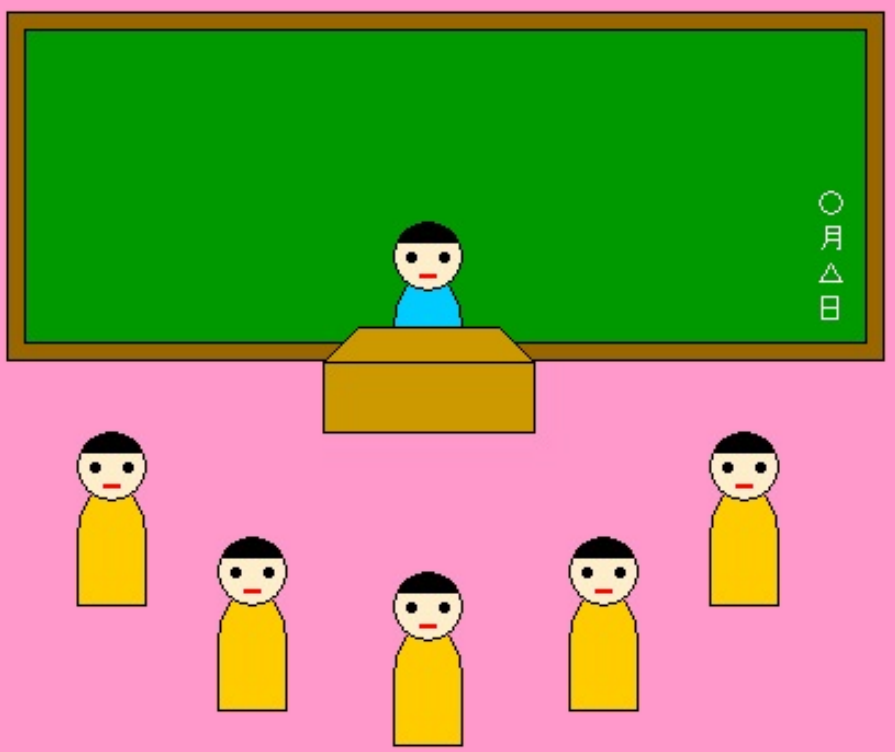
ＨＳＰ講座シリーズ

ゲーム製作のミニ講座

シューティング・ゲームを作ろう



科学太郎

HSPプログラミングで
世界に１つのゲームソフトを
製作してみよう！！

ゲームソフト製作は自分自身で拡張できる

Windows XP/7 対応
Windows Vista 対応

◆◆◆ はじめに ◆◆◆

1. ミニ講座とは
2. 考え方
3. 進め方
4. 目次

ミニ講座とは

このミニ講座は、フリーソフトで出回っている「HSP(Hot Soup Processor)」で、簡単なシューティング・ゲームを作成することを目的にしてます。元々は「プログラミングのメモ帳」というブログで掲載してる「［HSP］シューティング・ゲームのミニ講座」を電子書籍にしたものです。ブログと異なり、なるべくモノクロで統一しつつ、フローチャート図を多く載せてみました。その他、HSPの基本命令の使い方も所々で追加してます。そのため HSP でプログラミングを始めたばかりの入門者でも分かるようになってます。スクリプト言語の HSP については、公式ホームページの「HSPTV!」をどうぞ。

↑クリックすると公式ホームページへ移動します。

なお、この電子書籍と公式ホームページとは、何の関係もございません。今後とも電子書籍版のミニ講座もどうぞ。

考え方

シューティング・ゲームの基本的な考え方は、次の図のようになります。

進め方

ミニ講座の進め方は、自機、弾丸、敵機、当たり判定、...、完成という形で徐々に付け足していきます。この方法で簡単なゲーム作りの方法を学べると思います。付け足した部分は色を変えてハイライト部分としてます。ハイライト部分以外は、どこも修正してません。HSP付属のスクリプト・エディタを起動して、サンプルのハイライト部分を徐々に入力して行けば良いでしょう。入力が終わるたびに「F5」キーを押して実行結果を確認するように心がけましょう。

目次

◆◆◆ 第1章 自機の表示 ◆◆◆

1. 自機のアニメ
2. 繰り返し命令
3. ちらつき防止
4. 自機の表示

1.1 自機のアニメ

自機のアニメーション処理は、シューティング・ゲームだけではなく全ての基本です。まずは次のソースを試してみましょう。

リスト(1.1.1)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    color $00,$00,$00:boxf
    color $00,$FF,$00
    font MSGOTHIC,50
    ;アニメ処理
    pos   0,200:mes "山":await 100
    pos  50,200:mes "山":await 100
    pos 100,200:mes "山":await 100
    pos 150,200:mes "山":await 100
    pos 200,200:mes "山":await 100
    pos 250,200:mes "山":await 100
    pos 300,200:mes "山":await 100
    pos 350,200:mes "山":await 100
    pos 400,200:mes "山":await 100
    pos 450,200:mes "山":await 100
    pos 500,200:mes "山":await 100
    pos 550,200:mes "山":await 100
stop
```

0.1秒毎に自機を表す「山」文字が左側から右側に連続して描かれました。しかし、残像のように「山」文字が残り、綺麗なアニメーション処理とは見えませんね。これを防ぐには「山」文字を描く前に画面を消去すれば良いのです。

リスト(1.1.2)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    ;アニメ処理
    color:boxf:color $00,$FF,$00:pos   0,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos  50,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 100,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 150,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 200,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 250,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 300,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 350,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 400,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 450,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 500,200:mes "山":await 100
    color:boxf:color $00,$FF,$00:pos 550,200:mes "山":await 100
stop
```

