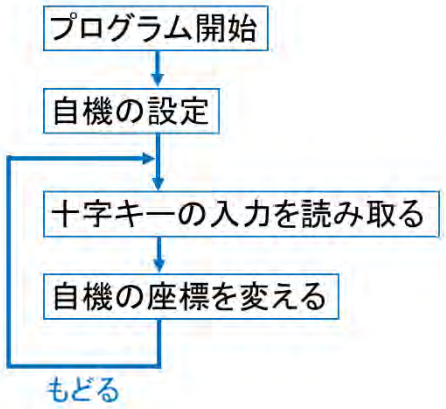
プチコンで横スクロールシューティングゲームを作る（２）

●ビームを発射する

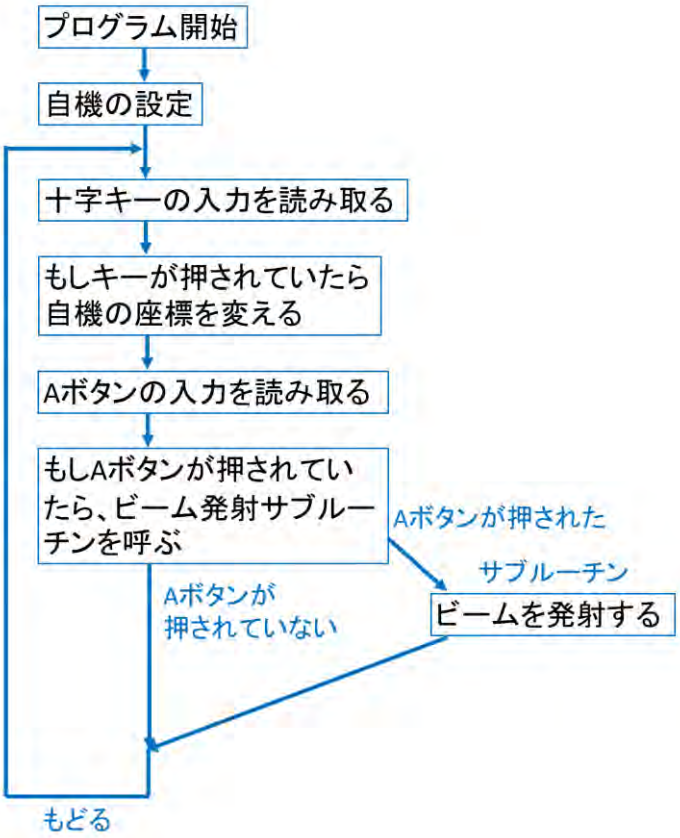
前回までで、自機を十字キーで上下左右に動かすプログラムを作りました。
シューティングゲームらしく、ボタンを押したらビームが発射できるようにしてみましょう。

★プログラムの流れを考える

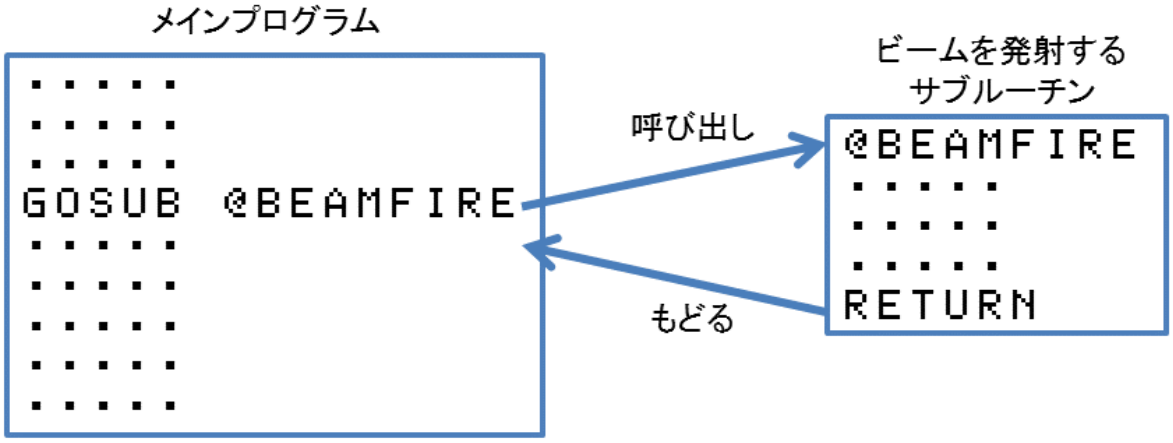
ビームを発射するために、まずプログラム全体の流れを考えてみます。
これまで作ったプログラムの流れは、右の図のようになっています。
（このようにプログラムの流れを書いた図を、「**フローチャート**」と言います）



このプログラムに、ビームを発射するプログラムを追加します。
あとのプログラムの作りやすさを考えて、ビームを発射するプログラムを「**サブルーチン**」にして、メインのプログラムから呼び出すようにします。



「サブルーチン」とは、メインプログラムから呼び出すサブのプログラムです。



サブルーチンを使うと、プログラム全体の流れがわかりやすくなります。また、同じプログラムを何回も実行する時に便利になります。

★ビームを設定する

まず、自機と同じように、ビームのスプライトなどを最初に設定します。

```
1  '*** SHOOT4 ***
(中略)
12  '--- ジキ ---
(中略)
19  SPSHOW 0
20
21  '--- ビーム ---
22  BM=0
23  SPSET 1,1305
24  SPHOME 1,4,8
25  SPROT 1,90
26  SPHIDE 1
27
28  '--- ジキヲ ウゴカス ---
29  @MOVELOOP
```

タイトルを「SHOOT4」にする

コメントを入れる

ビームが存在するかどうかの変数。最初は 0

ビームのスプライトを
管理番号 1 番に設定

最初はビームが発射されていないので、表示しない

22 行:ビームが存在するかどうかを指定する変数 BM を 0(存在しない)にします。
あとでビームを発射する時に、この変数を使います。

23～25 行:ビームのスプライトを管理番号 1 番として設定します。
ビームのスプライトは、1305 番を使います。
上向きのスプライトなので、SPHOME 命令で原点を移動した後、SPROT 命令で 90 度回転させています。
※SPSET 命令のスプライト番号を変えると、ビームのキャラクターが変わります。

