認用に、C(CP)の値を表示する

なお、分かれ道ポイントの値を下パネルに表示するのに、文字変換関数 **STR$（ストリング関数）**を使っています。()内の変数を文字型に変換する関数です。あとで消す行でもあるので、詳しくは説明しません。

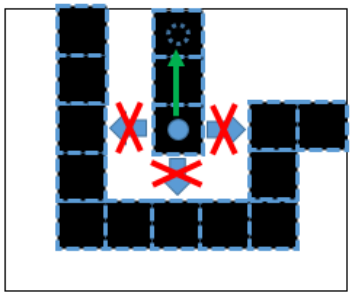
そして、道を掘った先のポイントで、道を掘れる方向をチェックします。その結果で、

- 掘れる方向が 1 以上（行き止まりではない）→新しい分かれ道ポイントとして追加
- 掘れる方向が 0（行き止まり）→前の分かれ道ポイントへもどり、方向チェックを続ける

という処理をします。



掘った先が行き止まりではない
→新しいチェックポイントと
して追加



掘った先が行き止まり
→前のチェックポイントへ
もどって、チェックし直し

```
66 PNLSTR X,Y,"  "
67 M(X,Y)=1
68
69 'ホレル ホウコウ チェック
70 CP=CP+1
71 @CCHECK
72 CH=0
73 FOR D=1 TO 4
74 X3=X+DX(D)*2
75 Y3=Y+DY(D)*2
76 IF X3<1 OR X3>29 THEN @DNEXT
77 IF Y3<1 OR Y3>21 THEN @DNEXT
78 IF M(X3,Y3)==0 THEN CH=CH+1
79 @DNEXT
80 NEXT D
81
82 PNLSTR X,Y,STR$(CH)
83 IF CH==0 THEN CP=CP-1:X=CX(CP):Y
=CY(CP):GOTO @CCHECK
84
85 C(CP)=CH
86 CX(CP)=X
87 CY(CP)=Y
88
89 @MATI
90 WAIT 1
91 IF BTRIG()!=16 THEN @MATI
92 GOTO @ANAHORI
```

分かれ道ポイント数を 1 つ先へ更新

掘れる方向を数える変数 CH を 0 に

4 方向でループ

2 つ先の座標(X3,Y3)を計算

迷路の外へは掘れないので次へ

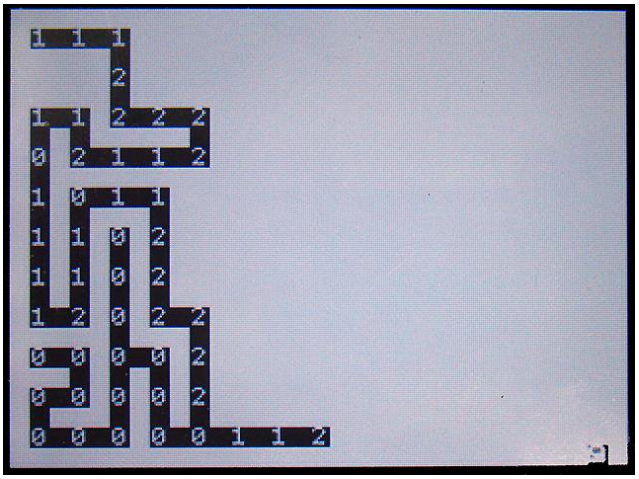
2 つ先がかべだったら、掘れる方向 CH を 1 増やす

CH が 0 なら行き止まりなので、1 つ前の分かれ道ポイントに戻って、チェックを続ける

行き止まりでなければ、掘った先を新しい分かれ道ポイントとして追加

A ボタンを押したら1つずつ進むようにする

プログラムを実行してみましょう。
A ボタンを押すと、一歩ずつ道が掘られていきます。
行き止まりになると、前の分かれ道ポイントに戻って、分かれ道が作られます。
分かれ道ポイントの値が表示されるので、それを見ながら動作を確認してください。



