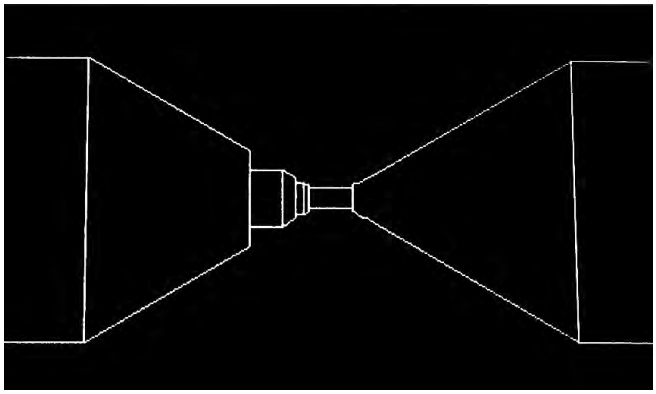
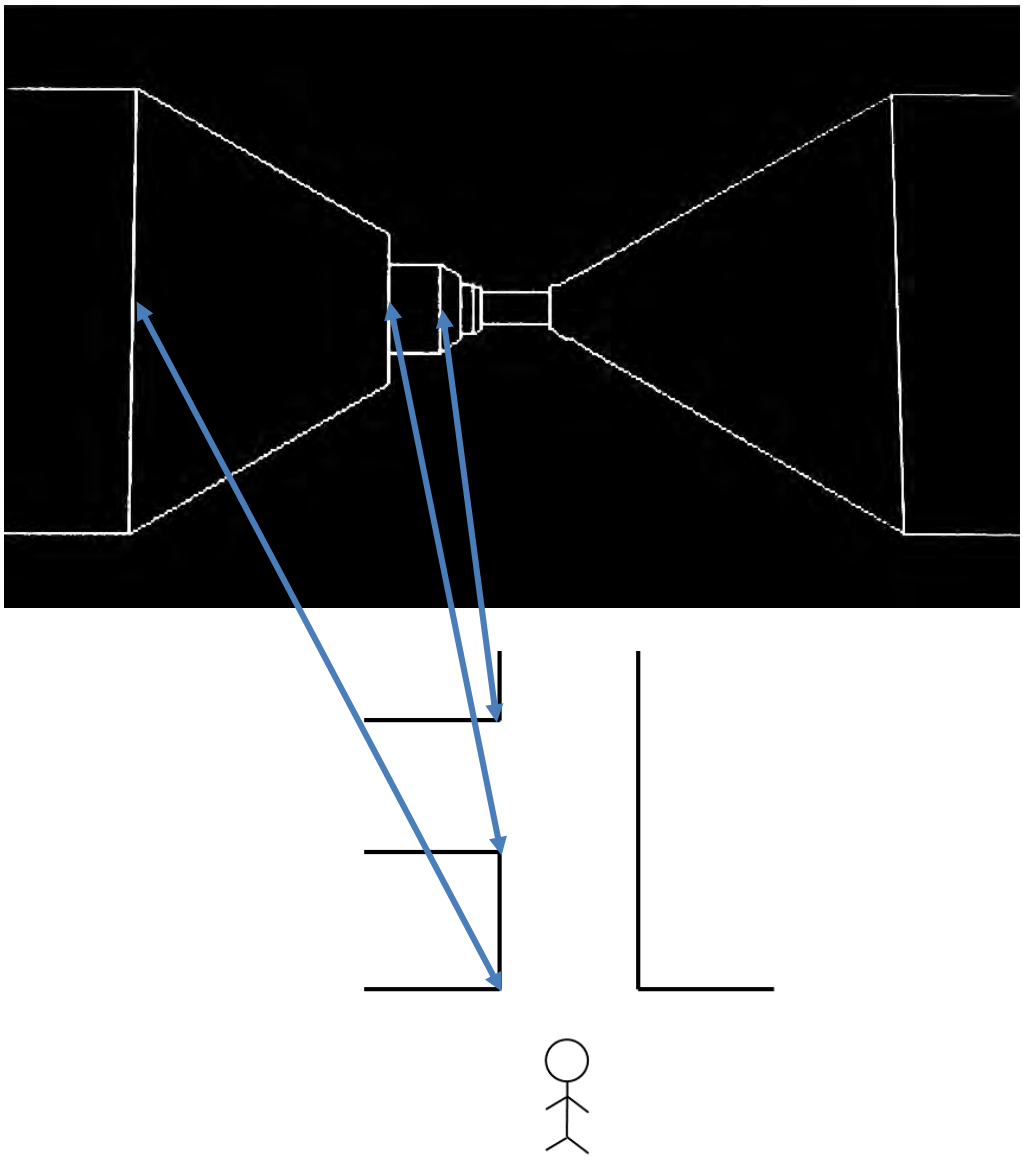
プチコンで 3D ダンジョンゲームを作る（３）

●迷路の 3D 表示の方法



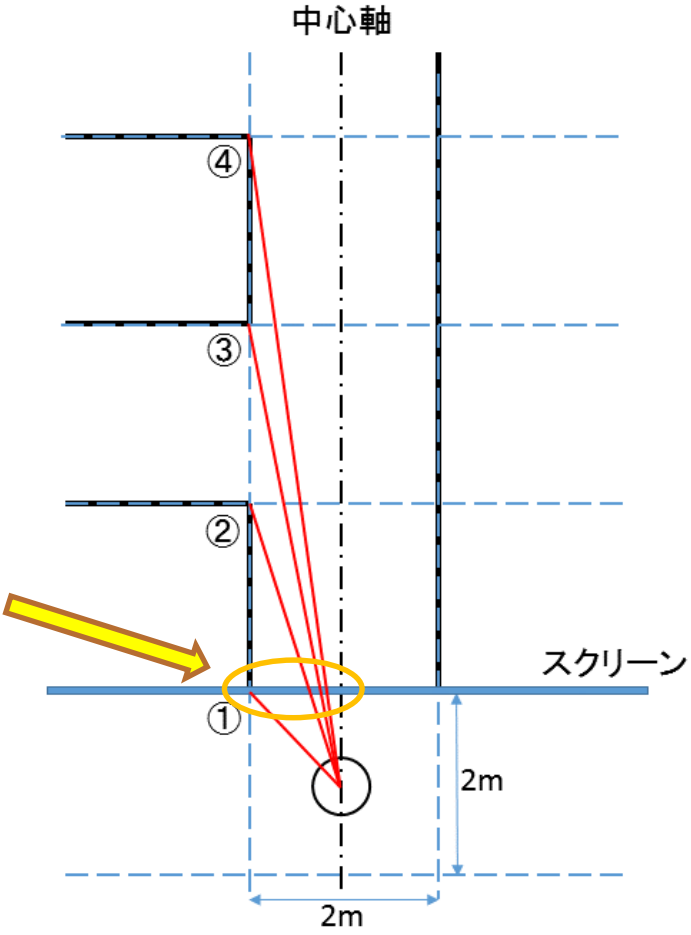
前回までで、下画面に迷路を作ってキャラクターを表示しました。
いよいよ上画面に迷路を 3D 表示してみましょう。

まず、キャラクターの視点から見て、迷路がどう見えるか？を考えます。



キャラクターから見て左側のかべと道の見え方を考えると、1番手前の角は一番左側に、2 番目の角はその右に、3 番目の角はその右に…となっています。
（右側も同じように考えればいいので省略）

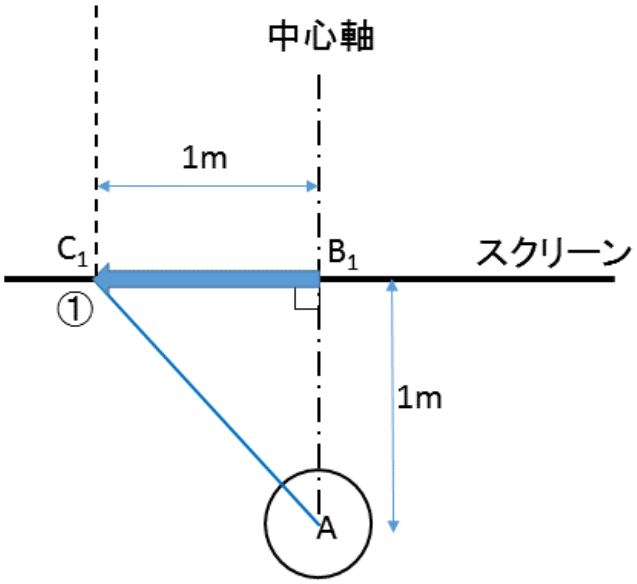
もう少し細かく、画面での表示位置を考えてみましょう。



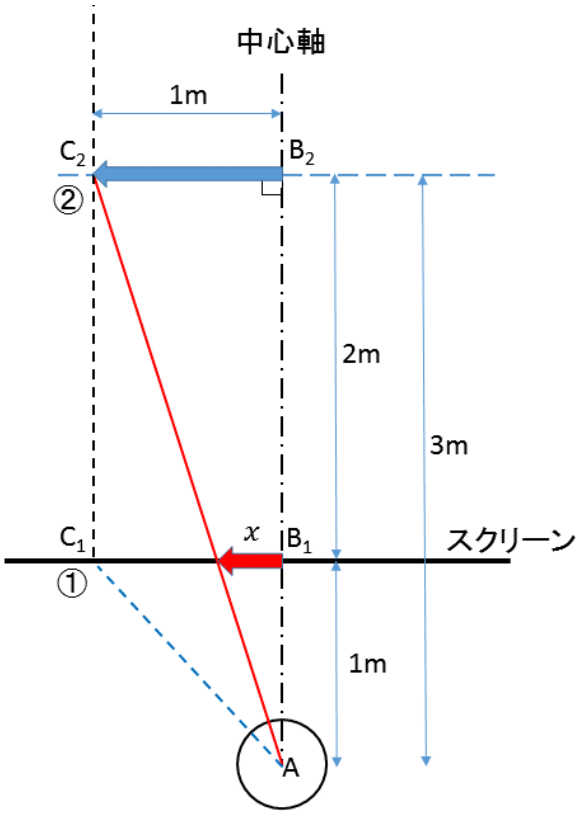
一番手前の角を①、二番目の角を②、三番目の角を③…として、①の角の位置にスクリーンを立てます。
スクリーン上で、それぞれの角がどの位置に見えるかを考えます。

また、わかりやすくするために道幅を 2m として、迷路全体をひとマス2m の方眼紙にのせて考えます。

まず 1 番目の角①を、中心軸からどのくらいの距離に表示するかです。
全体をひとマス 2m の方眼紙にのせて考えると、角①と中心軸との距離 B_1-C_1 (青矢印)は 1m です。
プレイヤーとスクリーンの距離 A-B_1 も 1m なので、三角形 A-B_1-C_1 は直角二等辺三角形になります。