26 行:最初はビームが存在しないので、SPHIDE 命令でビームを表示しないようにします。

★ビームを発射する

続いて、自機を動かすループのプログラムを改造します。
A ボタンを押したら、ビームが発射されるようにします。

```
28 '--- ジキヲ ウゴカス ---
29 @MOVELOOP
30 B=BUTTON(0)
31 IF B==1 AND PY>16 THEN PY=PY-2
32 IF B==2 AND PY<224 THEN PY=PY+2
33 IF B==4 AND PX>16 THEN PX=PX-2
34 IF B==8 AND PX<384 THEN PX=PX+2
35 SPOFS 0,PX,PY
36 IF B==16 AND BM==0 THEN GOSUB @
BEAMFIRE
37 VSYNC 1
38 GOTO @MOVELOOP
```

もし A ボタンが押されていて、かつビームが存在していなかったら、ビームを発射するサブルーチンを呼ぶ

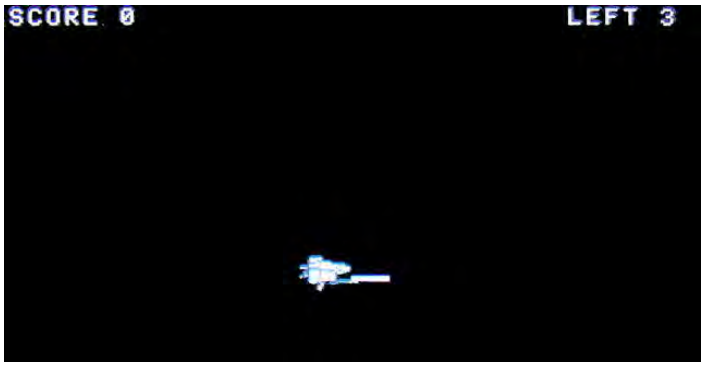
IF 命令を使って、もし A ボタンが押されていて(BUTTON 関数の値が 16)、かつビームが存在していない(BM の値が 0)時に、ビームを発射するサブルーチンを呼びます。

プログラムの最後に、ビームを発射するサブルーチンを追加します。

```
39
40 '--- ビーム ハッシャ ---
41 @BEAMFIRE
42 BM=1
43 BX=PX+16
44 BY=PY
45 SPOFS 1,BX,BY
46 SPSHOW 1
47 RETURN
```

コメント
ラベル
ビーム存在変数を 1 にする
ビームの座標 BX,BY を、自機の座標 PX,PY から設定
ビームのスプライトを座標(BX,BY)へ移動
ビームのスプライトを表示

プログラムを実行してみましょう。
A ボタンを押すと、自機からビームが発射されます。
ただし、発射されるだけでそれ以上は動きません。
ビームを動かすプログラムを、まだ作っていないからです。



★ビームを動かす

発射されたビームを動かすプログラムを作りましょう。
自機を動かすループプログラムで、ビームが存在している時は、ビームを動かすサブルーチンを呼ぶようにします。

```
28 '--- ジキヲ ウゴカス ---
29 @MOVELOOP
30 B=BUTTON(0)
31 IF B==1 AND PY>16 THEN PY=PY-2
32 IF B==2 AND PY<224 THEN PY=PY+2
33 IF B==4 AND PX>16 THEN PX=PX-2
34 IF B==8 AND PX<384 THEN PX=PX+2
35 SPOFS 0,PX,PY
36 IF B==16 AND BM==0 THEN GOSUB @BEAMFIRE
37 IF BM==1 THEN GOSUB @BEAMMOVE
38 VSYNC 1
39 GOTO @MOVELOOP
```

もしビームが存在していたら、ビームを動かすサブルーチンを呼ぶ

ビームが存在している＝ビーム存在変数 BM が 1 なので、その時はビームを動かすサブルーチンを呼びます。

プログラムの最後に、ビームを動かすサブルーチンを追加します。

```
49
50 '--- ビーム イドウ ---
51 @BEAMMOVE
52 BX=BX+8
53 SPOFS 1,BX,BY
54 IF BX>400 THEN BM=0:SPHIDE 1
55 RETURN
```

コメント
ラベル
ビームの横座標 BX を 8 加算
ビームのスプライトを座標(BX,BY)へ移動
もし BX が 400 より大きかったら、画面をはみ出すのでビームを消す

プログラムを実行してみましょう。
ビームを発射すると、画面右はじまで動いて消えます。
ビームが消えると、また次のビームをうつことができます。

52 行で、ビームの横座標 BX を 8 加算して、ビームを右へ動かしています。
加算する値を変えると、ビームの速さが変わります。試してみましょう。

54 行では、ビームが画面の右はじへ行ったかどうかをチェックしています。
もし右はじへ行くと、横座標 BX が 400 より大きくなるので、その時はビーム存在変数 BM を 0 にして、SPHIDE 命令でビームのスプライトを消しています。

★キー入力を改善する

今のプログラムだと、自機が止まっている時はビームを発射できるのですが、十字キーを押して自機を移動しながら A ボタンを押しても、ビームが発射されません。
キーやボタンが**同時に 2 つ以上押された**時のことを考えていないからです。
キー入力・ボタン入力のプログラムを、以下のように改造します。

```
28 '--- ジキヲ ウゴカス ---
29 @MOVELOOP
30 B=BUTTON(0)
31 IF (B AND 1)==1 AND PY>16 THEN PY=PY-2
32 IF (B AND 2)==2 AND PY<224 THEN PY=PY+2
33 IF (B AND 4)==4 AND PX>16 THEN PX=PX-2
34 IF (B AND 8)==8 AND PX<384 THEN PX=PX+2
35 SPOFS 0,PX,PY
36 IF (B AND 16)==16 AND BM==0 THEN GOSUB @BEAMFIRE
37 IF BM==1 THEN GOSUB @BEAMMOVE
38 VSYNC 1
39 GOTO @MOVELOOP
```