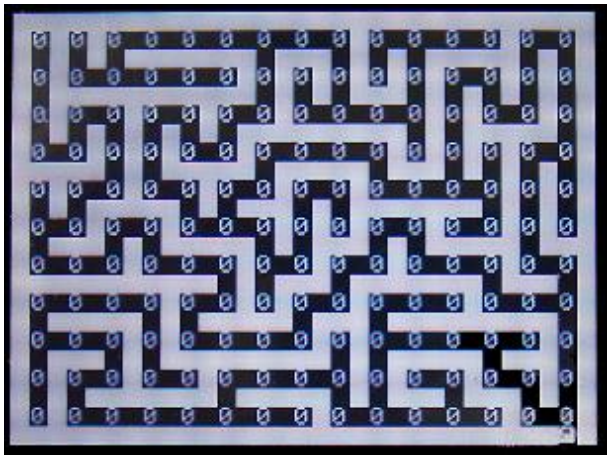
このままだと、迷路が全部掘れた所で、エラーになってしまいます。
分かれ道ポイントの数 CP が減らされて、マイナスの値になってしまうからです。
方向チェックの最初で CP の値をチェックして、もし 0 だったら迷路作りを終わりにします。

```
69 'ホレル ホウコウ チェック
70 CP=CP+1
71 @CCHECK
72 IF CP==0 THEN @ANAHORIEND
73 CH=0
74 FOR D=1 TO 4
 …(中略)…
90 @MATI
91 WAIT 1
92 IF BTRIG()!=16 THEN @MATI
93 GOTO @ANAHORI
94
95 @ANAHORIEND
96 WAIT 1
97 IF BTRIG()!=16 THEN @ANAHORIEND
```

CP が 0 なら、迷路完成なのでループの外へジャンプ

迷路が完成したら、A ボタンを押すまで待つ

これで、迷路が完成した様子を見られます。
迷路が完成したら、全部の分かれ道ポイントの値が 0 になるはずです。
A ボタンを押すと、プログラムが終了します。



このプログラムを、「DANJON3」の名前で保存（SAVE）しましょう。

●迷路を改良する

何度かプログラムを動かして迷路を作るとわかりますが、スタート地点から道をたどると、分かれ道があまり無くて、ほぼ一本道でゴールまで行ける迷路ができてしまうことがあります。
「スタート地点から掘れるだけ道を掘って、行き止まりになったら分かれ道を作る」というプログラムになっているからです。
これをさけるにはいろいろな方法が考えられますが、簡単なのは「逆に、ゴール地点から迷路を掘っていく」方法です。
プログラムを改造してみましょう。

まず、プログラムのタイトルを「DANJON4」に変更します。

```
1 '*** DANJON4 ***
2
```

迷路を掘り始める部分を改造して、ゴール地点から道を掘るプログラムにしてみましょう。

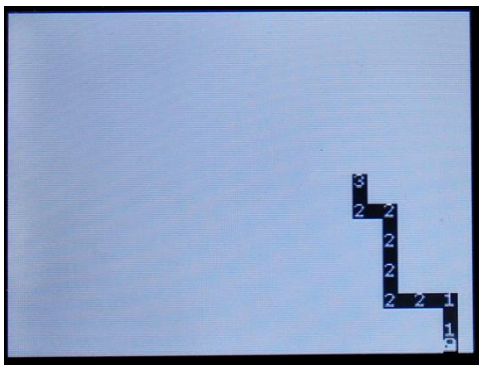
```
24 'スタート チテン
25 PNLSTR 29,21," "
26 M(29,21)=1
27 CP=1
28 C(1)=2
29 CX(29)=1
30 CY(21)=1
31 PNLSTR 29,21,"2"
32
33 'ゴール チテン
34 PNLSTR 29,22,"■"
35
36 X=29
37 Y=21
38
39 'ファホリ ジュンビ
```

ゴール地点を道にする

ゴール地点を分かれ道ポイントとして記録

“掘る人”の座標をゴール地点にセット

プログラムを実行してみましょう。
ゴール地点から迷路を掘っていきます。
何度か迷路を作ってみましょう。
スタート地点から辿ると、前のプログラムより正解が難しくなっていると思います。



これで、迷路作りのプログラムが一通り完成しました。
最後に、チェック用のプログラムの行の先頭に「’」（アポストロフィー）をつけて、コメントにして実行しないようにしましょう。