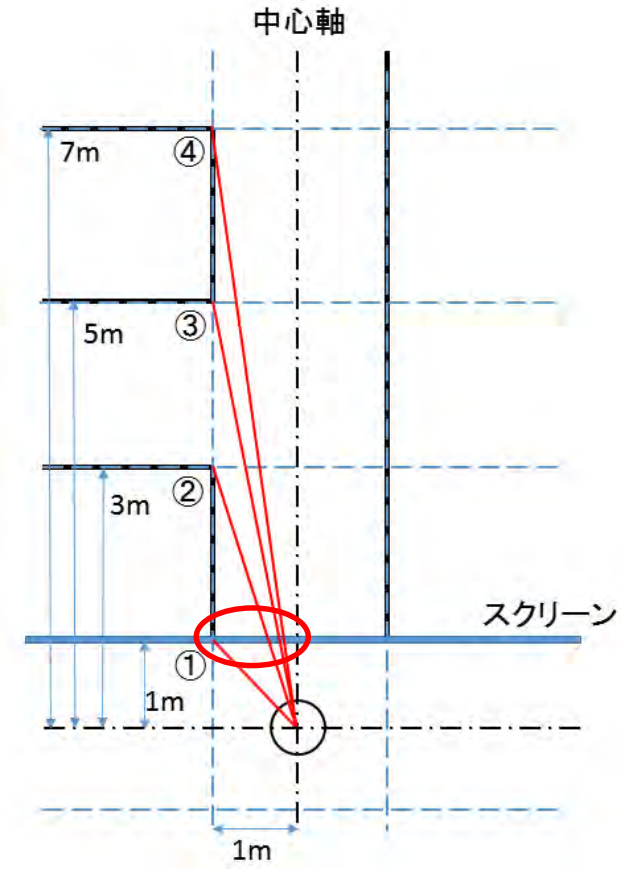


次に 2 番目の角②の表示位置を考えます。
角②は角①から 2m 向こうにあるので、プレイヤーから見ると 3m の距離になります。その分スクリーン上では、中心軸からの距離は短く見えます。(赤矢印)
青矢印の長さ(B_2-C_2、1m)と、赤矢印の長さ(x)の間には、プレイヤーからの距離との比例関係(3:1)が成り立ちます。
よって赤矢印は、青矢印の 3 分の 1 の長さ→$\frac{1}{3}$m になります。



同じように、3 番目の角③、4 番目の角④…の表示位置を考えると、やはりプレイヤーからの距離との比例関係になります。

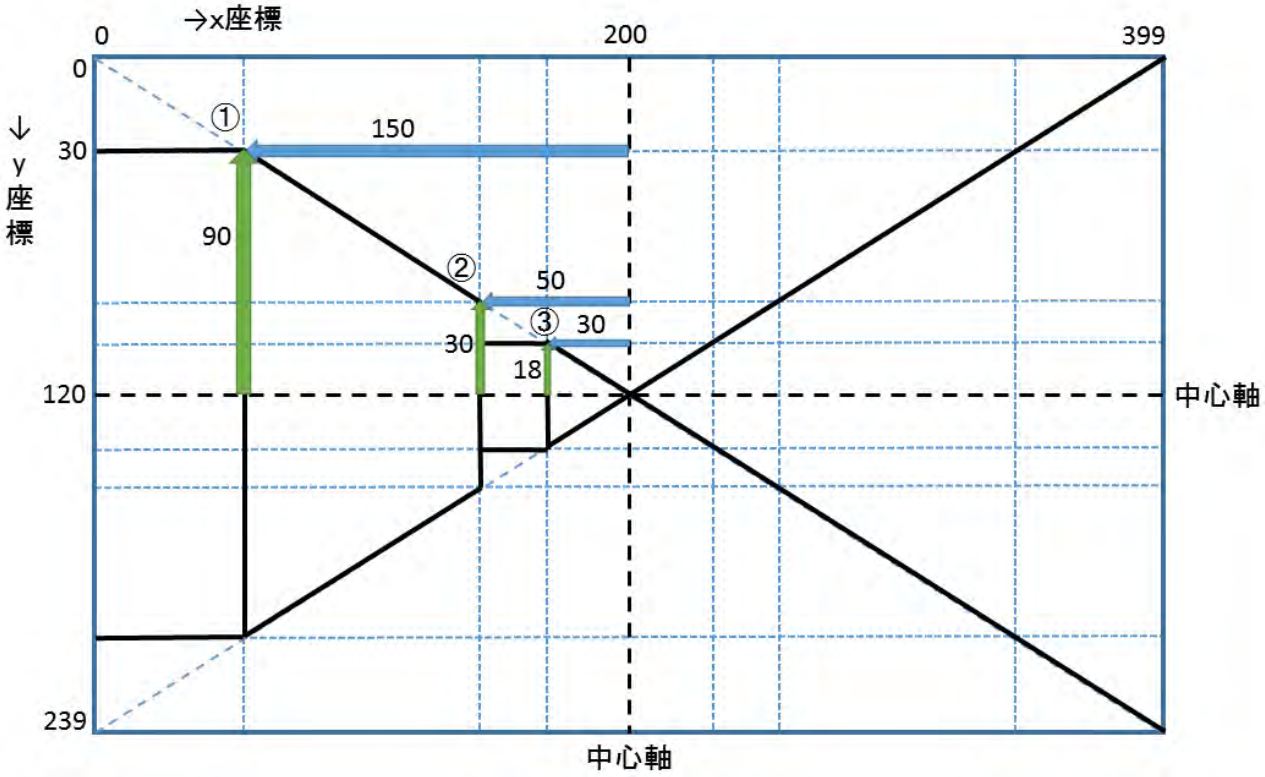
	プレイヤーからの距離	スクリーン中心軸からの表示距離
角③	5m	$\frac{1}{5}$m
角④	7m	$\frac{1}{7}$m
:	:	:
角n	$2\times n-1$	$\frac{1}{2\times n-1}$



以上の表示位置は、プレイヤーから見て右側も同様です。

また、ここまではプレイヤーから見た左右方向の表示位置を考えましたが、上下方向(天井ー床方向)も同様に考えます。

これまでの理論をふまえて、上画面での表示レイアウトを考えます。



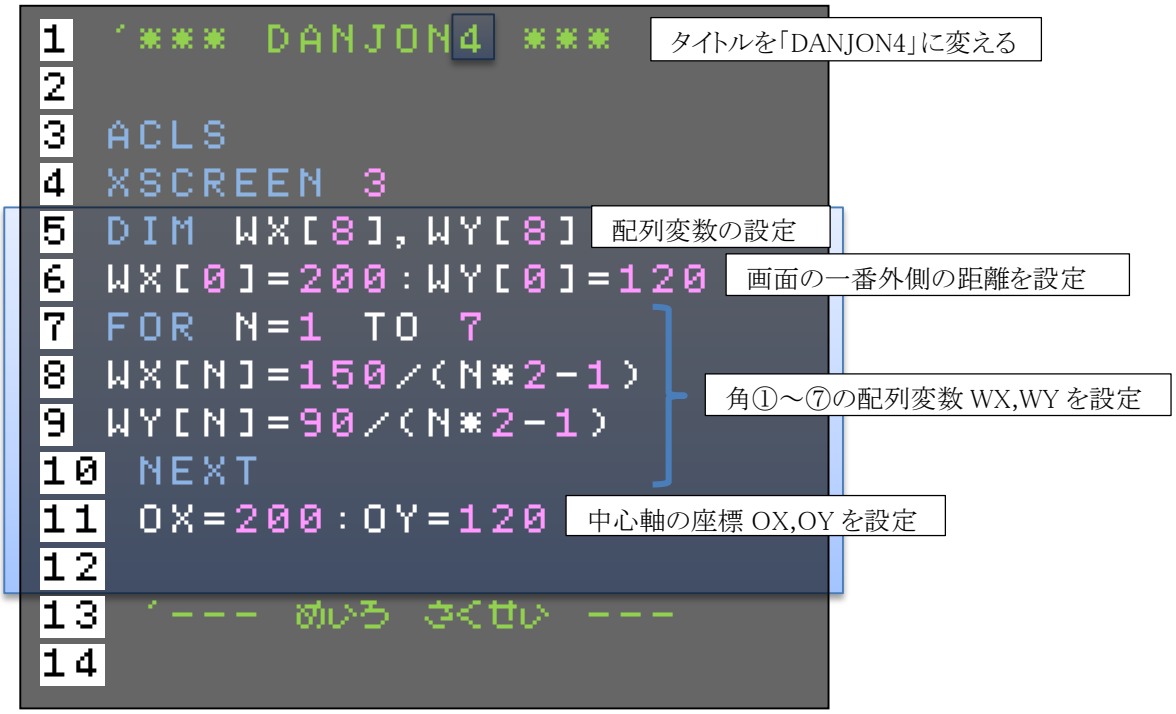
上画面は横 400 ドット×縦 240 ドットなので、中心軸は横(x座標)が 200 ドット、縦(y座標)が 120 ドットの場所になります。
一番手前の角①の位置(中心軸からの距離)が全体の基準になりますが、今回は横(x)方向の距離を 150 ドット、縦(y)方向の距離を 90 ドットにします。
そうすると 2 番目以降の角の位置は、以下のようになります。

	x方向の中心軸からの距離(ドット)	y方向の中心軸からの距離(ドット)
角②	$150 \times \frac{1}{3} = 50$	$90 \times \frac{1}{3} = 30$
角③	$150 \times \frac{1}{5} = 30$	$90 \times \frac{1}{5} = 18$
角④	$150 \times \frac{1}{7} = 21.4 \ldots$	$90 \times \frac{1}{7} = 12.8 \ldots$
:	:	:
角n	$150 \times \frac{1}{2 \times n - 1}$	$90 \times \frac{1}{2 \times n - 1}$

これでそれぞれの角の表示位置が決まりました。

●斜め線を表示する

それでは、迷路を 3D 表示するプログラムを作ります。
まず最初の初期設定部分を変更します。



先に決めた、それぞれの角の中心軸からの距離を、**配列（はいれつ）変数** WX,WY に設定します。
配列変数は、「[ ]」（中かっこ）で番号が付いた変数です。

例　A[0] A[1] A[2] …それぞれ別の変数として扱われる。

配列変数を使うには、まず **DIM**（ディム、ディメンジョン）命令で、配列の個数を指定します。

```
DIM WX[8] ,WY[8]  …
      配列の個数   配列の個数
```