もし十字キーの「上」が押されていたら、他のキーが押されていても、自機を上へ移動する。以下も同様。

もし A ボタンが押されていたら、他のキーが押されていても、ビームを発射する。

プログラムを実行してみましょう。今度は自機が移動しながらでもビームが発射できます。
また、十字キーも「同時押し」ができるので、斜めへ移動できるようになります。

★プログラムを「SHOOT4」の名前で保存しましょう。

SAVE ” SHOOT4”

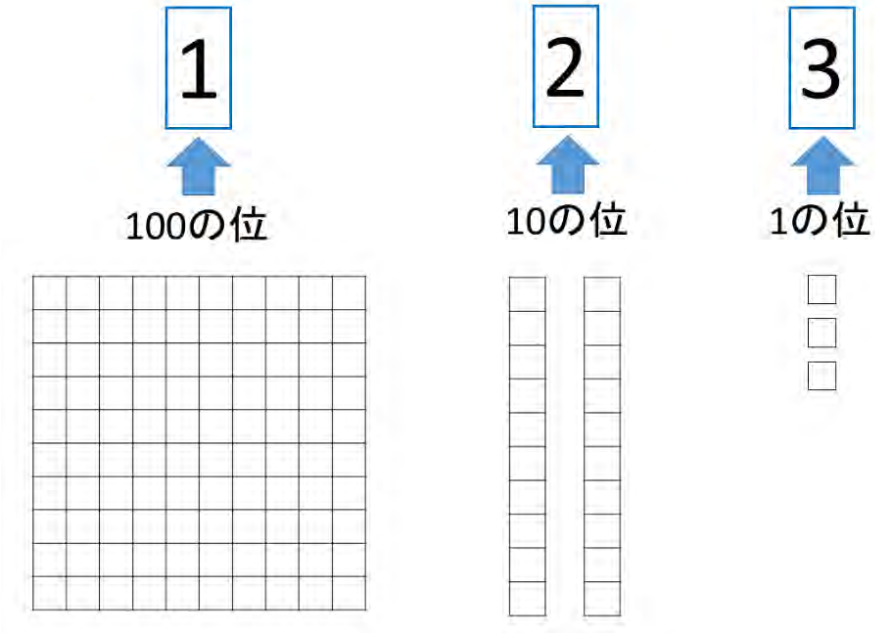
ここでは、AND を使って**論理演算**(ろんりえんざん)をしています。
説明がむずかしくなるので、以下は詳しく知りたい人だけ読んでください。

詳しく知りたい人だけ読んでください

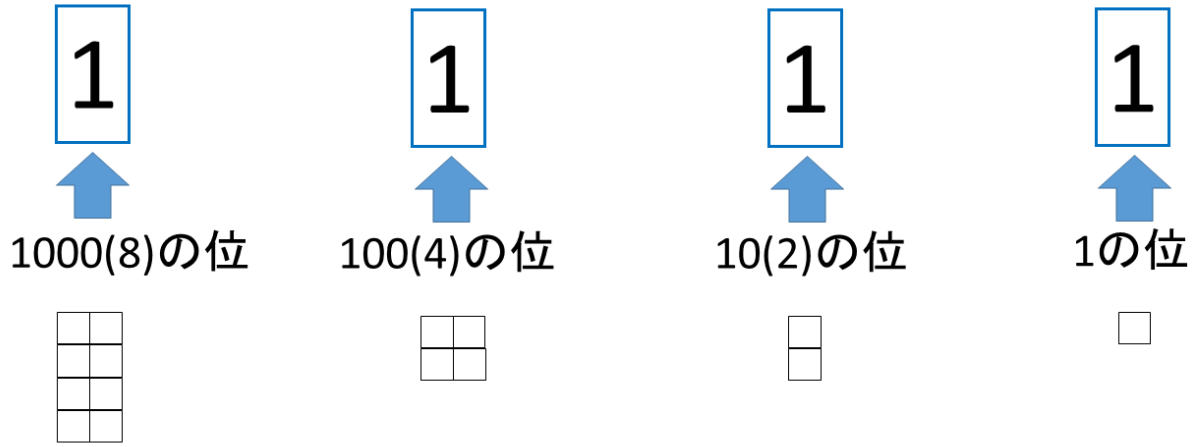
以前に、BUTTON 関数の説明で、押されたキーと関数の値の表を出しました。

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

この数は、実は **2 進数**(2 しんすう)から来ています。
私たちがふだん使っている数は **10 進数**(10 しんすう)で、「10 個集まると位が 1 つ上にあがる」数です。例えば「123」という数は、「100」が 1 個、「10」が 2 個、「1」が 3 個の集まりです。



これに対して 2 進数は、「2 個集まると位が 1 つ上にあがる」数です。
例えば 2 進数の「1111」という数は、「1000」(10 進数で 8)が 1 個、「100」(4)が 1 個、「10」(2)が 1 個、「1」(1)が 1 個の集まりです。



10 進数と 2 進数は、以下のように対応します。

10 進数	2 進数	タイルで考えると…
0	0	
1	1	□
2	10	□□
3	11	□□ □
4	100	□□□□
5	101	□□□□ □
6	110	□□□□ □□
7	111	□□□□ □□ □
8	1000	□□□□□□□□
9	1001	□□□□□□□□ □
10	1010	□□□□□□□□ □□
11	1011	□□□□□□□□ □□ □
12	1100	□□□□□□□□ □□□□
13	1101	□□□□□□□□ □□□□ □
14	1110	□□□□□□□□ □□□□ □□
15	1111	□□□□□□□□ □□□□ □□ □
16	10000	□□□□□□□□ □□□□□□□□
:	:	

コンピュータの中では、数字は 2 進数で扱われています。
(電気が ON＝1、OFF＝0 で表しています)