上記の color:boxf が画面を消去する部分です。ここで color 命令の３つの引数が省略されてますが、これは color 0,0,0 または color $00,$00,$00 と同じになります。また、boxf 命令も４つの引数が省略されてますが、これは画面全体を表す boxf 0,0,600,400 と同じです。詳しくはHSPスクリプト・エディタ上で color 命令にカーソルを合わせて[F1]キーを押します。これでHSPのマニュアルが起動して命令の詳しい指定方法などが確認できますよ。

1.2 繰り返し命令

さて、リスト(1.1.2)のソースを見るとpos命令の座標位置以外は同じですね。このような場合は、座標を表す部分を変数に置き換え、共通部分を繰り返し命令の内部に書くのが普通です。それでは書き直してみましょう。

リスト(1.2.1)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    ;アニメ処理
    x=0:y=200
    repeat 12
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        await 100
        x=x+50
    loop
stop
```

これで同じような処理を複数回書かなくても良くなりました。またrepeat命令の繰り返しにより、アニメ処理の部分が分かりやすくもなりましたね。さらにHSPのプログラミングらしく見えてきます。(笑)

なお、このアニメ処理は次のような構造になります。(フローチャート)

1. スクリーンの初期化
2. フォントの設定
3. アニメ処理の初期化(x=0、y=200)
4. アニメ処理を12回だけ(a)～(d)まで繰り返す
    a. 画面の消去
    b. 自機の描画
    c. ウェイトの処理
    d. 座標の計算
5. プログラムの停止

上記の方法でアニメ処理は行われますが 12 回して「山」文字が右側に来ると停止します。これはrepeat命令で回数を 12 と指定してるからです。そこで、今度は無限ループにして、右端から「山」文字が消えたら左端から現れるように改良します。また、移動ドット数も 50 ドットから 5 ドットに減らして、ウエイト時間も 20 ミリ秒(0.02秒)に短くしてみました。

リスト(1.2.2)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    ;アニメ処理
    x=0:y=200
    repeat ←修正
        color $00,$00,$00:boxf
```

```
        color $00,$FF,$00:pos x,y:mes "山"
        await 20 ←修正
        x=x+5:if(x>=600):x=0 ←修正
    loop
stop
```

どうですか？かなり滑らかに左から右に移動してますよね。これこそがアニメーション処理の基本です。

1.3 ちらつき防止

それでは、どれぐらい高速な移動が可能なのでしょうか。一番早くするには、ウエイト時間を最小の 0 にすることでしょうね。また、移動ドット数も最小値の 1 ドットにしてみます。それでは試してみましょう。

リスト(1.3.1)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    ;アニメ処理
    x=0:y=200
    repeat
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        await 0 ←修正
        x=x+1:if(x>=600):x=0 ←修正
    loop
stop
```

どうですか？この速度がHSPでアニメーション処理を行う最高速度の限界です。早いですね。しかし、何だかちらつきませんか？ちらつかない場合は、[F5]キーをもう一度押して、２つのウインドウを実行してみましょう。このちらつきが発生するのは、画面を消去して「山」文字を描いてますよね。ここの部分で「山」文字がチョー短い時間で消えてるのが原因です。これを防ぐには、ウインドウの再描画を制御する必要があり次のようにします。

1. 再描画の停止 ←注目
2. 画面の消去
3. 自機の描画
4. 再描画の開始 ←注目
5. ウェイトの処理
6. 座標の計算

ウインドウの再描画の停止と開始は redraw 命令で行います。この命令を使ってちらつきが防止できるか試してみましょう。

リスト(1.3.2)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    ;アニメ処理
    x=0:y=200
    repeat
        redraw 0 ←追加
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        redraw 1 ←追加
        await 0
        x=x+1:if(x>=600):x=0
    loop
stop
```

上記の redraw 0 が再描画の停止で redraw 1 が再描画の開始です。前回のちらつきバージョンと今

回のちらつき防止バージョンを比べてみて下さい。どうですか？２つのウインドウを見比べれば、明らかに今回の方がちらつきがなく滑らかですよね。このちらつき防止対策が、アニメーション処理の一番重要な部分です。

1.4 自機の表示

それでは第1章の「自機の表示」ソースを紹介します。

リスト(1.4.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第1章「自機の表示」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    repeat
        redraw 0
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        redraw 1
        await (1000/60)
    loop
stop