配列の個数	配列変数として使う個数を指定する。 配列の番号は「0」から始まるので、最大番号は「個数-1」になる。 例えば、WX[8]と指定すると、WX[0]～WX[7]が使える。

配列の個数は、今回は 8 個（番号 0～7）にしています。
0 番は画面の一番外側を指定するのに使い、7 つの角を1～7 番の7個で指定します。
（角⑧以降は画面に表示せずに省略します）
7～10 行目で、FOR～NEXT のくり返しを使って、WX[1]～WX[7]、WY[1]～WY[7]の値を指定しています。

次に、プレイヤーを動かすプログラムで、3D 迷路表示のサブルーチンを呼ぶようにします。

```
47 '--- プレイヤーをうごかす ---

  (中略)

58 B=BUTTON(2)
59 DISPLAY 1
60 IF B==1 AND PY>1 AND CHKCHR(PX,
PY-1)==0 THEN GOSUB @UP

  (中略)

70 LOCATE 34,29
71 PRINT M;":";S;" ";
72
73 GOSUB @3DMAZE
74
75 VSYNC 1
76 IF PX==37 AND PY==27 THEN GOTO
 @GOAL
77 GOTO @PLAYERMOVE
```

画面操作を下画面に切り替える

3D 迷路表示サブルーチンを呼ぶ

プログラムの最後に、3D 迷路表示のサブルーチンを追加します。
とりあえず、斜めのかべの線だけ表示します。

```
113 '--- 3D めいろ ひょうじ ---
114 @3DMAZE
115
116 DISPLAY 0
117 GPAGE 0,0
118 GCLS
119 GCOLOR RGB(255,255,255)

  (↓つづく↓)
```

画面操作を上画面に切り替え

グラフィックページ表示ページを 0、操作ページを 0 にする

グラフィック表示をクリアする

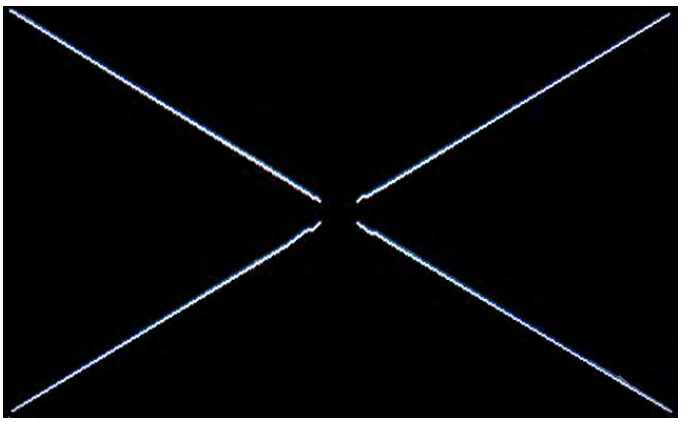
グラフィック描画色を白に

（↓つづき↓）

```
120 FOR N=0 TO 6
121 GLINE OX-WX[N],OY-WY[N],OX-WX[N+1],OY-WY[N+1]
122 GLINE OX-WX[N],OY+WY[N],OX-WX[N+1],OY+WY[N+1]
123 GLINE OX+WX[N],OY-WY[N],OX+WX[N+1],OY-WY[N+1]
124 GLINE OX+WX[N],OY+WY[N],OX+WX[N+1],OY+WY[N+1]
125 NEXT
126 RETURN
```

※121～124 行はほぼ同じなので、コピー（COPY）と貼り付け（PASTE）を使うといいでしょう。

プログラムを実行してみましょう。
上画面に、迷路のかべの斜め線が表示されます。



プログラムの中身を説明します。
117 行「GPAGE 0,0」は、画面に表示するグラフィックページと、線を引くなどの操作をするグラフィックページを指定します。

```
GPAGE    0       ,0
         表示ページ  操作ページ
```

表示ページ	画面に表示するグラフィックページ（0～5）。
操作ページ	操作をするグラフィックページ（0～5）。

ページ 4 はスプライト画像、ページ 5 は BG（背景）画像があらかじめ入っています。
今回は表示ページ・操作ページともページ 0 にしています。

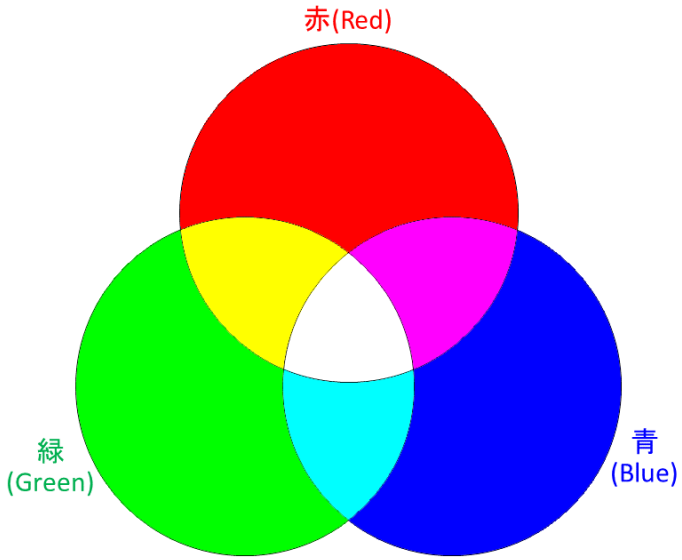
118 行「GCLS」は、グラフィック画面をクリアする命令です。

```
GCLS
```

119 行「GCOLOR RGB(255,255,255)」は、画面に描くグラフィックの色を指定する命令です。

```
GCOLOR RGB( 255 ,255 ,255)
           赤の強さ  緑の強さ  青の強さ
```

赤の強さ	0～255
緑の強さ	0～255
青の強さ	0～255



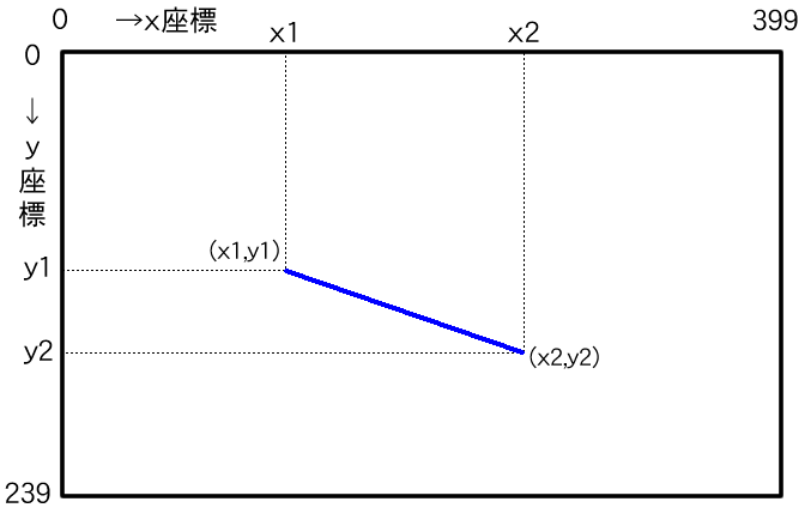
光の三原色、赤(Red)・緑(Green)・青(Blue)の強さを指定します。
(255,255,255)は白になります。

※他の色の例
黒(0,0,0)　赤(255,0,0)
緑(0,255,0)　青(0,0,255)
水色(80,208,255)　黄色(255,224,32)

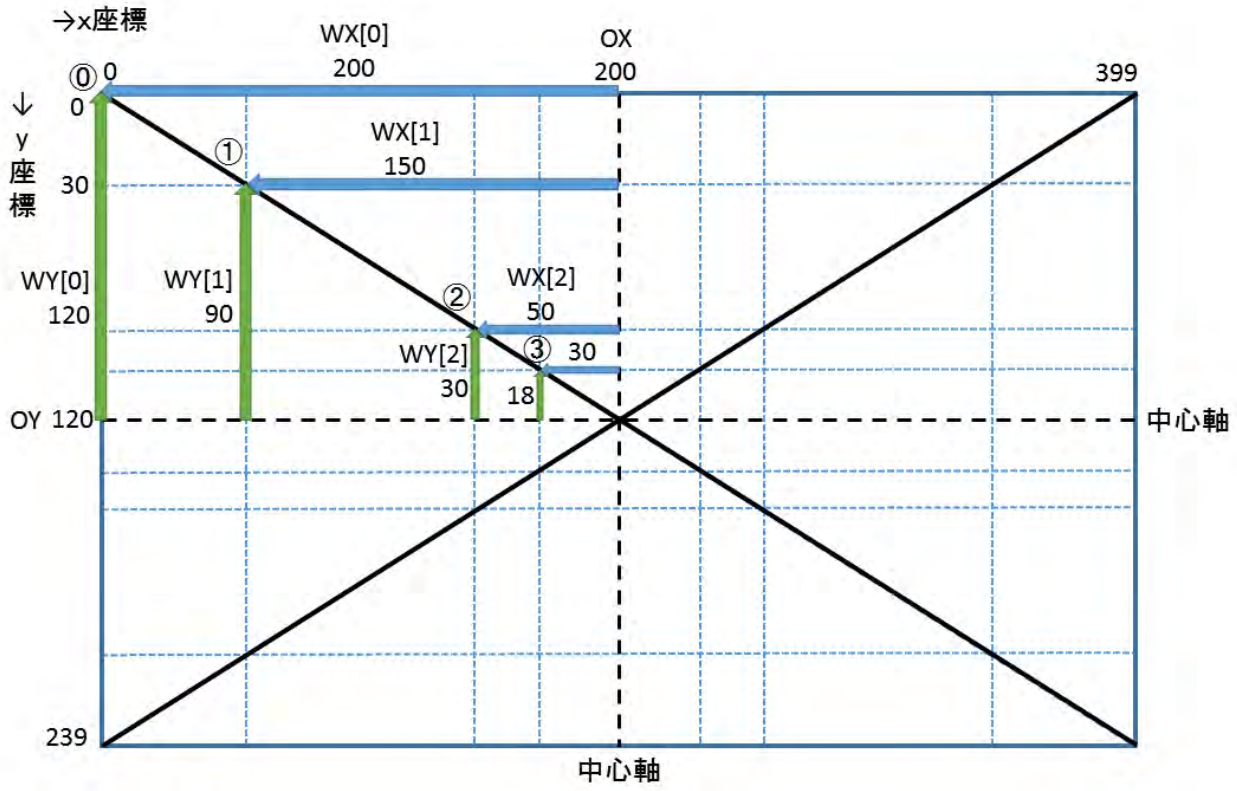
120～125 行は、角①～⑦までの斜め線を引いています。
グラフィックで線を引くには、GLINE(ジーライン)命令を使います。

```
GLINE  X1  ,Y1  ,X2  ,Y2  ,RGB(,,)
       x 座標 1  y 座標 1  x 座標 2  y 座標 2  色指定
```

x 座標 1,y 座標 1	線の始点の座標。
x 座標 2,y 座標 2	線の終点の座標。
色指定	線の色。RGB で指定できる。 省略された場合は GCOLOR などで指定された色。