- 分かれ道ポイントの数値を表示している PNLSTR 命令の行
- 1 段階ずつ待つための BUTTON 命令などの行

```
29 CX(29)=1
30 CY(21)=1
31 'PNLSTR 29,21,"2"
32
```

行の先頭に「’」を付ける

```
58 C(CP)=C(CP)-1
59 'PNLSTR X,Y,STR$(C(CP))
60 X=X+DX(D)
61 Y=Y+DY(D)
```

行の先頭に「’」を付ける

```
81 NEXT D
82
83 'PNLSTR X,Y,STR$(CH)
84 IF CH==0 THEN CP=CP-1:X=CX(CP):
Y=CY(CP):GOTO @CCHECK
```

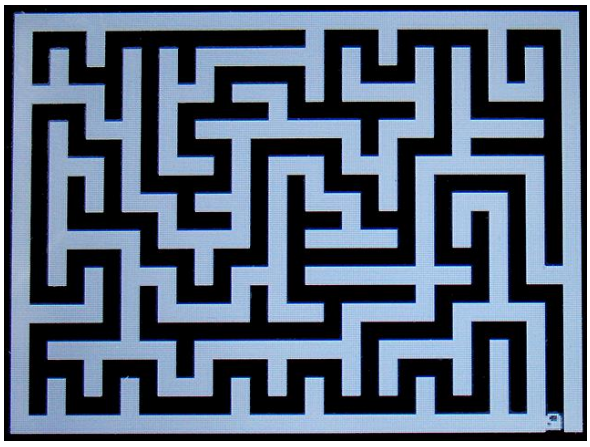
行の先頭に「’」を付ける

```
90 '@MATI
91 'WAIT 1
92 'IF BTRIG()!=16 THEN @MATI
93 GOTO @ANAHORI
```

行の先頭に「’」を付ける

行を消さずに「’」を付けてコメントにするのは、もしプログラムがうまく動かない時に、「’」を消して再びチェックできるようにするためです。

プログラムを実行してみましょう。
迷路が 1 秒もかからずにできます。



このプログラムを、「DANJON4」の名前で保存（SAVE）しましょう。

●迷路にプレイヤーを表示する

迷路に自分のキャラクター（プレイヤー）を表示して、動かしてみましょう。

まず、プログラムのタイトルを
「DANJON5」に変更します。

```
1 '*** DANJON5 ***
2
```

「@ANAHORIEND」の後の行を書き換えて、プレイヤーのキャラクターを表示します。
キャラクターは人型の「人」を使います。

```
 95 @ANAHORIEND
 96
 97 PX=1
 98 PY=1
 99
100 '--- ｹﾞｰﾑ ﾙｰﾌﾟ ---
101 @GAMELOOP
102
103 PNLSTR PX,PY,"人"
104
105 GOTO @GAMELOOP
```

プレイヤーのx座標を PX、y 座標を PY で設定
最初はスタート地点（1,1）

(PX,PY)の位置にプレイヤーのキャラクターを表示

あとでぐるぐるループしてゲームの処理をするので、「ゲームループ」を作ります。
今はとりあえずプレイヤーのキャラクターを表示するだけです。

プログラムを実行してみましょう。
スタート地点にプレイヤーが表示されます。

●プレイヤーを動かす

表示したプレイヤーのキャラクターを、十字キーで上下左右に動かしてみましょう。
十字キーの入力を読み取るには、「BUTTON（ボタン）関数」を使います。
以下のように、BUTTON 関数のプログラムを、ゲームループの中に追加します。

```
100 '--- ｹﾞｰﾑ ﾙｰﾌﾟ ---
101 @GAMELOOP
102
103 B=BUTTON()
104 LOCATE 27,0
105 PRINT B;"  "
106 PNLSTR PX,PY,"ｷ"
107
108 GOTO @GAMELOOP
```