//-------------------------------------------------------------------------------
// End of lesson-1.hsp
//-------------------------------------------------------------------------------
```

上記のソースは、自機の表示のみですから座標計算の x=x+1:if(x>=600):x=0 の行は記述してません。また、ウエイト時間が await (1000/60) となってますが、これはゲームのフレーム・レートが1秒間に 60 回を意味してます。多くのゲームは、画面(ディスプレイ)の更新周期の 60 回に合わせて滑らかに表示できるようにしてます。ここでは分かりやすく (1000/60) という数式にしてます。

なお、HSPの標準命令の await 命令では正確な 60 フレームにすることは出来ません。これは await 命令の引数に整数しか渡せず 1000÷60＝16.6666 という小数を扱えないからです。正確な 60 フレーム(60FPS)にするには工夫するか、DirectX の命令である HSPDXFIX と呼ばれるプラグインを使います。しかし、今回はミニ講座であるため標準命令だけでシューティング・ゲームを作成するので利用しませんよ。これで「自機の表示」は終わります。次章はキーボードで「自機の操作」を紹介します。

◆◆◆ 第2章 自機の操作 ◆◆◆

1. キーボードの検出
2. stick命令のテスト
3. 自機の移動
4. 画面外の補正
5. 自機の操作

2.1 キーボードの検出

自機を移動する方法は、キーボード、マウス、ジョイパッドなどがあります。今回は処理を簡単にするため、キーボード操作のみに限定します。HSPにはキーボードの入力チェックに getkey 命令、stick 命令の２つが用意されてます。どちらも押されてるキーボードをチェックできますが、１回の呼び出しで必要キーの全てを取得できる stick 命令を使います。この命令は、カーソル・キー(4個)、特殊キー(5個)、マウス・ボタン(2個)を１回の呼び出しでチェックできます。特殊キーは、ゲームで良く使われる[SPC]キー、[ENTER]キー、[CTRL]キー、[ESC]キー、[TAB]キーが含まれます。この命令は、押されてるか、押されてないかを０と１のビットデータで返します。
チェックする場合は、ビットデータを２進数(バイナリ値)の考えで０と１を調べます。まずは、次の一覧をご覧ください。
表(2.1.1)

押されてるキー		変数の値		
種類	キーボード	2進表記	10進表記	16進表記
カーソル・キー 「左上/右下」と覚えよう	[←]キー(左)	%00000000001	1	$001
	[↑]キー(上)	%00000000010	2	$002
	[→]キー(右)	%00000000100	4	$004
	[↓]キー(下)	%00000001000	8	$008
特殊キー 「SOURCE」(ソース)と覚えよう	[SPC]キー(Space)	%00000010000	16	$010
	[RET]キー(Enter)	%00000100000	32	$020
	[CTRL]キー(Control)	%00001000000	64	$040
	[ESC]キー(Escape)	%00010000000	128	$080
マウス・ボタン 「左右」と覚えよう	マウスの[左]ボタン	%00100000000	256	$100
	マウスの[右]ボタン	%01000000000	512	$200
特殊キー	[TAB]キー(Tab)	%10000000000	1024	$400

上記の一覧は[←]キー、[↑]キー、[SPC]キーの３つが同時に押されてると次の値が変数に代入されてます。

- 2進数で解釈すると「%00000010011」の値が変数にセットされる。
- 10進数で解釈すると「19」の値が変数にセットされる。
- 16進数で解釈すると「$013」の値が変数にセットされる。

上記の「19」とは、「1+2+16」の値ですよ。この「19」を2進数で表現すると「%00000010011」、16進数で表現すると「$013」となるのです。チェックするときは、ビット論理積の「&」演算子、または「and」マクロ演算子を使います。等号の「=」演算子を使うと１つしかキーが押されてない時しかチェックできません。この方法が意図的なものなら良いのですが、通常は複数のキーをチェックするので「=」演算子は使いません。

2.2 stick命令のテスト

それでは、以下のサンプルを試してみましょう。カーソル・キー([←]・[↑]・[→]・[↓])、特殊キー([SPC]・[RET]・[CTRL]・[ESC]・[TAB])、[左]ボタン、[右]ボタンを押すことで表示が変わります。同時に複数のキーを押すと複数の表示になりますよね。これで複数のキーをチェックできます。

リスト(2.2.1)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,20
    repeat
        redraw 0
        color $00,$00,$00:boxf
        color $00,$FF,$00
        stick key,$7FF
        if(key & %00000000001):pos 0,  0:mes "   1:[←]キー"
        if(key & %00000000010):pos 0, 20:mes "   2:[↑]キー"
        if(key & %00000000100):pos 0, 40:mes "   4:[→]キー"
        if(key & %00000001000):pos 0, 60:mes "   8:[↓]キー"
        if(key & %00000010000):pos 0, 80:mes "  16:[SPC]キー"
        if(key & %00000100000):pos 0,100:mes "  32:[RET]キー"
        if(key & %00001000000):pos 0,120:mes "  64:[CTRL]キー"
        if(key & %00010000000):pos 0,140:mes " 128:[ESC]キー"
        if(key & %00100000000):pos 0,160:mes " 256:[左]ボタン"
        if(key & %01000000000):pos 0,180:mes " 512:[右]ボタン"
        if(key & %10000000000):pos 0,200:mes "1024:[TAB]キー"
        redraw 1
        await (1000/60)
    loop
stop
```

上記のソースでは、key 変数のチェックに2進数を使ってます。その他にも10進数、16進数でも同じことが出来ます。

リスト(2.2.2)…16進数の記述

```
stick key,$7FF
if(key & $001):pos 0,  0:mes "   1:[←]キー"
if(key & $002):pos 0, 20:mes "   2:[↑]キー"
if(key & $004):pos 0, 40:mes "   4:[→]キー"
if(key & $008):pos 0, 60:mes "   8:[↓]キー"
if(key & $010):pos 0, 80:mes "  16:[SPC]キー"
if(key & $020):pos 0,100:mes "  32:[RET]キー"
if(key & $040):pos 0,120:mes "  64:[CTRL]キー"
if(key & $080):pos 0,140:mes " 128:[ESC]キー"
if(key & $100):pos 0,160:mes " 256:[左]ボタン"
if(key & $200):pos 0,180:mes " 512:[右]ボタン"
if(key & $400):pos 0,200:mes "1024:[TAB]キー"
```