今回は配列変数 WX,WY を使って、角①～⑦まで斜め線を引いています。

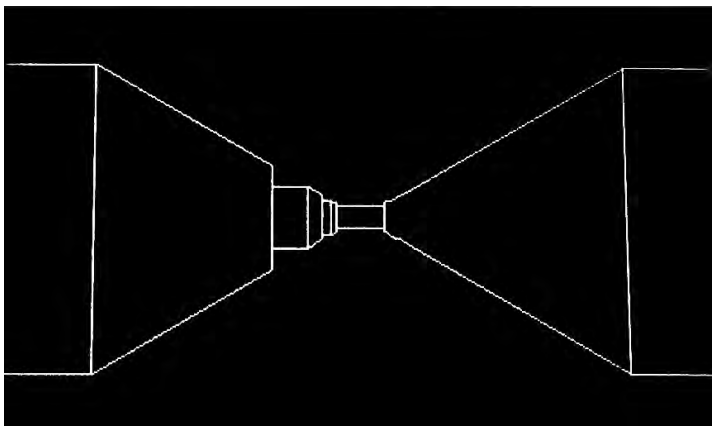


角⓪の座標 (OX-WX[0],OY-WY[0])
角① (OX-WX[1],OY-WY[1])
角② (OX-WX[2],OY-WY[2])
角③ (OX-WX[3],OY-WY[3])
：

プログラムを「DANJON4」の名前で保存しましょう。

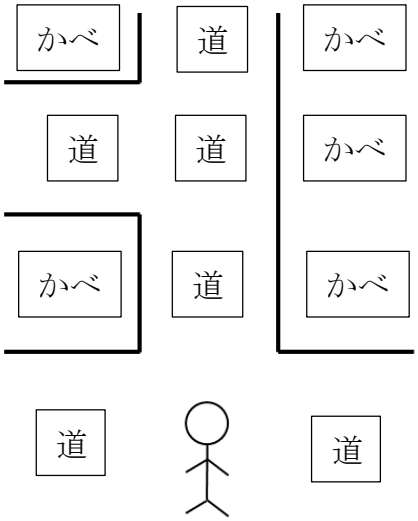
```
SAVE "DANJON4"
```

●道とかべを表示

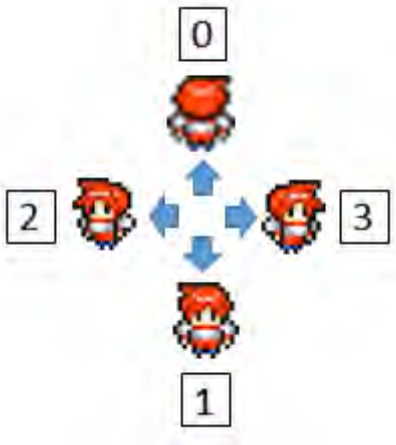


とりあえず斜め線を引きました。
実際は迷路のブロックが道か、かべかに従って、画面に道やかべを表示しなければいけません。

プレイヤーから見える迷路の各ブロックが、道かかべかを判断しながら3D 表示するプログラムを考えてみましょう。



まず、プレイヤーが迷路の中で、上下左右どちらへ向いているかを知る必要があります。方向がわからないと、そもそも迷路のどこを見て判断すればいいかわかりません。
新しい変数 PD を作って、プレイヤーの向き（上下左右）を 0～3 で表すことにします。



プレイヤーを動かすプログラムを改造して、プレイヤーの方向 PD を設定します。

```
1  '*** DANJON5 ***
```
タイトルを「DANJON5」に変える

```
   (中略)

47  '--- プレイヤーをうごかす ---
48
49  SPSET 0,496
50  PX=1
51  PY=1
52  PD=3
53  SPOFS 0,PX*8-4,PY*8-4
```
スタート時は右向き

```
90  'うえ
91  @UP
92  PY=PY-1
93  SPANIM 0,3,10,508,10,509,10,510,1
0,511,0
94  PD=0
95  RETURN
96
97  'した
98  @DOWN
99  PY=PY+1
100 SPANIM 0,3,10,500,10,501,10,502,
10,503,0
101 PD=1
102 RETURN
103
104 'ひだり
105 @LEFT
106 PX=PX-1
107 SPANIM 0,3,10,504,10,505,10,506,
10,507,0
108 PD=2
109 RETURN
  (↓つづく↓)
```
プレイヤーの方向を上向きに

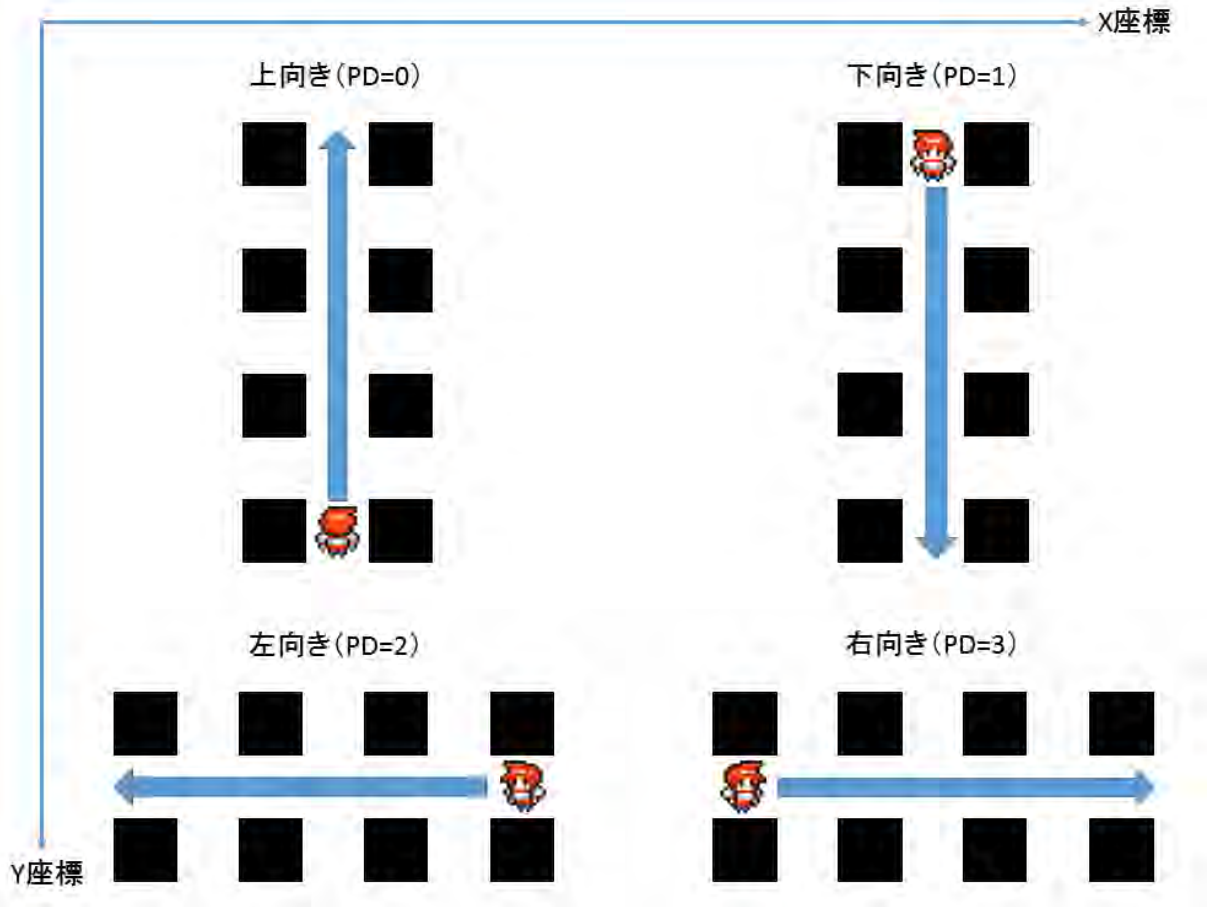
プレイヤーの方向を下向きに

プレイヤーの方向を左向きに

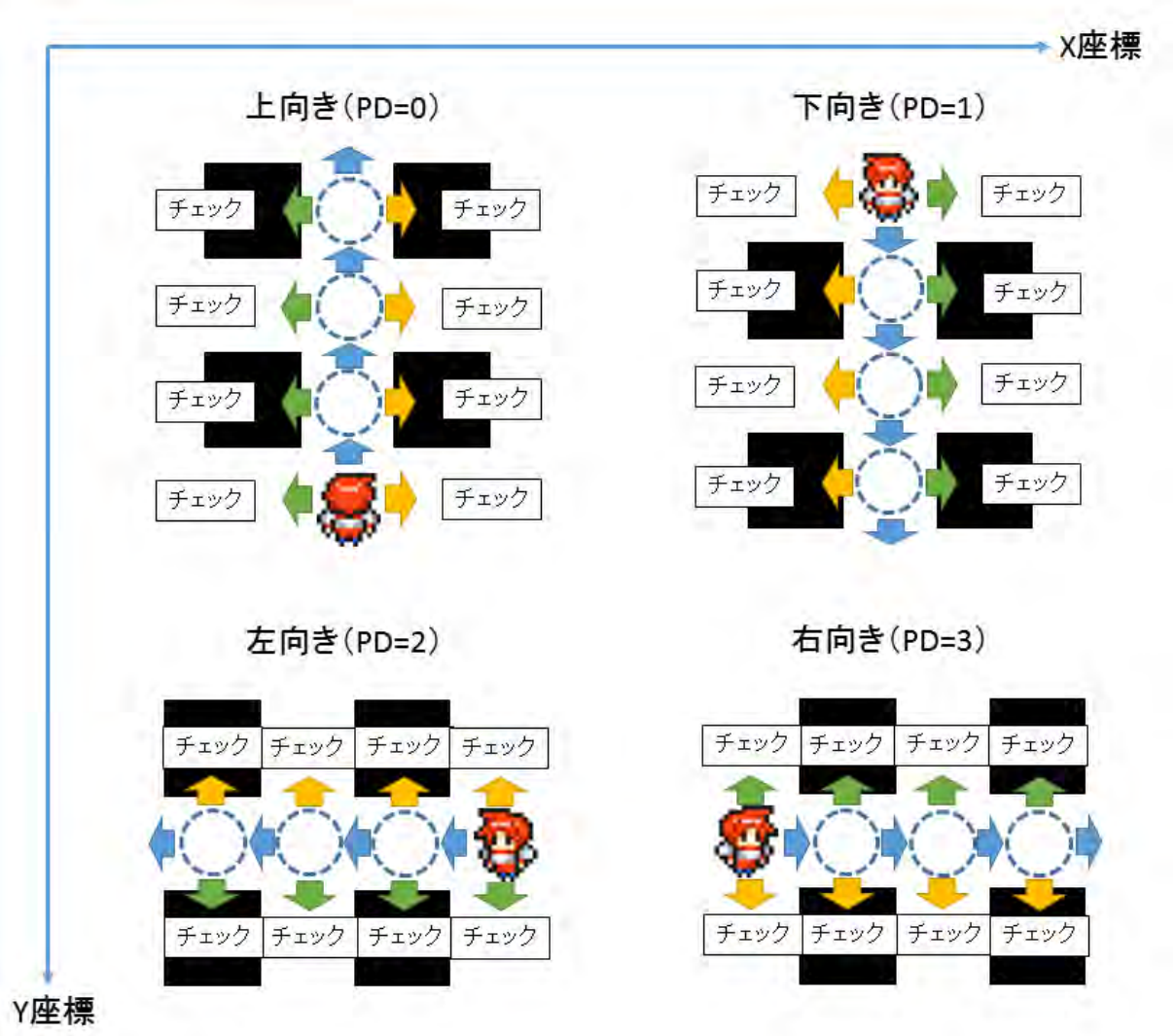
```
（↓つづき↓）
110
111  'みぎ
112  @RIGHT
113  PX=PX+1
114  SPANIM 0,3,10,496,10,497,10,498,
10,499,0
115  PD=3
116  RETURN
```