BUTTON 関数の値を B に入れる

B の値を上画面の右上に表示

プログラムを実行してみましょう。
今回はとりあえずテストということで、BUTTON 関数の値を上画面の右上に表示します。

最初は「0」が表示されるはずです。
DS の十字キーやボタンをいろいろ押してみましょう。値が変化するはずです。

BUTTON 関数の値は、押されたキーやボタンによって、以下の値になります。

押されたボタン	値
何も押さない	0
十字キーの上	1
十字キーの下	2
十字キーの左	4
十字キーの右	8
A ボタン	16
B ボタン	32
X ボタン	64
Y ボタン	128
L ボタン	256
R ボタン	512
START ボタン	1024

なお、2 つのボタンを同時に押すと、それぞれの値を足し算した値になります。
例えば「十字キーの上」を押しながら「A ボタン」を押すと、1＋16＝17 になります。
十字キーやボタンをいろいろ押して、表の値になることを確認してください。

十字キーの上下左右を押すと、BUTTON 関数の値が「上＝1」「下＝2」「左＝4」「右＝8」となります。
それをチェックして、プレイヤーの座標(PX,PY)を変えます。
数字を表示している部分を書き換えます。

```
100 '--- ｹﾞｰﾑ ﾙｰﾌﾟ ---
101 @GAMELOOP
102
103 B=BUTTON()
104 PNLSTR PX,PY,"  "
105 IF B==1 THEN PY=PY-1
106 IF B==2 THEN PY=PY+1
107 IF B==4 THEN PX=PX-1
108 IF B==8 THEN PX=PX+1
109 PNLSTR PX,PY,"夫"
110
111 GOTO @GAMELOOP
```

プレイヤーを一度消す

B の値によって、プレイヤーの座標を変える

プログラムを実行してみましょう。十字キーを押すと、プレイヤーが上下左右に動きます。
ただし、あまりに動きが速いのと、かべのチェックやはみだしチェックをしていないので、迷路をつきぬけて画面の外へ飛び出してしまいます。
まず、動きを遅くしてみましょう。GOTO 命令でループの最初へ戻る前に、WAIT 命令の時間待ちを入れます。

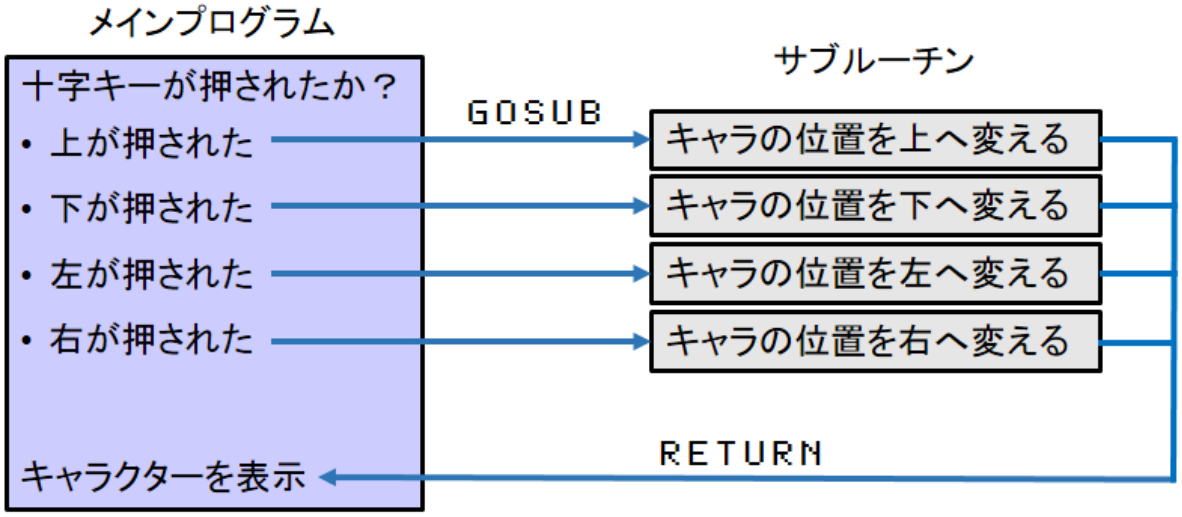
```
100 '--- ｹﾞｰﾑ ﾙｰﾌﾟ ---
101 @GAMELOOP
102
103 B=BUTTON()
104 PNLSTR PX,PY,"  "
105 IF B==1 THEN PY=PY-1
106 IF B==2 THEN PY=PY+1
107 IF B==4 THEN PX=PX-1
108 IF B==8 THEN PX=PX+1
109 PNLSTR PX,PY,"夫"
110
111 WAIT 1
112 GOTO @GAMELOOP
```