リスト(2.2.3)…10進数の記述

```
stick key,$7FF
if(key &    1):pos 0,  0:mes "   1:[←]キー"
if(key &    2):pos 0, 20:mes "   2:[↑]キー"
if(key &    4):pos 0, 40:mes "   4:[→]キー"
if(key &    8):pos 0, 60:mes "   8:[↓]キー"
if(key &   16):pos 0, 80:mes "  16:[SPC]キー"
if(key &   32):pos 0,100:mes "  32:[RET]キー"
if(key &   64):pos 0,120:mes "  64:[CTRL]キー"
if(key &  128):pos 0,140:mes " 128:[ESC]キー"
if(key &  256):pos 0,160:mes " 256:[左]ボタン"
```

```
if(key &  512):pos 0,180:mes " 512:[右]ボタン"
if(key & 1024):pos 0,200:mes "1024:[TAB]キー"
```

また、一般的にはカーソル・キー部分を次のように記述してます。

リスト(2.2.4)…一般的な記述

```
stick key,15
if(key&1):[←]キーの処理
if(key&2):[↑]キーの処理
if(key&4):[→]キーの処理
if(key&8):[↓]キーの処理
```

それから stick 命令の第２引数は、非トリガー・タイプのキーをビットデータで指定します。この非トリガー・タイプとは、押しっぱなし状態でもキーボードが押されてるか検出できる点です。そのためカーソル・キーなどの場合は、押しっぱなし状態でも検出可能にするために 15 を指定しましょう。この 15 の値は 1+2+4+8 の合計で10進数を意味し、2進数ならば「%1111」、16進数ならば「$F」となります。なお、自機ショット(弾丸)の発射は、キーボードを押した瞬間だけ単発で打ちたいことがあります。このような場合にはトリガー・タイプですから第２引数の非トリガー・タイプを指定しません。つまり、指定しないとキーボードを押した瞬間の１回だけを検出するのが基本となります。

2.3 自機の移動

それでは本題のカーソル・キーによる自機の移動を考えてみます。全体的には、次のような感じです。

1. [←]キーが押されてるかチェック → 横軸の変数(x)を減少
2. [→]キーが押されてるかチェック → 横軸の変数(x)を増加
3. [↑]キーが押されてるかチェック → 縦軸の変数(y)を減少
4. [↓]キーが押されてるかチェック → 縦軸の変数(y)を増加

リスト(2.3.1)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    repeat
        redraw 0
        stick key,%1111 ┐
        if(key&1):x-=5  │
        if(key&2):y-=5  ├追加
        if(key&4):x+=5  │
        if(key&8):y+=5  ┘
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        redraw 1
        await (1000/60)
    loop
stop
```

どうですか？これで自機が画面を自由自在に動き回れますね。しかし、ずっと移動し続けると画面から消えてしまいます。そこで画面外に移動したら画面内に座標を戻す処理を追加してみましょう。

2.4 画面外の補正

リスト(2.4.1)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    repeat
        redraw 0
        stick key,%1111
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        redraw 1
        await (1000/60)
    loop
stop
```

修正

上記の[←]キーの処理だけを考えると次のようになります。

1. [←]キーが押されてるかチェック
2. 横軸の変数(x)を減少
3. 横軸の変数(x)が画面左端から消えたかチェック
4. 横軸の変数(x)に画面左端を意味する 0 をセット

全体のフローチャートは次のようになります。

1. [←]キーが押されてるかチェック
    a. 横軸の変数(x)を減少
    b. 横軸の変数(x)が画面左端から消えたかチェック
    c. 横軸の変数(x)に画面左端を意味する 0 をセット
2. [↑]キーが押されてるかチェック
    a. 縦軸の変数(y)を減少
    b. 縦軸の変数(y)が画面上部から消えたかチェック
    c. 縦軸の変数(y)に画面上部を意味する 0 をセット
3. [→]キーが押されてるかチェック
    a. 横軸の変数(x)を増加
    b. 横軸の変数(x)が画面右端から消えたかチェック
    c. 横軸の変数(x)に画面右端を意味する 550 をセット
4. [↓]キーが押されてるかチェック
    a. 縦軸の変数(y)を増加
    b. 縦軸の変数(y)が画面下部から消えたかチェック
    c. 縦軸の変数(y)に画面下部を意味する 350 をセット

2.5 自機の操作

それでは第2章の「自機の操作」ソースを紹介します。

リスト(2.5.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第2章「自機の操作」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    repeat
        redraw 0
        stick key,%1111
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        redraw 1
        await (1000/60)
    loop
stop