プレイヤーの方向を右向きに

これでプレイヤーの方向 PD が設定できたので、これを使って 3D 表示を考えます。



例えばプレイヤーが上向き（PD=0）の時は、迷路の上方向をチェックする必要があります。同様に下向き（PD=1）の時は迷路の下方向、左向き（PD=2）の時は左方向、右向き（PD=3）の時は右方向をチェックする必要があります。



さらに詳しく考えると、プレイヤーが上向き（PD=0）の時は、

- プレイヤーがいるブロックの左側・右側が「道」か「かべ」かをチェック
- プレイヤーの 1 つ上のブロックの左側・右側が「道」か「かべ」かをチェック
- プレイヤーの 2 つ上のブロックの左側・右側が「道」か「かべ」かをチェック
- ：

という処理が必要です。

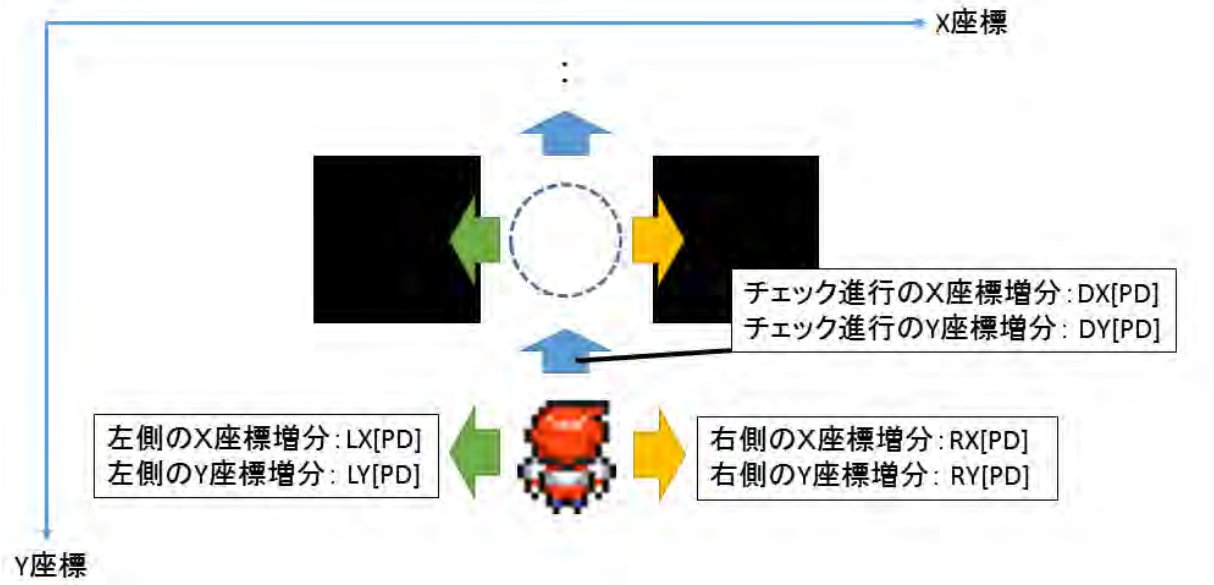
同じように、プレイヤーが下向き（PD=1）の時を考えると、

- プレイヤーがいるブロックの左側・右側が「道」か「かべ」かをチェック
- プレイヤーの 1 つ下のブロックの左側・右側が「道」か「かべ」かをチェック
- プレイヤーの 2 つ下のブロックの左側・右側が「道」か「かべ」かをチェック
- ：

となります。プレイヤーが上向きの時とは、チェック進行方向が逆になりますし、「ブロックの右側・左側」も逆になります。

つまり上下左右 4 方向それぞれで、チェック方向や「ブロックの右側・左側」の方向が変わることになります。わけがわかりませんね。どうすればいいでしょうか。

これを解決するために、チェック進行方向、チェック左側方向、チェック右側方向それぞれの X 座標・Y 座標の増分（増えたり減ったりする量）を、配列変数で管理することにします。



プログラムの最初の設定部分に、配列変数の設定を追加します。

```
1  '*** DANJON5 ***
  （中略）
11  OX=200:OY=120
12  DIM DX[4],DY[4]
13  DX[0]=0:DY[0]=-1
14  DX[1]=0:DY[1]=1
15  DX[2]=-1:DY[2]=0
16  DX[3]=1:DY[3]=0
17  DIM LX[4],LY[4]
18  LX[0]=-1:LY[0]=0
19  LX[1]=1:LY[1]=0
20  LX[2]=0:LY[2]=1
21  LX[3]=0:LY[3]=-1
22  DIM RX[4],RY[4]
23  RX[0]=1:RY[0]=0
24  RX[1]=-1:RY[1]=0
25  RX[2]=0:RY[2]=-1
26  RX[3]=0:RY[3]=1
27
28  '--- めいろ さくせい ---
```

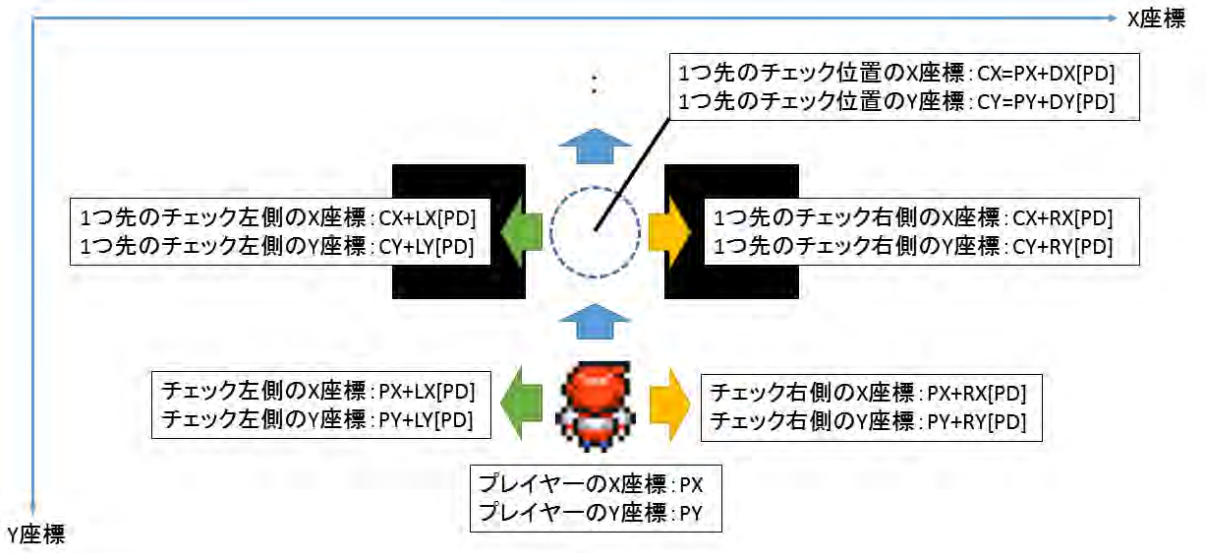
チェック進行方向の X 座標・Y 座標の増分 DX[],DY[]を設定

ブロック左側の X 座標・Y 座標の増分 LX[],LY[]を設定

ブロック右側の X 座標・Y 座標の増分 RX[],RY[]を設定

DX[],DY[],LX[],LY[],RX[],RY[]のそれぞれの値は、前ページの図のチェック方向を見ながら、確認してください。

このように配列変数を設定すると、チェック位置の座標が以下のように計算できます。
プレイヤーの向き(PD)が上下左右(0～3)に変化しても、この計算は変わりません。