ここで BUTTON 関数の値を、2 進数で考えてみます。

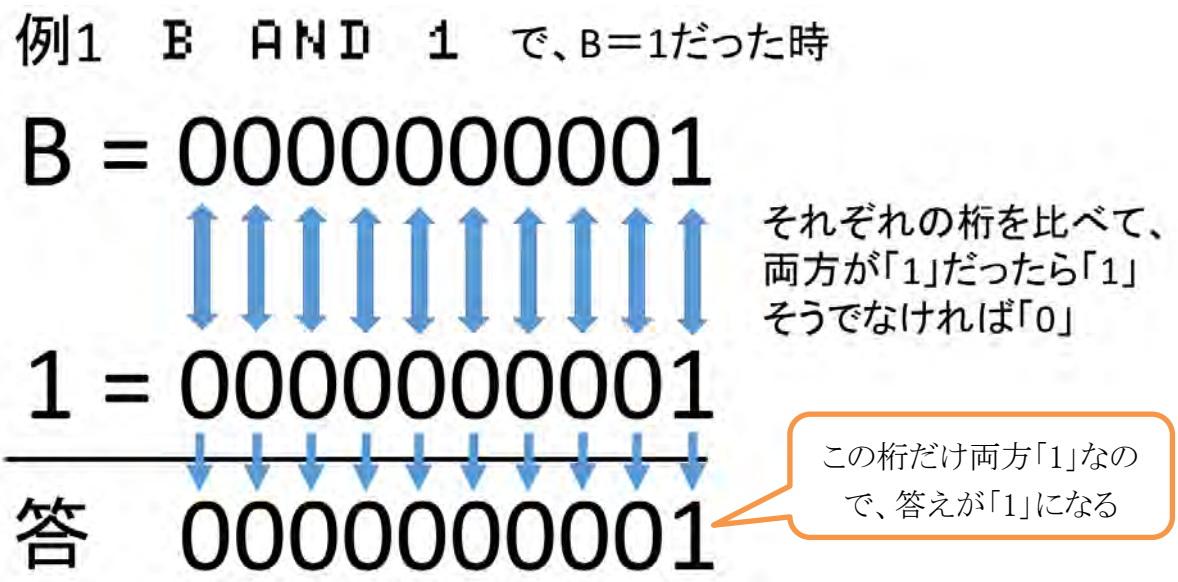
押されたボタン	値（10 進数）	値（2 進数）
何も押さない	0	0000000000
十字キーの上	1	0000000001
十字キーの下	2	0000000010
十字キーの左	4	0000000100
十字キーの右	8	0000001000
A ボタン	16	0000010000
B ボタン	32	0000100000
X ボタン	64	0001000000
Y ボタン	128	0010000000
L ボタン	256	0100000000
R ボタン	512	1000000000

つまり、2 進数のそれぞれの桁が、それぞれのキーやボタンに対応していて、「押されていない＝0」「押されている」＝1になっています。
例えば、十字キーの上と A ボタンを同時に押すと、値は 2 進数で「0000010001」→10 進数で 17 になります。
ですから、例えば「他のキーが押されているかどうかにかかわらず、A ボタンが押されていることをチェックする」ためには、「BUTTON 関数の値を 2 進数で見て、下から 5 桁目が 1 かどうか」を調べればいいことになります。

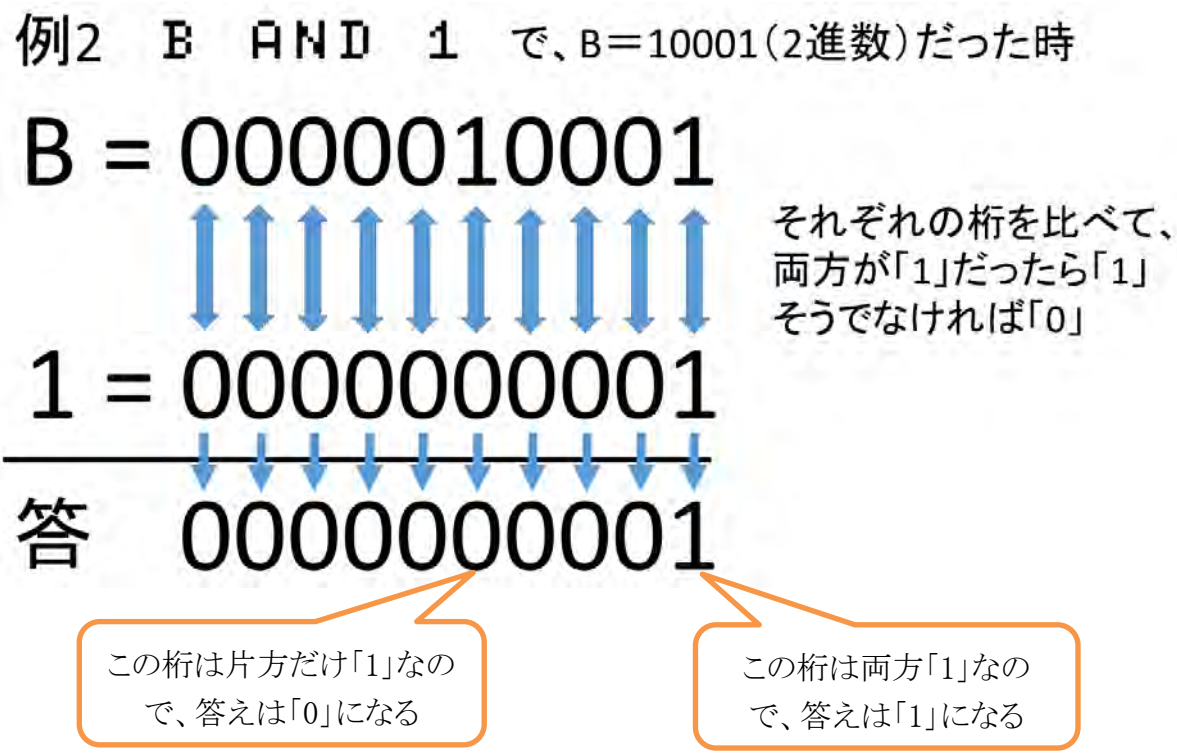
今回のプログラムでは、「2 進数のある桁の値が 1 かどうか」をチェックするのに、**AND**（アンド）を使っています。
例えば十字キーの上が押されているかどうかをチェックするプログラムは、以下のとおりです。

```
IF (B AND 1)==1 AND PY>16 THEN PY=PY-2
```