//-------------------------------------------------------------------------------
// End of lesson-2.hsp
//-------------------------------------------------------------------------------
```

上記のソースで x-=5 という記述は x=x-5 と同じ処理です。これは変数(x)の内容を -5 するという意味で、C/C++言語の複合代入と呼ばれる演算子です。HSPでも、この省略された記述が許されてるので便利に使いましょう。それから 600 ではなく 550 で比較してるのは、自機を表す「山」文字の左上隅が座標になってます。そのため自機の横サイズである 50 を画面の横サイズ 600 から引いた 550 という値で比較するのです。同様に、自機の縦サイズである 50 を画面の縦サイズ 400 から引いた 350 という値で比較します。これで「自機の操作」は終わります。次章はスペース・キーで「弾丸の発射」を紹介します。

◆◆◆ 第3章 弾丸の発射 ◆◆◆

1. 弾丸の管理
2. 弾丸の処理
3. 弾丸の発生
4. 弾丸の描画
5. 弾丸の発射
6. 弾丸の連射
7. 弾丸の間隔
8. 弾丸の連射

3.1 弾丸の管理

自機の弾丸は、最大10個発射できることを考えてみます。まずは弾丸の横軸(x)、縦軸(y)は絶対に必要なので、この２つを配列で管理します。さらに弾丸の配列が、画面内を移動中(有効)か、画面外に外れた(無効)かを判定するフラグ(flag)も必要になりますね。それでは３つを配列で管理すると次のようになります。

リスト(3.1.1)

```
dim tamaF,10
dim tamaX,10
dim tamaY,10
```

上記の３つの配列は次の意味があります。

- tamaF は、弾丸の有無フラグで 0 なら無効、1 なら有効を意味
- tamaX は、弾丸の横軸で 0 ～ 550 の値を管理(弾丸の横サイズが 50 の場合)
- tamaY は、弾丸の縦軸で 0 ～ 350 の値を管理(弾丸の縦サイズが 50 の場合)

3.2 弾丸の処理

弾丸を処理するには、弾丸の発射、移動、描画の最低でも３つの処理が必要です。今回は、弾丸の発生部(TamaBirth)、弾丸の描画部(TamaDraw)の２つをサブルーチンにします。なお、弾丸の移動は描画部(TamaDraw)処理と一緒に記述して、サブルーチンの数を減らしてみます。それでは前章のリスト(2.5.1)ソースに追加してみましょう。

リスト(3.2.1)

```
*Init            ┐
    dim tamaF,10  ├追加
    dim tamaX,10  │
    dim tamaY,10  ┘
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    repeat
        redraw 0
        stick key,%1111
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth ←追加
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        gosub *TamaDraw ←追加
        redraw 1
        await (1000/60)
    loop
    stop
*TamaBirth  ┐
    :       │
    return  ├追加
*TamaDraw   │
    :       │
    return  ┘
```

とりあえず、上記の４カ所を追加します。

3.3 弾丸の発生

それでは、弾丸の発生ルーチン(TamaBirth)を考えてみましょう。弾丸の発生手順は、次のようになると思います。

1. tamaF 配列から弾丸情報が無効である場所を探す
2. 無効領域が見つかったら弾丸を発生(a)～(c)を行う
    a. 弾丸の有無フラグに 1 をセット
    b. 弾丸の横軸(x)に自機の x をセット
    c. 弾丸の縦軸(y)に自機の y をセット
3. 繰り返しを抜ける

リスト(3.3.1)

```
*TamaBirth
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    return
```

上記のように tamaF 配列の分だけ繰り返す場合は repeat 命令よりも foreach 命令が便利です。たったのこれだけで、無効領域に有効な弾丸情報をセットして、新しい弾丸を発生できます。どうですか？以外にも簡単ですね。これで「弾丸の発生」は終わりですよ。

3.4 弾丸の描画

続いて、弾丸の描画ルーチン(TamaDraw)を考えてみましょう。こちらは、弾丸の移動も含まれますので、先に移動処理、続いて描画処理を行います。

1. tamaF 配列から弾丸情報が有効である場所を探す
2. 有効領域が見つかったら弾丸の移動(a)～(d)を行う
    a. 弾丸の縦軸(y)から 8 ドット引く
    b. 弾丸の縦軸(y)が画面外に外れたかチェック
    c. 画面外に外れた場合は、弾丸の有無フラグに 0 をセット
    d. 次の繰り返しへジャンプ(continue)
3. 弾丸の描画位置を設定(pos)
4. 弾丸の描画色を設定(color)
5. 弾丸の文字を描画(mes)

リスト(3.4.1)

```
*TamaDraw
    foreach tamaF
        if tamaF(cnt){
            tamaY(cnt)-=8:if(tamaY(cnt)<-50):tamaF(cnt)=0:continue
            pos tamaX(cnt),tamaY(cnt)
            color $FF,$FF,$00:mes ":"
        }
    loop
    return
```

上記のソースで continue 命令がありますが、これを実行すると次の繰り返し処理にジャンプします。そのため次の行の pos 命令、color 命令、mes 命令は実行されません。これで「弾丸の移動」＋「弾丸の描画」は終わりですね。

3.5 弾丸の発射

それでは第3章の「弾丸の発射」ソースを紹介します。

リスト(3.5.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第3章「弾丸の発射」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Init
    ;弾丸の管理
    dim tamaF,10
    dim tamaX,10
    dim tamaY,10
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    repeat
        redraw 0
        stick key,%1111
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        gosub *TamaDraw
        redraw 1
        await (1000/60)
    loop
    stop
//--------------------------------------
// 弾丸の発生
//--------------------------------------
*TamaBirth
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    return
//--------------------------------------
// 弾丸の描画
//--------------------------------------
*TamaDraw
    foreach tamaF
        if tamaF(cnt){
            tamaY(cnt)-=8:if(tamaY(cnt)<-50):tamaF(cnt)=0:continue
            pos tamaX(cnt),tamaY(cnt)
            color $FF,$FF,$00:mes ":"
        }
    loop
    return
//-------------------------------------------------------------------------------
// End of lesson-3.hsp
//-------------------------------------------------------------------------------
```