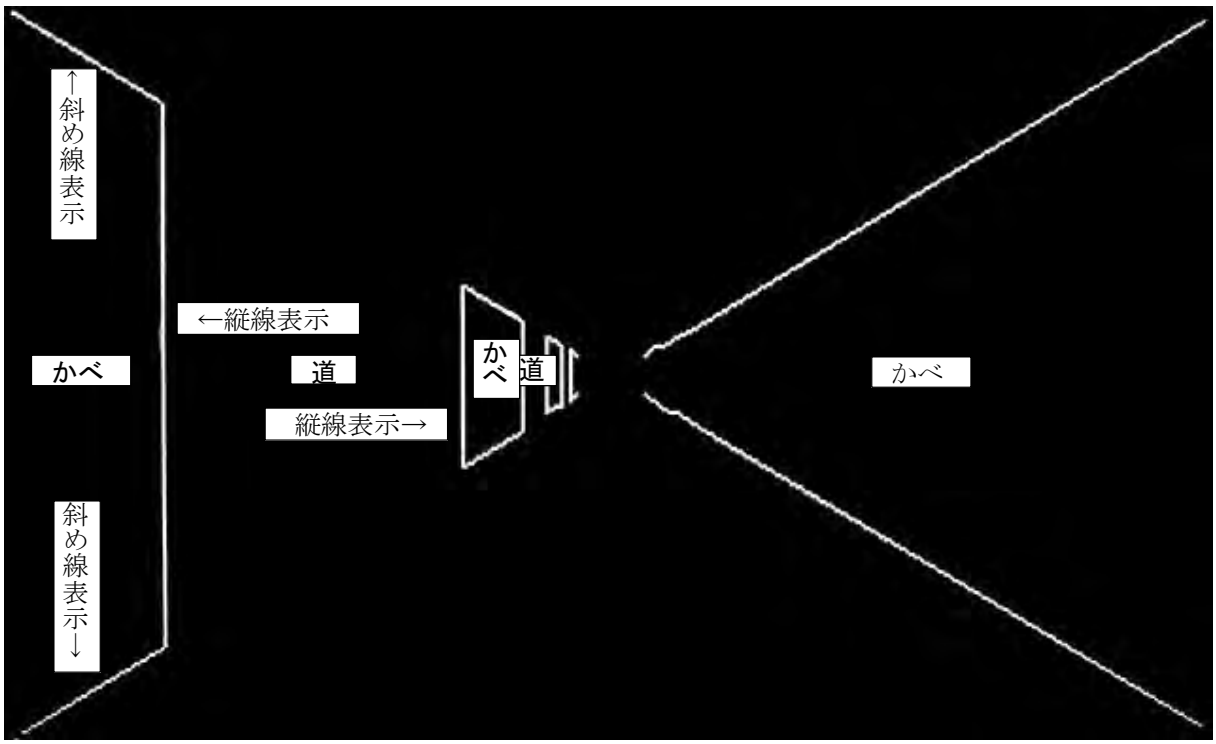
3D 迷路表示のサブルーチンを書きかえます。
迷路の左側と右側をチェックする部分をサブルーチンにして、「かべだったら斜め線 2 本、道だったら縦線 2 本」を表示するようにします。

```
133 '--- 3D めいろ ひょうじ ---
  (中略)
138 GCLS
139 GCOLOR RGB(255,255,255)
140 CX=PX:CY=PY                    チェック座標 CX,CY を
                                   プレイヤー位置 PX,PY にする
141 FOR N=0 TO 6
142 X1=CX+LX[PD]:Y1=CY+LY[PD]
143 DISPLAY 1
144 C1=CHKCHR(X1,Y1)               左側のブロックをチェック
145 IF C1==0 THEN GOSUB @LMICHI ELSE
  GOSUB @LKABE                     道表示／かべ表示サブルーチンを呼ぶ
146 X2=CX+RX[PD]:Y2=CY+RY[PD]
147 DISPLAY 1
148 C2=CHKCHR(X2,Y2)               右側のブロックをチェック
149 IF C2==0 THEN GOSUB @RMICHI ELSE
  GOSUB @RKABE                     道表示／かべ表示サブルーチンを呼ぶ
150 CX=CX+DX[PD]:CY=CY+DY[PD]      チェック座標を
                                   次の位置へ
151 NEXT
152 RETURN
  (↓つづく↓)
```

（↓つづき↓）

```
153
154 'ひだり みち　左側が道の場合のサブルーチン
155 @LMICHI
156 DISPLAY 0
157 GLINE OX-WX[N+1],OY-WY[N+1],OX-W
X[N+1],OY+WY[N+1]　奥側の縦線を表示
158 IF N==0 THEN RETURN　プレイヤーの位置の時は、手前側の縦線は要らないのでもどる
159 GLINE OX-WX[N],OY-WY[N],OX-WX[N]
,OY+WY[N]　手前側の縦線を表示
160 RETURN
161
162 'ひだり かべ　左側がかべの場合のサブルーチン
163 @LKABE
164 DISPLAY 0
165 GLINE OX-WX[N],OY-WY[N],OX-WX[N+
1],OY-WY[N+1]　上側の斜め線を表示
166 GLINE OX-WX[N],OY+WY[N],OX-WX[N+
1],OY+WY[N+1]　下側の斜め線を表示
167 RETURN
168
169 'みぎ みち　右側が道の場合のサブルーチン
170 @RMICHI
171 DISPLAY 0
172 GLINE OX+WX[N+1],OY-WY[N+1],OX+W
X[N+1],OY+WY[N+1]　奥側の縦線を表示
173 IF N==0 THEN RETURN　プレイヤーの位置の時は、手前側の縦線は要らないのでもどる
174 GLINE OX+WX[N],OY-WY[N],OX+WX[N]
,OY+WY[N]　手前側の縦線を表示
175 RETURN
176
177 'みぎ かべ　右側がかべの場合のサブルーチン
178 @RKABE
179 DISPLAY 0
180 GLINE OX+WX[N],OY-WY[N],OX+WX[N+
1],OY-WY[N+1]　上側の斜め線を表示
181 GLINE OX+WX[N],OY+WY[N],OX+WX[N+
1],OY+WY[N+1]　下側の斜め線を表示
182 RETURN
```

プログラムを実行してみましょう。迷路に合わせて道とかべが表示されます。
キャラクターを十字キーでいろいろ移動させて、確認してください。



道のところは手前と奥の縦線だけが表示されていますが、道が横へ続いているように表示したいですね。道を表示するサブルーチンを改造して、横線 2 本も表示するようにしましょう。

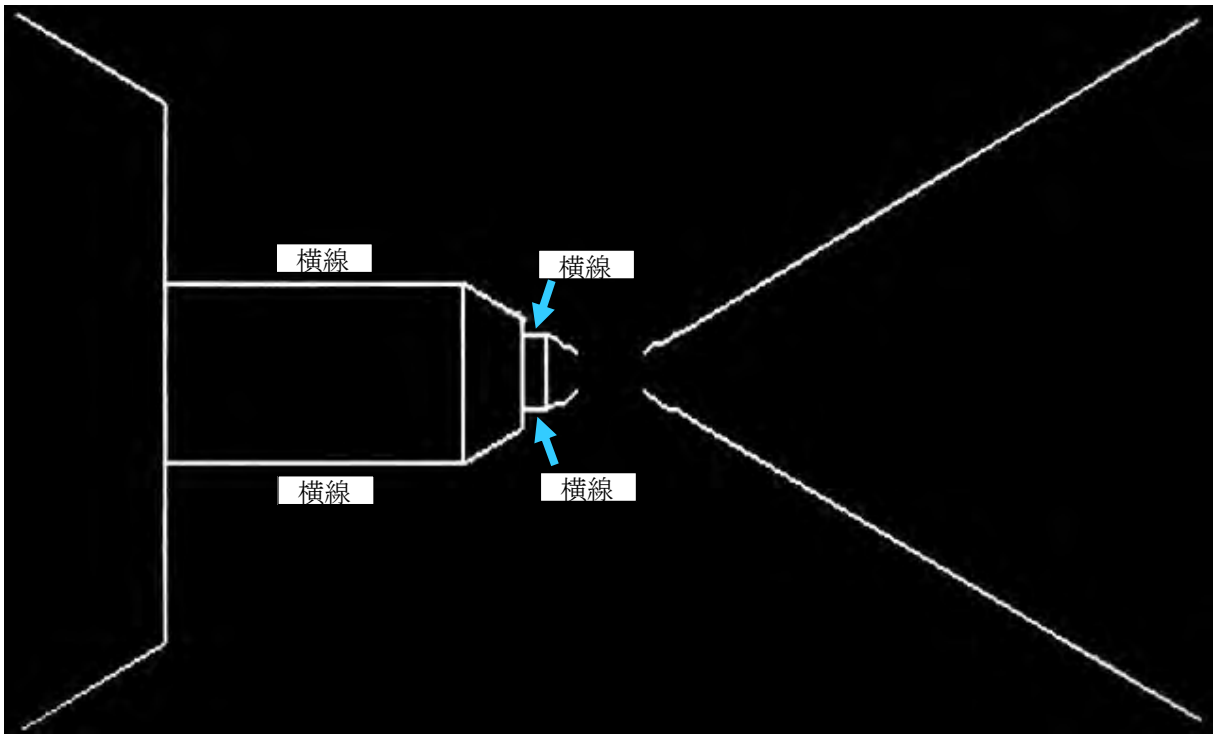
```
154  'ひだり みち
155  @LMICHI
156  DISPLAY 0
157  GLINE  OX-WX[N+1],OY-WY[N+1],OX-W
X[N+1],OY+WY[N+1]
158  GLINE  OX-WX[N],OY-WY[N+1],OX-WX[
N+1],OY-WY[N+1]  上側の横線表示
159  GLINE  OX-WX[N],OY+WY[N+1],OX-WX[
N+1],OY+WY[N+1]  下側の横線表示
160  IF  N==0  THEN  RETURN
161  GLINE  OX-WX[N],OY-WY[N],OX-WX[N]
,OY+WY[N]
162  RETURN
```

（↓つづく↓）

```
（↓つづき↓）

171 'みぎ みち
172 @RMICHI
173 DISPLAY 0
174 GLINE OX+WX[N+1],OY-WY[N+1],OX+W
X[N+1],OY+WY[N+1]
175 GLINE OX+WX[N],OY-WY[N+1],OX+WX[
N+1],OY-WY[N+1]  上側の横線表示
176 GLINE OX+WX[N],OY+WY[N+1],OX+WX[
N+1],OY+WY[N+1]  下側の横線表示
177 IF N==0 THEN RETURN
178 GLINE OX+WX[N],OY-WY[N],OX+WX[N]
,OY+WY[N]
179 RETURN
```

プログラムを実行してみましょう。道のところに横線 2 本が表示されて、道らしくなります。



今のままだと、道が行き止まりになっていても、突き抜けて向こう側が表示されてしまいます。
行き止まりの場合は、突き当たりのかべを表示して、それより奥は表示しないようにします。

```
133 '--- 3D めいろ ひょうじ ---
  (中略)
149 IF C2==0 THEN GOSUB @RMICHI ELSE
GOSUB @RKABE
150 CX=CX+DX[PD]:CY=CY+DY[PD]
151 DISPLAY 1
152 C3=CHKCHR(CX,CY)
153 IF C3>0 THEN GOSUB @IKIDOMARI:N=6
154 NEXT
155 RETURN
```

1 つ先のブロックをチェック

もしかべだったら、行き止まりのサブルーチンを呼んだ後、
N を 6 にしてループを終わらせる(それ以上表示する必要がない)