AND は、「**2 つの数値を 2 進数にして比べて、両方とも 1 の桁だけ 1 に、そうでなければ 0 にする**」という計算をします。
このプログラムの場合、「B」（BUTTON 関数の値）と「1」を 2 進数にして比べます。
例えば十字キーの上が押されていた場合、上の表でわかるように、B は 2 進数で「0000000001」になります。



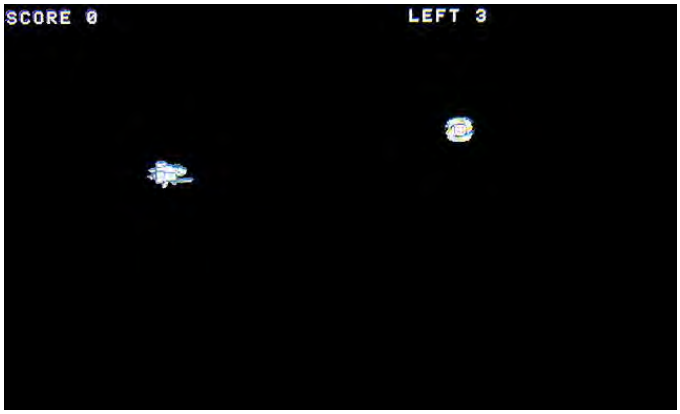
となるので、「B AND 1」の答えは「1」になります。
IF 命令で「IF (B AND 1)==1 THEN ～」となっているので、THEN 以下の命令が実行されて、自機が上へ移動します。
違う例として、もし十字キーの上とAボタンが一緒に押されていたとすると、B は 2 進数で「0000010001」になりますから、



となるので、「B AND 1」の答えはやはり「1」になります。
つまり、AND を使うことで、他のキーやボタンが同時に押されていても、一番右の桁が1かどうか（＝十字キーの上が押されているかどうか）をチェックできます。
十字キーの下・左・右、A ボタンのチェックも同様です。

●敵を登場させる

次は、シューティングでうつ相手、敵を登場させてみましょう。



★敵を出す

最初に、ビームと同じように、敵のスプライトなどを設定します。

```
1  '*** SHOOT5 ***
(中略)
21  '--- ビーム ---
(中略)
26  SPHIDE 1
27
28  '--- テキ ---
29  TK=0
30  SPSET 11,1210
31  SPHIDE 11
32
33  '--- ジキヲ ウゴカス ---
34  @MOVELOOP
```

タイトルを「SHOOT5」にする

コメントを入れる

敵が存在するかどうかの変数。最初は 0

敵のスプライトを管理番号 11 番に設定

最初は敵が出ていないので、表示しない

29 行：敵が存在するかどうかを指定する変数 TK を 0（存在しない）にします。
あとで敵を登場させる時に、この変数を使います。

30～31 行：敵のスプライトを管理番号 11 番として設定します。
敵のスプライトは、1210 番を使います。
SPSET 命令のスプライト番号を変えると、敵のキャラクターが変わります。

メインループのプログラムで、敵を登場させるサブルーチンを呼びます。

```
33 '--- ジキヲ ウゴカス ---
(中略)
41 IF (B AND 16)==16 AND BM==0 THEN GOSUB @BEAMFIRE
42 IF BM==1 THEN GOSUB @BEAMMOVE
43 IF RND(100)<1 AND TK==0 THEN GOSUB @TEKIDERU
44 VSYNC 1
45 GOTO @MOVELOOP
```

もし確率が 100 分の 1 で、かつ敵が登場していなかったら、敵を登場させるサブルーチンを呼ぶ

敵が画面に出ていない(＝敵の存在変数 TK が 0)の時だけ、敵を登場させるサブルーチンを呼びます。
いつも同じタイミングで登場すると面白くないので、乱数で「RND(100)<1」という条件を追加して、出るタイミングをずらしています。「RND(100)」だと 0～99 の乱数が出るので、「RND(100)<1」だと 0 の時しか条件が成り立たず、確率は 100 分の 1 になります。

プログラムの最後に、敵を登場させるサブルーチンを追加します。

```
56 '--- ビーム イドウ ---
(中略)
60 IF BX>400 THEN BM=0:SPHIDE 1
61 RETURN
62
63 '--- テキ トウジョウ ---        コメント
64 @TEKIDERU                       ラベル
65 TK=1                            ビーム存在変数を 1 にする
66 TX=391                          敵の座標 TX,TY を設定
67 TY=RND(208)
68 SPOFS 11,TX,TY                  敵のスプライトを座標(TX,TY)へ移動
69 SPSHOW 11                       敵のスプライトを表示
70 RETURN
```

プログラムを実行してみましょう。
画面右はじに敵のキャラクターが登場します。
ただし、敵を動かすプログラムをまだ作っていないので、右はじに止まったままです。

★敵を動かす

メインループのプログラムで、敵を動かすサブルーチンを呼ぶようにします。

```
33 '--- ジキヲ ウゴカス ---
（中略）
43 IF RND(100)<1 AND TK==0 THEN GOSUB @TEKIDERU
44 IF TK==1 THEN GOSUB @TEKIMOVE
45 VSYNC 1
46 GOTO @MOVELOOP
```

もし敵が登場していたら、敵を動かすサブルーチンを呼ぶ

プログラムの最後に、敵を動かすサブルーチンを追加します。

```
64 '--- テキ トウジョウ ---
（中略）
70 SPSHOW 11
71 RETURN
72
73 '--- テキ イドウ ---
74 @TEKIMOVE
75 TX=TX-2
76 SPOFS 11,TX,TY
77 IF TX<-16 THEN TK=0:SPHIDE 11
78 RETURN
```