上記のソースを実行すると[SPACE]キーを押すことで、１発だけ弾丸が発射されますね。これは stick 命令で[SPACE]キーがトリガー・タイプなので単発なのです。

3.6 弾丸の連射

それでは連射する場合は、どうすれば良いでしょうか。とりあえず、単純に[SPACE]キーを非トリガー・タイプにしてみましょう。

リスト(3.6.1)

```
repeat
    redraw 0
    stick key,%11111 ←修正
    if(key&1):x-=5:if(x<0):x=0
    if(key&2):y-=5:if(y<0):y=0
    if(key&4):x+=5:if(x>550):x=550
    if(key&8):y+=5:if(y>350):y=350
    if(key&16):gosub *TamaBirth
    color $00,$00,$00:boxf
    color $00,$FF,$00:pos x,y:mes "山"
    gosub *TamaDraw
    redraw 1
    await (1000/60)
loop
```

変更箇所は１カ所です。メイン部の stick 命令の第２引数を %1111 から %11111 に変更してみました。それでは、実行して見ましょう。どうですか？期待通りでしたか。なぜかレーザのような弾丸が一気に発射されちゃいましたね。あらら。何これ。という感じでしょうか。このように一気に弾丸が発射されるのは[SPACE]キーが押しっぱなしの場合は、連続して発射されてるからですよ。だって連射だもの。ね。おかしくはないでしょう。でも、期待通りではないよね。

3.7 弾丸の間隔

それでは、期待通りの連射とは何か。考えてみましょう。まずは、１個おきに弾丸が発射されるように改良しましょう。改良する部分は TamaBirth のサブルーチンのみで行います。

リスト(3.7.1)

```
*TamaBirth
    if(tamaTrigg):tamaTrigg--:return ←追加
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    tamaTrigg=2 ←追加
    return
```

上記の２カ所だけ追加します。これで tamaTrigg 変数が 0 以外の場合は、デクリメントしてサブルーチンを終わります。そして tamaTrigg 変数が 0 のみの場合は、弾丸を１発だけ発射しますね。弾丸の発射が終わったら tamaTrigg 変数に 2 をセットします。これで2サイクルで1回だけ弾丸が発射されるわけです。でも、まだ、まだ、発射間隔が短すぎて一気に10発も発射されてますね。それでは tamaTrigg 変数に 3、4、5、… 60 と発射間隔を多くしてみましょう。5 〜 10 当たりが丁度良さそうですが 15 もお気に入りの間隔です。なお、60 にすると１秒間に１回しか弾丸が発射されず期待外れになります。この発射間隔は、お好みで設定して欲しいと思います。とりあえず、ミニ講座では 8 に設定しておきますよ。

3.8 弾丸の連射

それでは第3章の「弾丸の連射」ソースを紹介します。

リスト(3.8.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第3章「弾丸の発射」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Init
    ;弾丸の管理
    dim tamaF,10
    dim tamaX,10
    dim tamaY,10
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    repeat
        redraw 0
        stick key,%11111
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        gosub *TamaDraw
        redraw 1
        await (1000/60)
    loop
    stop
//--------------------------------------
// 弾丸の発生
//--------------------------------------
*TamaBirth
    if(tamaTrigg):tamaTrigg--:return
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    tamaTrigg=8
    return
//--------------------------------------
// 弾丸の描画
//--------------------------------------
*TamaDraw
    foreach tamaF
        if tamaF(cnt){
            tamaY(cnt)-=8:if(tamaY(cnt)<-50):tamaF(cnt)=0:continue
            pos tamaX(cnt),tamaY(cnt)
            color $FF,$FF,$00:mes ":"
        }
    loop
    return
//-------------------------------------------------------------------------------
```

```
// End of lesson-3.hsp
//-----------------------------------------------------------------------------
```

これで「弾丸の発射」は終わります。次章は自動的に敵機が発生する「敵機の表示」を紹介します。

◆◆◆ 第4章 敵機の表示 ◆◆◆

1. 敵機の管理
2. 敵機の処理
3. 敵機の発生
4. 敵機の描画
5. 敵機の表示
6. 発生の間隔
7. 敵機の表示

4.1 敵機の管理

敵機は、最大20匹発生できることを考えてみます。まずは敵機の横軸(x)、縦軸(y)は絶対に必要なので、この２つを配列で管理します。さらに敵機の配列が、画面内を移動中(有効)か、画面外に外れた(無効)かを判定するフラグ(flag)も必要になりますね。それでは３つを配列で管理すると次のようになります。

リスト(4.1.1)