プログラムの最後に、行き止まりの表示をするサブルーチンを追加します。

```
190
191 'いきどまり
192 @IKIDOMARI
193 DISPLAY 0
194 CL=RGB(255,255,255)
195 IF C1==0 THEN CL=RGB(0,0,0)
196 GLINE OX-WX[N+1],OY-WY[N+1],OX-WX
[N+1],OY+WY[N+1],CL
197 CL=RGB(255,255,255)
198 IF C2==0 THEN CL=RGB(0,0,0)
199 GLINE OX+WX[N+1],OY-WY[N+1],OX+WX
[N+1],OY+WY[N+1],CL
200 GLINE OX-WX[N+1],OY-WY[N+1],OX+WX
[N+1],OY-WY[N+1]
201 GLINE OX-WX[N+1],OY+WY[N+1],OX+WX
[N+1],OY+WY[N+1]
202 RETURN
```

線の色を白に設定

もし左側が道だったら
線の色を黒に(線を消す)

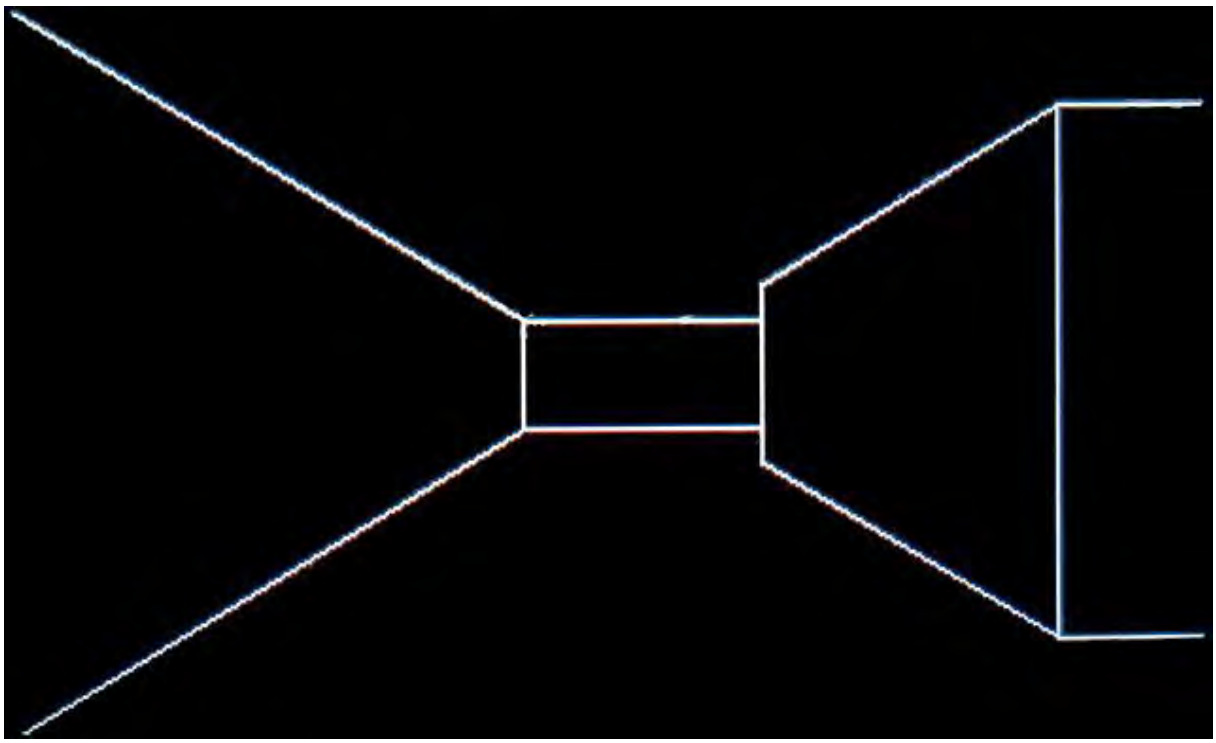
左側の縦線を引く

右側の縦線の処理(左側と同様)

上側の横線を表示

下側の横線を表示

プログラムを実行してみましょう。
迷路が行き止まりのところでも、ちゃんと表示されるようになります。



これで 3D 迷路の表示は完成です。

プログラムを「DANJON5」の名前で保存しましょう。

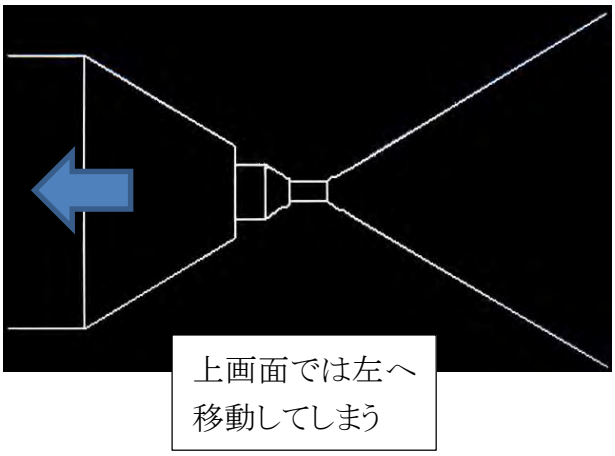
```
SAVE "DANJON5"
```

★できる人は

今回は線を全部白で表示していますが、道の横線・縦線・斜め線をうまく色分けすると、見やすい迷路になるでしょう。やってみましょう。

●十字キー操作を 3D 迷路に合わせる

上画面に 3D 迷路を表示できました。
しかし十字キーの操作が、下画面の迷路で上下左右に動くようになっているので、上画面の 3D 迷路を見て操作するとおかしな感じになります。
例えば下画面の迷路で、プレイヤーが下を向いている時に十字キーの「右」を押して移動すると、上画面の 3D 迷路では左へ移動してしまいます。

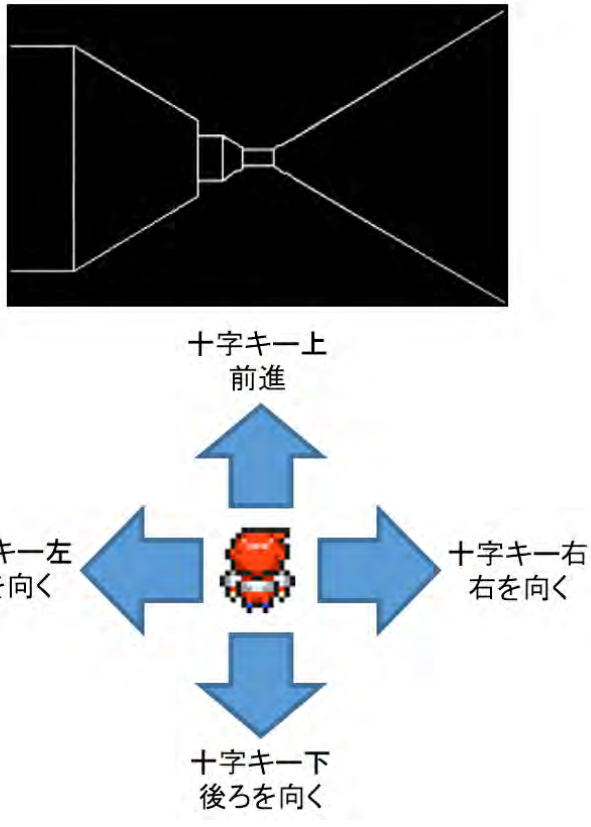


これではわかりづらいので、上画面の 3D 表示に合わせた移動操作に変えましょう。

操作は右図のように、

十字キー上＝前進
十字キー左＝左を向く
十字キー右＝右を向く
十字キー下＝後ろを向く

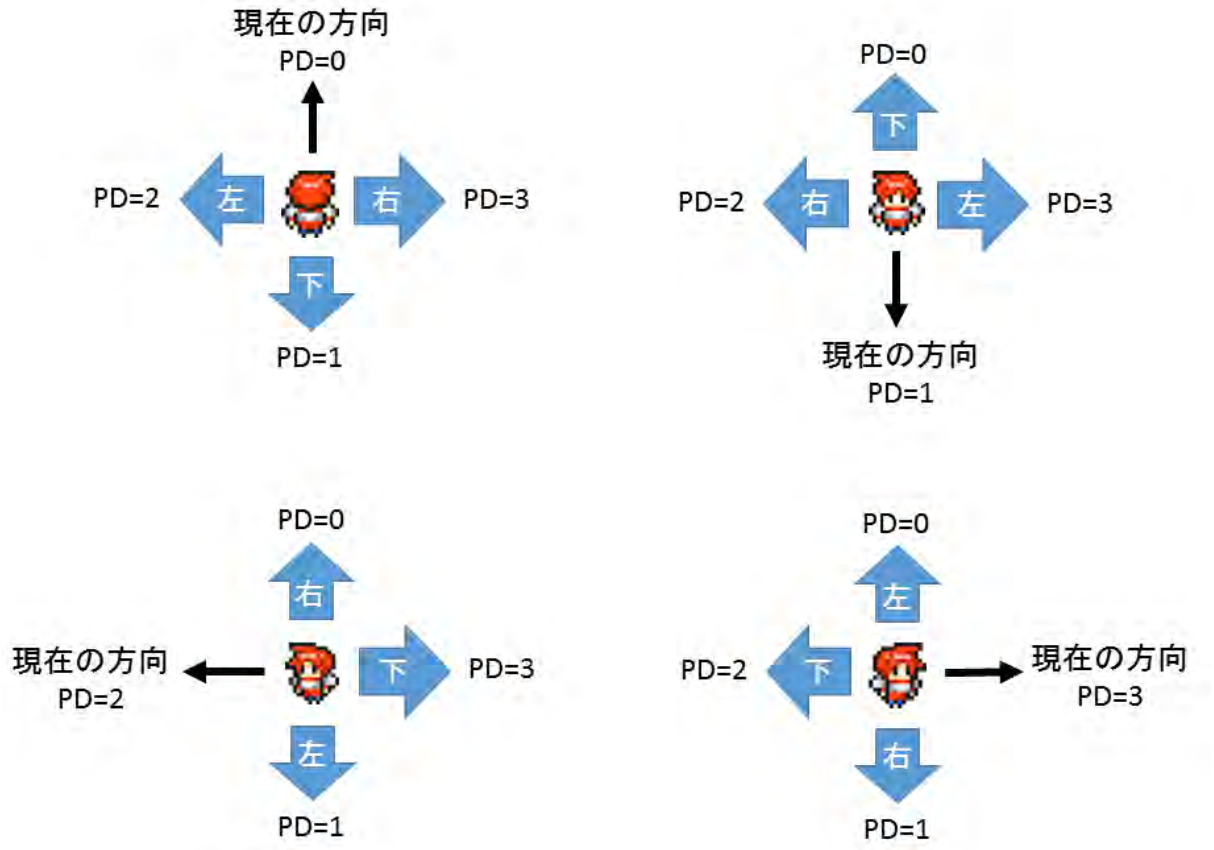
にします。



こうすると、十字キーの上を押して前進する時の移動先座標 NX,NY が、進行方向の増分 DX[],DY[]とプレイヤーの向き PD を使って、

```
NX=PX+DX[PD]
NY=PY+DY[PD]
```