時間待ちを入れる（1/60 秒）

こうすると、目で追える程度の速さになります。

次に、迷路の外へはみ出さないための「はみ出しチェック」と、迷路の道だけを歩くように「行き先が道かどうかのチェック」をします。

判断が複雑になるので、上下左右それぞれのプログラムを**サブルーチン**にします。
サブルーチンとは、メインのプログラムから呼び出して処理をして、終わるとまた戻ってくるプログラムのことです。



メインプログラムからサブルーチンを呼び出すには「GOSUB」（ゴーサブ）命令、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。
GOSUB 命令、RETURN 命令の文法は、以下のとおりです。

```
GOSUB          @UE
               ジャンプ先の
               サブルーチンのラベル
```

サブルーチンへジャンプします。

```
RETURN
               なし
```

メインルーチンへ戻ります。

GOSUB～RETURN を使って、上下左右へ移動するプログラムをサブルーチンにします。

```
100 '--- ｹﾞｰﾑ ﾙｰﾌﾟ ---
101 @GAMELOOP
102
103 B=BUTTON()
104 PNLSTR PX,PY,"  "
105 IF B==1 THEN GOSUB @UE
106 IF B==2 THEN GOSUB @SHITA
107 IF B==4 THEN GOSUB @HIDARI
108 IF B==8 THEN GOSUB @MIGI
109 PNLSTR PX,PY,"主"
110
111 WAIT 1
112 GOTO @GAMELOOP
113
114 'ｲﾄﾞｳ ｳｴ
115 @UE
116 PY=PY-1
117 RETURN
118
119 'ｲﾄﾞｳ ｼﾀ
120 @SHITA
121 PY=PY+1
122 RETURN
123
124 'ｲﾄﾞｳ ﾋﾀﾞﾘ
125 @HIDARI
126 PX=PX-1
127 RETURN
128
129 'ｲﾄﾞｳ ﾐｷﾞ
130 @MIGI
131 PX=PX+1
132 RETURN
```

上下左右のサブルーチンを呼び出す

上へ移動するサブルーチン

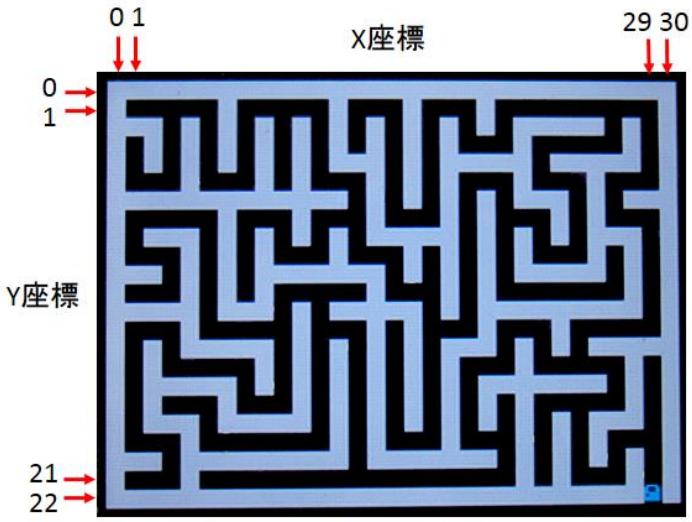
下へ移動するサブルーチン

左へ移動するサブルーチン

右へ移動するサブルーチン

プログラムを実行してみましょう。見た目の動作はこれまでと変わりません。

それぞれのサブルーチンで、まずはみ出しチェックをします。
迷路を掘る時のチェックと同様に、座標 PX、PY の値で判断します。
迷路の道の範囲は、
　X座標＝1～29
　Y座標＝1～21
なので、そこから外へ出ないようにします。
IF 命令の条件判断の行を追加します。




```
114 'ｲﾄﾞｳ　ｳｴ
115 @UE
116 IF PY<=1 THEN RETURN
117 PY=PY-1
118 RETURN
119
120 'ｲﾄﾞｳ　ｼﾀ
121 @SHITA
122 IF PY>=21 THEN RETURN
123 PY=PY+1
124 RETURN
125
126 'ｲﾄﾞｳ　ﾋﾀﾞﾘ
127 @HIDARI
128 IF PX<=1 THEN RETURN
129 PX=PX-1
130 RETURN
131
132 'ｲﾄﾞｳ　ﾐｷﾞ
133 @MIGI
134 IF PX>=29 THEN RETURN
135 PX=PX+1
136 RETURN
```