コメント

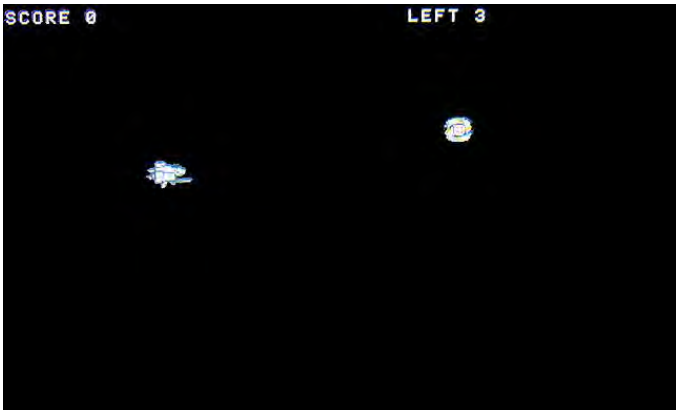
ラベル

敵の横座標 TX を 2 減らす

敵のスプライトを座標(TX,TY)へ移動

もし TX が-16 より小さかったら、画面をはみ出すので敵を消す

プログラムを実行してみましょう。
敵が右はじに登場して、左へ移動していきます。



移動させるプログラムは、ビームを動かすプログラムとほとんど一緒です。

75 行:敵の横座標 TX を減らす（＝左へ移動）
※減らす値「-2」を変えると、敵が動く速度が変わります。
76 行:敵のスプライトを移動させる
77 行:もし TX が-16 より小さくなったら、画面左へはみ出しているので敵を消す

★敵をアニメーションさせる

敵の動きの演出として、アニメーションさせてみましょう。
敵を登場させるサブルーチンで、敵キャラクターのアニメーションを設定します。

```
64 '--- テキ トウジョウ ---
65 @TEKIDERU
66 TK=1
67 TX=391
68 TY=RND(208)
69 SPOFS 11,TX,TY
70 SPSHOW 11
71 SPANIM 11,3,10,1210,10,1211,10,
1212,0
72 RETURN
```

敵のスプライトを 3 コマ分アニメーションさせる

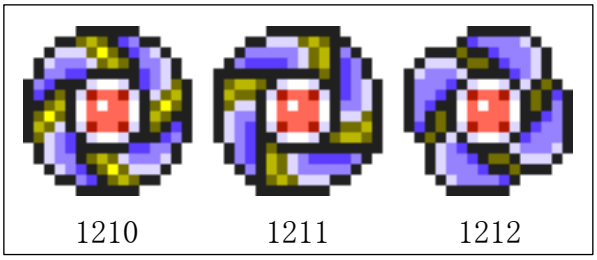
プログラムを実行してみましょう。
敵がアニメーション（回転）しながら動くようになります。

スプライトをアニメーションさせるには、SPANIM（エスピーアニメ）命令を使います。

```
SPANIM   11      ,3     ,10     ,1210    …    ,0
        管理番号  動作   時間   スプライト       ループ
                                 番号           回数
```

管理番号	アニメーションしたいスプライトの管理番号。
動作	3＝コマアニメ ほかに、4＝回転、5＝拡大縮小、6＝色の変更　などがある。
時間	コマアニメにした時:コマを表示する時間。60 分の 1 秒単位で指定。
スプライト番号	コマアニメにした時:表示したいスプライト番号。
…	以下、コマアニメで表示する「時間」「スプライト番号」をくり返し書く。
ループ回数	アニメをくり返す回数を指定。「0」だと無限にくり返し。

今回のアニメ指定では、スプライト番号 1210、1211、1212 の 3 つのスプライトを、それぞれ 60 分の 10 秒＝6 分の 1 秒ずつ表示して、ずっとくり返します。



★プログラムを「SHOOT5」の名前で保存しましょう。

●ビームと敵の当たり判定

今のプログラムでは、ビームをうって敵に当てても、何も起きません。
ビームが敵に当たったら、敵を倒せるようにしてみましょう。

★当たり判定領域の設定

まずビームと敵のスプライトで、当たり判定をする領域を指定します。

```
1  '*** SHOOT6 ***
(中略)
21 '--- ビーム ---
22 BM=0
23 SPSET 1,1305
24 SPHOME 1,4,8
25 SPROT 1,90
26 SPHIDE 1
27 SPCOL 1,-8,-4,16,8
28
29 '--- テキ ---
30 TK=0
31 SPSET 11,1210
32 SPHIDE 11
33 SPCOL 11,0,0,16,16
34
35 '--- ジキヲ ウゴカス ---
```

タイトルを「SHOOT6」にする

ビームのスプライトで、
当たり判定用の領域を設定

敵のスプライトで、
当たり判定用の領域を設定

SPCOL（エスピーカラム）命令を使って、当たり判定領域を指定します。

SPCOL	1	,-8	,-4	,16	,8
	管理番号	領域左上 x 座標	領域左上 y 座標	領域横幅	領域縦幅

管理番号	当たり判定領域を設定するスプライトの管理番号。
領域左上 x 座標	当たり判定領域の左上の x 座標。
領域左上 y 座標	当たり判定領域の左上の y 座標。
領域横幅	当たり判定領域の横幅。
領域縦幅	当たり判定領域の縦幅。

今回は、ビームと敵の当たり判定領域を、以下のように設定しています。

ビーム　SPCOL　1,-8,-4,16,8

原点をスプライトの中央にしているので、
そこから左に8ドット、上へ4ドットの位置を
当たり判定領域の左上の位置にする。
そこから横幅16ドット、縦幅8ドットの長方形
を当たり判定領域とする。