```
dim enemyF,20
dim enemyX,20
dim enemyY,20
```

上記の３つの配列は次の意味があります。

- enemyF は、敵機の有無フラグで 0 なら無効、1 なら有効を意味
- enemyX は、敵機の横軸で 0 ～ 550 の値を管理(敵機の横サイズが 50 の場合)
- enemyY は、敵機の縦軸で 0 ～ 350 の値を管理(敵機の縦サイズが 50 の場合)

4.2 敵機の処理

敵機を処理するには、敵機の発生、移動、描画の最低でも３つの処理が必要です。今回は、敵機の発生部(EnemyBirth)、敵機の描画部(EnemyDraw)の２つをサブルーチンにします。なお、敵機の移動は描画部(EnemyDraw)処理と一緒に記述して、サブルーチンの数を減らしてみます。それでは第2章の「自機の操作」ソースに追加してみましょう。

リスト(4.2.1)

```
*Init              ┐
    dim enemyF,20  ├追加
    dim enemyX,20  │
    dim enemyY,20  ┘
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize ←追加
    repeat
        redraw 0
        stick key,%1111
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        gosub *EnemyBirth ←追加
        gosub *EnemyDraw  ←追加
        redraw 1
        await (1000/60)
    loop
    stop
*EnemyBirth ┐
    :       │
    return  ├追加
*EnemyDraw  │
    :       │
    return  ┘
```

とりあえず、上記の11カ所を追加します。

4.3 敵機の発生

それでは、敵機の発生ルーチン(EnemyBirth)を考えてみましょう。敵機の発生手順は、次のようになると思います。

1. enemyF 配列から敵機情報が無効である場所を探す
2. 無効領域が見つかったら敵機を発生(a)～(c)を行う
    a. 敵機の有無フラグに 1 をセット
    b. 敵機の横軸(x)に 0 ～ 550 をセット(乱数で決定)
    c. 敵機の縦軸(y)に -50 をセット
3. 繰り返しを抜ける

リスト(4.3.1)

```
*EnemyBirth
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            break
        }
    loop
    return
```

上記のように enemyF 配列の分だけ繰り返す場合は repeat 命令よりも foreach 命令が便利です。たったのこれだけで、無効領域に有効な敵機情報をセットして、新しい敵機を発生できます。どうですか？前章の「弾丸の発射」とほぼ同じですね。つまり、弾丸の発生、敵機の発生、流星の発生などは、同じ要領で管理できるのです。これで「敵機の発生」は一通り終わりです。

4.4 敵機の描画

続いて、敵機の描画ルーチン(EnemyDraw)を考えてみましょう。こちらは、敵機の移動も含まれますので、先に移動処理、続いて描画処理を行います。

1. enemyF 配列から敵機情報が有効である場所を探す
2. 有効領域が見つかったら敵機の移動(a)～(d)を行う
    a. 敵機の縦軸(y)に 3 ドット足す
    b. 敵機の縦軸(y)が画面外に外れたかチェック
    c. 画面外に外れた場合は、敵機の有無フラグに 0 をセット
    d. 次の繰り返しへジャンプ(continue)
3. 敵機の描画位置を設定(pos)
4. 敵機の描画色を設定(color)
5. 敵機の文字を描画(mes)

リスト(4.4.1)

```
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            enemyY(cnt)+=3:if(enemyY(cnt)>400):enemyF(cnt)=0:continue
            pos enemyX(cnt),enemyY(cnt)
            color $00,$FF,$FF:mes "Ж"
        }
    loop
    return
```

上記のソースで continue 命令がありますが、これを実行すると次の繰り返し処理にジャンプします。そのため次の行の pos 命令、color 命令、mes 命令は実行されません。これで「敵機の移動」＋「敵機の描画」は終わりですね。

4.5 敵機の表示

リスト(4.5.1)

```
//-------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//=========================================================================
// 第4回「敵機の表示」by 科学太郎
//-------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Init
    ;敵機の管理
    dim enemyF,20
    dim enemyX,20
    dim enemyY,20
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%1111
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        gosub *EnemyBirth
        gosub *EnemyDraw
        redraw 1
        await (1000/60)
    loop
    stop
//--------------------------------------
// 敵機の発生
//--------------------------------------
*EnemyBirth
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            break
        }
    loop
    return
//--------------------------------------
// 敵機の描画
//--------------------------------------
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            enemyY(cnt)+=3:if(enemyY(cnt)>400):enemyF(cnt)=0:continue
            pos enemyX(cnt),enemyY(cnt)
            color $00,$FF,$FF:mes "Ж"
        }
    loop
    return
//-------------------------------------------------------------------------
// End of lesson-4.hsp
//-------------------------------------------------------------------------
```

上記のソースを実行すると一度に20匹の敵機が発生しますね。これは敵機が発生するタイミングを連続して誕生させてるからです。前章の「弾丸の発射」の連射と同じ現象です。

4.6 発生の間隔

それでは前章の「弾丸の発射」を参考に敵機の発生間隔を組み込んでみましょう。とりあえず10サイクルに1匹の敵機が発生することにしましょう。改良する部分は EnemyBirth のサブルーチンですよ。

リスト(4.6.1)