とかんたんに計算できます。
むずかしいのは、十字キーの下・左・右を押した時の、方向(PD)の変化です。



プレイヤーが上を向いている時(PD=0)
- 十字キーの下を押す:PD=1 になる
- 十字キーの左を押す:PD=2 になる
- 十字キーの右を押す:PD=3 になる

プレイヤーが下を向いている時(PD=1)
- 十字キーの下を押す:PD=0 になる
- 十字キーの左を押す:PD=3 になる
- 十字キーの右を押す:PD=2 になる

プレイヤーが左を向いている時(PD=2)
- 十字キーの下を押す:PD=3 になる
- 十字キーの左を押す:PD=1 になる
- 十字キーの右を押す:PD=0 になる

プレイヤーが右を向いている時(PD=3)
- 十字キーの下を押す:PD=2 になる
- 十字キーの左を押す:PD=0 になる
- 十字キーの右を押す:PD=1 になる

関係が複雑ですね。どうすればいいでしょうか？

これを解決するために、方向を変換する配列変数を設定します。

```
1  '*** DANJON6 ***        タイトルを「DANJON6」にする
  （中略）
22 DIM RX[4],RY[4]
23 RX[0]=1:RY[0]=0
24 RX[1]=-1:RY[1]=0
25 RX[2]=0:RY[2]=-1
26 RX[3]=0:RY[3]=1
27 DIM DD[4],DL[4],DR[4]
28 DD[0]=1:DD[1]=0:DD[2]=3:DD[3]=2
29 DL[0]=2:DL[1]=3:DL[2]=1:DL[3]=0
30 DR[0]=3:DR[1]=2:DR[2]=0:DR[3]=1
31
32 '--- めいろ さくせい ---
```

配列変数 DD[]は、「十字キーの下が押された時に、どの方向へ変化するか」を設定しています。例えば現在の方向 PD=0 だったら、変化後の方向は DD[PD]=DD[0]=1 になります。同様に、DL[]は「十字キーの左が押された時に、どの方向へ変化するか」、DR[]は「十字キーの右が押された時に、どの方向へ変化するか」を設定しています。

この配列変数を使って、プレイヤーを動かすプログラムを改造します。

```
66 '--- プレイヤーをうごかす ---
  （中略）
76 @PLAYERMOVE
77
78 B=BUTTON(2)
79 DISPLAY 1                    条件式を「B==～」だけ残して削除
80 IF B==1 THEN GOSUB @UP
81 IF B==2 THEN GOSUB @DOWN
82 IF B==4 THEN GOSUB @LEFT
83 IF B==8 THEN GOSUB @RIGHT
84 SPOFS 0,PX*8-4,PY*8-4
85
```

```
109  'うえ
110  @UP
111  NX=PX+DX[PD]:NY=PY+DY[PD]
112  IF CHKCHR(NX,NY)>0 THEN RETURN
113  PX=NX:PY=NY
114  SPANIM 0,3,10,508,10,509,10,510,
10,511,0
115  RETURN
116
117  'した
118  @DOWN
119  SPANIM 0,3,10,500,10,501,10,502,
10,503,0
120  PD=DD[PD]
121  RETURN
122
123  'ひだり
124  @LEFT
125  SPANIM 0,3,10,504,10,505,10,506,
10,507,0
126  PD=DL[PD]
127  RETURN
128
129  'みぎ
130  @RIGHT
131  SPANIM 0,3,10,496,10,497,10,498,
10,499,0
132  PD=DR[PD]
133  RETURN
```

プレイヤーの向き(PD)に前進した、次の位置を計算

行き先がかべだったら、進めないのでもどる

「PD=0」の行を削除

「PY=PY+1」の行を削除

配列変数で方向変換

「PX=PX-1」の行を削除

配列変数で方向変換

「PX=PX+1」の行を削除

配列変数で方向変換

プログラムを実行してみましょう。上画面の 3D 迷路に合った操作になります。
ただし、下画面のスプライトのアニメーションが、おかしな向きに動いてしまいます。
(押したキーの方向へ向いたアニメーションになってしまうため)
アニメーションの部分を、配列変数を使って書きかえましょう。

プログラムの最初で、アニメーション用の配列変数を設定します。

```
27 DIM DD[4],DL[4],DR[4]
28 DD[0]=1:DD[1]=0:DD[2]=3:DD[3]=2
29 DL[0]=2:DL[1]=3:DL[2]=1:DL[3]=0
30 DR[0]=3:DR[1]=2:DR[2]=0:DR[3]=1
31 DIM SA[4]
32 SA[0]=508:SA[1]=500:SA[2]=504:SA
[3]=496
33
34 '--- めいろ さくせい ---
```

アニメで使うスプライト番号を、配列変数 SA[]に設定

プレイヤーを上下左右に動かすプログラムで、アニメーションの設定を変更します。

```
111 'うえ
112 @UP
113 NX=PX+DX[PD]:NY=PY+DY[PD]
114 IF CHKCHR(NX,NY)>0 THEN RETURN
115 PX=NX:PY=NY
116 RETURN
117
118 'した
119 @DOWN
120 PD=DD[PD]
121  SPANIM  0,3,10,SA[PD],10,SA[PD]+
1,10,SA[PD]+2,10,SA[PD]+3,0
122 RETURN
123
124 'ひだり
125 @LEFT
126 PD=DL[PD]
127  SPANIM  0,3,10,SA[PD],10,SA[PD]+
1,10,SA[PD]+2,10,SA[PD]+3,0
128 RETURN
129
130 'みぎ
131 @RIGHT
129 PD=DR[PD]
130  SPANIM  0,3,10,SA[PD],10,SA[PD]+
1,10,SA[PD]+2,10,SA[PD]+3,0
131 RETURN
```

「SPANIM ～」の行を削除

「SPANIM ～」の行を削除

「SPANIM ～」の行を削除

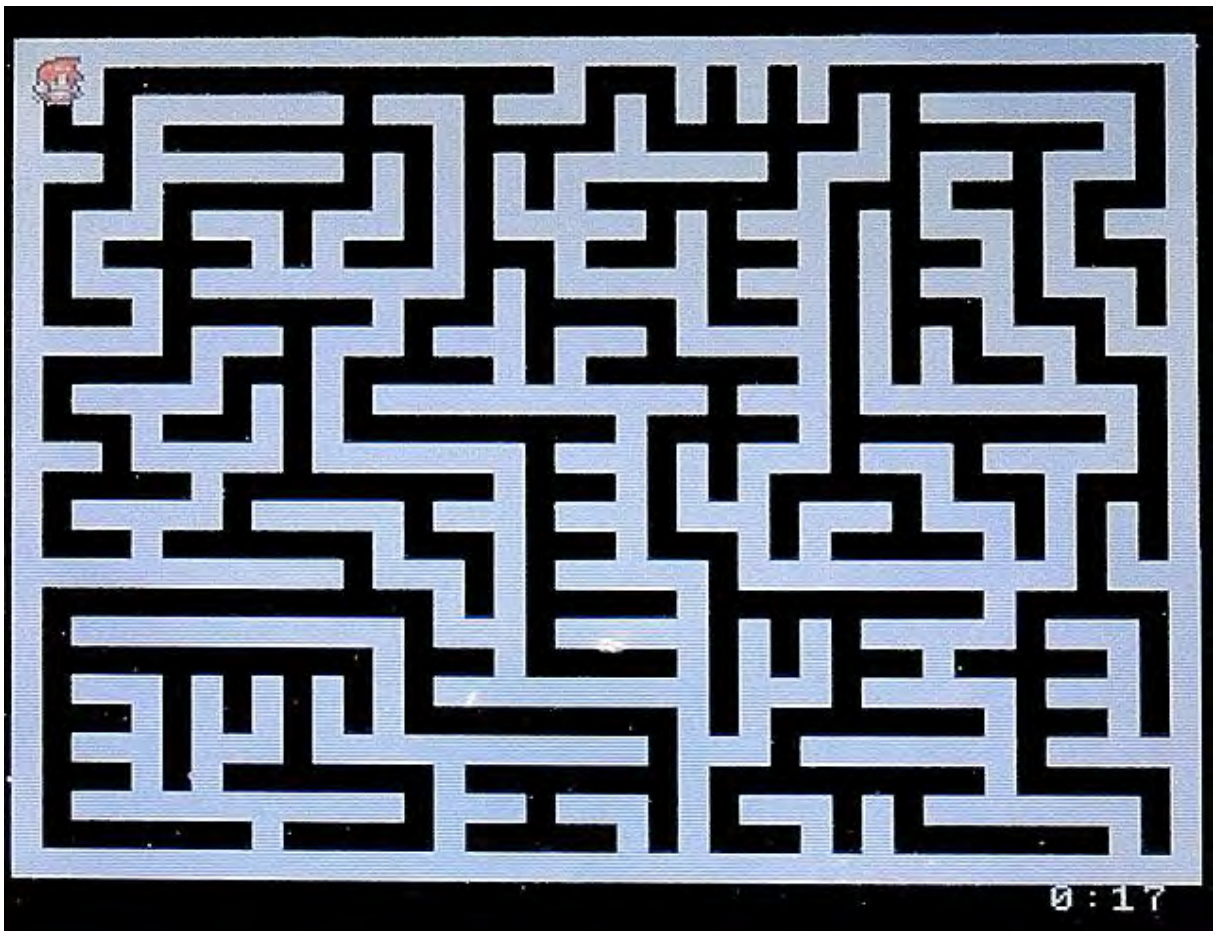
「SPANIM ～」の行を削除

スプライトのアニメーションを配列変数で設定

スプライトのアニメーションを配列変数で設定

スプライトのアニメーションを配列変数で設定

プログラムを実行してみましょう。
下画面のスプライトが、ちゃんとプレイヤーの向きのアニメーションになります。



プログラムを「DANJON6」の名前で保存しましょう。

```
SAVE "DANJON6"
```

★できる人は

32 行目の配列変数 SA[]の値を変えると、スプライトのキャラクターが変わります。
試してみてください。
このように、プログラムの最初に変数でキャラクター番号などを設定しておくと、後でかんたんに変えられます。