敵　SPCOL　11,0,0,16,16

原点はスプライトの左上のままなので、
当たり判定領域の左上も同じ位置にする。
そこから横幅16ドット、縦幅16ドットの正方
形を当たり判定領域とする。

2つの当たり判定領域が重なると
「当たった」と判定される。

これで当たり判定をする準備ができたので、実際に当たり判定をするプログラムを追加します。

★当たり判定

ビームを移動するサブルーチンで、2 つのスプライトの当たり判定をします。

```
60 '--- ビーム イドウ ---
61 @BEAMMOVE
62 BX=BX+8
63 SPOFS 1,BX,BY
64 IF BX>400 THEN BM=0:SPHIDE 1
65 IF SPHITSP(1,11)==1 THEN GOSUB
@BEAMTEKIHIT
66 RETURN
```

もしビームと敵が当たっていたら、
当たりのサブルーチンを呼ぶ

当たり判定には、SPHITSP（エスピー・ヒット・エスピー）関数を使います。

SPHITSP(1 ,11)
管理番号 1　管理番号 2

管理番号 1	当たり判定をする 1 個目のスプライトの管理番号。
管理番号 2	当たり判定をする 2 個目のスプライトの管理番号。
関数の値	2 つのスプライトが当たっている＝1 2 つのスプライトが当たっていない＝0

プログラムの最後に、ビームと敵が当たった時のサブルーチンを追加します。

```
78 '--- テキ イドウ ---
（中略）
82 IF TX<-16 THEN TK=0:SPHIDE 11
83 RETURN
84
85 '--- ビーム・テキ ヒット ---
86 @BEAMTEKIHIT
87 BM=0
88 SPHIDE 1
89 TK=0
90 SPHIDE 11
91 SC=SC+10
92 LOCATE 0,0
93 PRINT "SCORE ";SC
94 RETURN
```

コメント
ラベル
ビームを消す
敵を消す
スコアを 10 点加算
スコアを表示

プログラムを実行してみましょう。
ビームが敵に当たると、ビームと敵が消えて、スコアが 10 点加算されます。

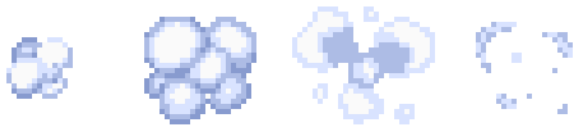
★爆発の演出

今のプログラムだと、ビームが敵に当たると両方が消えるだけなので、「敵をやっつけた」感じがあまりしません。
ビームが当たった処理をするサブルーチンを改造して、敵が爆発するようにしてみましょう。

```
78 '--- テキ イドウ ---
（中略）
82 IF TX<-16 THEN TK=0:SPHIDE 11
83 RETURN
84
85 '--- ビーム・テキ ヒット ---
86 @BEAMTEKIHIT
87 BM=0
88 SPHIDE 1
89 TK=0
90 SPANIM 11,3,10,1376,10,1377,10,
1378,10,1379,10,32,1
91 SC=SC+10
92 LOCATE 0,0
93 PRINT "SCORE ";SC
```

敵のスプライトで、爆発のアニメーションを表示

90 行の SPHIDE 命令を消して、代わりに SPANIM 命令でアニメーションの設定をします。
爆発のコマアニメを 5 コマで指定します。



1376　　1377　　1378　　1379　　32（空白）

プログラムを実行してみましょう。
ビームが敵に当たると爆発します。

※何度か敵をうっているとわかりますが、爆発アニメが表示されている最中に次の敵が登場すると、アニメが途中で消えてしまいます。
これをさける方法もあるのですが、プログラムが難しくなるので、今回は気にしないことにします。