プレイヤーの y 座標 PY が 1 以下だったら、上へ行かない

プレイヤーの y 座標 PY が 21 以上だったら、下へ行かない

プレイヤーの x 座標 PX が 1 以下だったら、左へ行かない

プレイヤーの x 座標 PX が 29 以上だったら、右へ行かない

これで、キャラクターが迷路からはみ出さないようになります。

次に、行き先がかべかどうかチェックして、かべだったらその方向へは進まないようにします。
迷路の状態は、マップ配列変数 M(x,y)でチェックして、0 だったらかべだと判断します。

```
114 ｲﾄﾞｳ ｳｴ
115 @UE
116 IF PY<=1 THEN RETURN
117 IF M(PX,PY-1)==0 THEN RETURN
118 PY=PY-1
119 RETURN
120
121 ｲﾄﾞｳ ｼﾀ
122 @SHITA
123 IF PY>=21 THEN RETURN
124 IF M(PX,PY+1)==0 THEN RETURN
125 PY=PY+1
126 RETURN
127
128 ｲﾄﾞｳ ﾋﾀﾞﾘ
129 @HIDARI
130 IF PX<=1 THEN RETURN
131 IF M(PX-1,PY)==0 THEN RETURN
132 PX=PX-1
133 RETURN
134
135 ｲﾄﾞｳ ﾐｷﾞ
136 @MIGI
137 IF PX>=29 THEN RETURN
138 IF M(PX+1,PY)==0 THEN RETURN
139 PX=PX+1
140 RETURN
```

迷路の 1 つ上をチェック
かべだったら上へ行かずにもどる

迷路の 1 つ下をチェック
かべだったら下へ行かずにもどる

迷路の 1 つ左をチェック
かべだったら左へ行かずにもどる

迷路の 1 つ右をチェック
かべだったら右へ行かずにもどる

これで迷路の道のところだけキャラクターが動くようになります。
キャラクターを操作して、ゴールまで行ってみましょう。

今のままだと、キャラクターの動きが速すぎて、なかなかうまく操作できないと思います。
こんな時は、十字キーの状態を読み取る BUTTON 関数を、「BTRIG」（ビートリガー）関数に変えてみましょう。

```
100 '--- ｹﾞｰﾑ ﾙｰﾌﾟ ---
101 @GAMELOOP
102
103 B=BTRIG()
104 PNLSTR PX,PY,"  "
```

BTRIG 関数は BUTTON 関数と似ていますが、十字キーやボタンを押しっぱなしにしても、最初の 1 回しか入力したことになりません。
今回の場合、例えば十字キーを押しっぱなしにしても、一歩ずつしかキャラクターが進まなくなります。

これで、キャラクターを動かすプログラムは完成です。
プログラムを「DANJON5」の名前で保存（SAVE）してください。

★できる人は

ここまでのプログラムでも、ちょっと改造すれば簡単なゲームができます。
例えば、スタートからゴールへたどりつくまでの時間を測って、タイムアタックするゲームが作れます。考えてみてください。

```
  1 '--- ｼﾞｶﾝﾖ ﾊｶﾙ ｻﾝﾌﾟﾙ ---
  2 ACLS
  3 TMREAD(TIME$),H0,M0,S0
  4
  5 @LOOP
  6 TMREAD(TIME$),H1,M1,S1
  7 S=S1-S0
  8 C=0
  9 IF S<0 THEN C=1:S=S+60
 10 M=M1-M0-C
 11 IF M<0 THEN M=M+60
 12 LOCATE 0,0
 13 PRINT M;":";S;"  "
 14 GOTO @LOOP
```