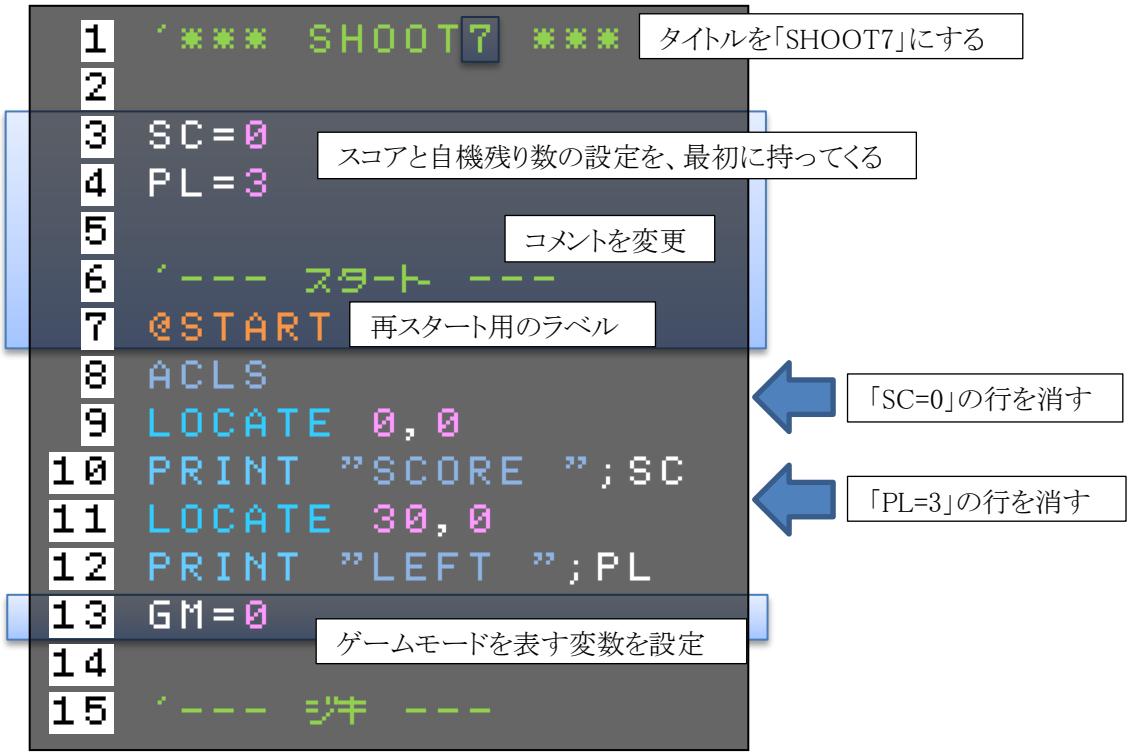
★プログラムを「SHOOT6」の名前で保存しましょう。

●自機と敵の当たり判定

今のままだと、一方的に敵をうって倒しているだけで、自分は死なないので、ゲームになりません。
自機と敵が当たったら、ミスになるようにしてみましょう。

★プログラムの先頭部分の変更

まず、ミスした時の再スタートのために、プログラムの先頭部分を変更します。



13 行のゲームモードを表す変数 GM は、以下の値を取ることにします。

- GM=0　通常のプレイ中
- GM=1　ミスした状態

この GM を使って、ゲーム全体の進行をコントロールします。

★自機の当たり判定を設定

自機の当たり判定をするために、ビームや敵と同様に、当たり判定領域を設定します。

```
15  '--- ｼｷ ---
16  PX=100
17  PY=100
18  SPSET 0,1260
19  SPOFS 0,PX,PY
20  SPHOME 0,16,16
21  SPROT 0,90
22  SPSHOW 0
23  SPCOL 0,-8,-8,16,16
24
25  '--- ビーム ---
```

自機の当たり判定領域を設定

自機　SPCOL 0,-8,-8,16,16

原点をスプライトの中央にしているので、
そこから左に8ドット、上へ8ドットの位置を
当たり判定領域の左上の位置にする。
そこから横幅16ドット、縦幅16ドットの
正方形を当たり判定領域とする。

2つの当たり判定領域が重なると
「当たった」と判定される。

自機を動かすループを改造して、GM が 1 だったらミスの処理をするようにします。

```
39  '--- ジキヲ ウゴカス ---
40  @MOVELOOP
(中略)
51  VSYNC 1
52  IF GM==0 THEN GOTO @MOVELOOP
53
54  '--- ミス ---
55  SPANIM 0,3,10,3424,10,3425,10,3426,
10,3427,1
56  PL=PL-1
57  LOCATE 30,0
58  PRINT "LEFT ";PL
59  VSYNC 120
60  GOTO @START
61
62  '--- ビーム ハッシャ ---
```

もし GM が 0 だったら、ミスではないのでもどる

コメント

自機が爆発するコマアニメを表示

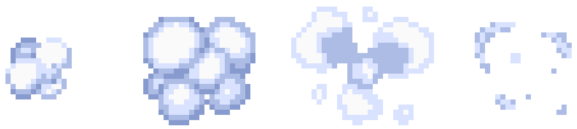
自機の残り数を 1 減らす

自機の残り数を表示

しばらく待つ

再スタートする

55 行では、SPANIM 命令で自機が爆発するアニメーションを設定しています。
敵の爆発と同様ですが、原点がスプライトの中心にあるスプライト 3424 番～3427 番を使っています。(自機と同じ位置に表示するため)



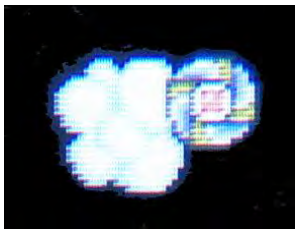
3424　　3425　　3426　　3427

最後に、敵を移動させるサブルーチンで、自機との当たり判定を追加します。

```
89  '--- テキ イドウ ---
90  @TEKIMOVE
91  TX=TX-2
92  SPOFS 11,TX,TY
93  IF TX<-16 THEN TK=0:SPHIDE 11
94  IF SPHITSP(0,11)==1 THEN GM=1
95  RETURN
```

もし自機と敵が当たったら、ゲームモード GM を 1 にする

プログラムを実行してみましょう。
自機と敵が当たるとミスになり、
自機の残り数が1機減ります。



このままだと、自機の残り数が減ってマイナスになっても、ゲームが続いてしまいます。
自機が 0 になったらゲームオーバーになるようにしましょう。
ミスのプログラムに続けて、ゲームオーバーのプログラムを追加します。

```
54 '--- ミス ---
55 SPANIM 0,3,10,3424,10,3425,10,3426,
10,3427,1
56 PL=PL-1
57 LOCATE 30,0
58 PRINT "LEFT ";PL
59 VSYNC 120
60 IF PL>0 THEN GOTO @START
61
62 '--- ゲーム オーバー ---
63 LOCATE 16,14
64 COLOR 3
65 PRINT "XXX GAME OVER XXX"
66 COLOR 15
67 END
68
69 '--- ビーム ハッシャ ---
```

もし自機の残り数があるなら、再スタート
文字の色を赤にする
ゲームオーバー表示
文字の色を白にもどす
プログラムを終了する

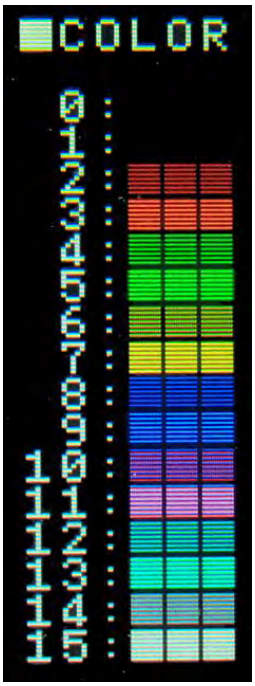
プログラムを実行してみましょう。
ミスして自機の残り数が 0 になると、
ゲームオーバーになって、
プログラムが終了します。

64 行の **COLOR**（カラー）命令は、文字の色を変える命令です。

```
COLOR      3          ,0
          文字色      背景色
```

文字色	文字の色。0～15 で指定。
背景色	文字の背景の色。0～15 で指定。省略可能。

色の番号はこのようになっています。
（SMILE BASIC メニューで確認できます）
0 は透明（背景の色がすけて見える）、
1 は黒です。



今回は文字色を赤にして、「GAME OVER」を表示しています。
そのままだと、プログラムを終了した後も文字が赤のままで見づらいので、66 行で白にもどしています。

67 行の **END**（エンド）命令は、プログラムを終了する命令です。

★プログラムを「SHOOT7」の名前で保存しましょう。