```
*EnemyBirth
    if(enemyCycle):enemyCycle--:return ←追加
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            break
        }
    loop
    enemyCycle=10 ←追加
    return
```

上記の 2 カ所だけ追加します。これで enemyCycle 変数が 0 以外の場合は、デクリメントしてサブルーチンを終わります。そして enemyCycle 変数が 0 のみの場合は、敵機が 1 匹だけ誕生しますね。敵機の誕生が終わったら enemyCycle 変数に 10 をセットします。これで10サイクルで1匹だけ敵機が発生するわけです。でも、まだ、発生間隔が短すぎて大量に敵機が発生してますね。これでは、ゲームの難易度が高くなってしまいます。しかし、この発生間隔を変えることで「ゲームの難易度」を変えることが可能かもしれません。それでは enemyCycle 変数に 20、30、40、… 60 と発生間隔を多くしてみましょう。30 〜 40 当たりが丁度良さそうですが 60 の 1 秒間に 1 匹の敵機が発生してもおかしくありませんね。この発生間隔は、お好みで設定して欲しいと思います。とりあえず、ミニ講座では 30 に設定しておきますよ。

4.7 敵機の表示

それでは第4章の「敵機の表示」ソースを紹介します。

リスト(4.7.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第4章「敵機の表示」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Init
    ;敵機の管理
    dim enemyF,20
    dim enemyX,20
    dim enemyY,20
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%1111
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
        gosub *EnemyBirth
        gosub *EnemyDraw
        redraw 1
        await (1000/60)
    loop
    stop
//--------------------------------------
// 敵機の発生
//--------------------------------------
*EnemyBirth
    if(enemyCycle):enemyCycle--:return
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            break
        }
    loop
    enemyCycle=30
    return
//--------------------------------------
// 敵機の描画
//--------------------------------------
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            enemyY(cnt)+=3:if(enemyY(cnt)>400):enemyF(cnt)=0:continue
            pos enemyX(cnt),enemyY(cnt)
            color $00,$FF,$FF:mes "Ж"
        }
    loop
    return
```

```
//-----------------------------------------------------------------------------
// End of lesson-4.hsp
//-----------------------------------------------------------------------------
```

これで「敵機の表示」は終わります。次章は自機、敵機、自機弾の「当たり判定」を紹介します

。

◆◆◆ 第5章 当たり判定 ◆◆◆

1. 当たり判定とは
2. 敵機の当たり判定
3. 自機のサブルーチン
4. 当たり判定の考え
5. 自機の衝突判定
6. 自機弾の衝突判定
7. 当たり判定

5.1 当たり判定とは

まず最初に「当たり判定」とは、キャラクタ同士の衝突判定の事です。今回は、ミニ講座であるため敵機弾、アイテム、背景などの当たり判定は考えないモノとします。そうなると登場するキャラクタは、自機、自機弾、敵機の３つです。この３つは、次のようなときに衝突しますよね。

1. 敵機が移動した時に、自機と衝突
2. 敵機が移動した時に、自機弾と衝突

ここでの注意点は、自機弾は、複数個の衝突判定が必要という事です。そこで分かりやすくユーザ定義関数を作成します。用意すれば良さそうな衝突判定の関数は次の２つになります。

1. 自機の１つと衝突判定を行う(FightCrash)
2. 自機弾の全てと衝突判定を行う(TamaCrash)

上記のカッコ内は、ユーザ定義関数の名前です。この２つを用意することで「当たり判定」の処理を手軽に書けます。

リスト(5.1.1)

```
foreach enemyF
    if enemyF(cnt){
        if FightCrash(enemyX(cnt),enemyY(cnt)):enemyF(cnt)=0:continue    ;自機と衝突
        if  TamaCrash(enemyX(cnt),enemyY(cnt)):enemyF(cnt)=0:continue    ;自機弾と衝突
    }
loop
```

どうですか？上記のように書くことで当たり判定が、スッキリ見やすくなりましたね。なお、FightCrash 関数も TamaCrash 関数も衝突したら 1 を戻します。この 1 が返されたならば、敵機も消滅しなければならないので enemyF(cnt)=0 と有無フラグに 0 をセットしてます。その後は、次の繰り返し処理にジャンプさせるために continue 命令を使ってます。

5.2 敵機の当たり判定

それでは、当たり判定の部分をどこに書けばよいでしょうか。一般的には、ゲームループの移動処理が終わった後に書きます。しかし、今回はサブルーチンの個数を削減するために描画部に移動処理を書いてますね。そこで、ゲームループには書かずに描画部の敵機移動と敵機描画の間に書くことにします。

リスト(5.2.1)

```
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            enemyY(cnt)+=3:if(enemyY(cnt)>400)       :enemyF(cnt)=0:continue
            if FightCrash(enemyX(cnt),enemyY(cnt))   :enemyF(cnt)=0:continue ←追加
            if  TamaCrash(enemyX(cnt),enemyY(cnt))   :enemyF(cnt)=0:continue ←追加
            pos enemyX(cnt),enemyY(cnt)
            color $00,$FF,$FF:mes "X"
        }
    loop
    return
```

これで次の２つは、描画部に組み込んだことになります。

1. 敵機が移動した時に、自機と衝突
2. 敵機が移動した時に、自機弾と衝突

5.3 自機のサブルーチン

続いて自機の処理部ですが、ゲームループに自機の移動＆描画、自機弾の発射を書いてますね。
この部分を分かりやすくサブルーチンにしましょう。

リスト(5.3.1)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%11111
        gosub *EnemyBirth
        gosub *FightDraw
        gosub *EnemyDraw
        gosub *TamaDraw
        redraw 1
        await (1000/60)
    loop
    stop
*FightDraw
    if(key&1):x-=5:if(x<0):x=0
    if(key&2):y-=5:if(y<0):y=0
    if(key&4):x+=5:if(x>550):x=550
    if(key&8):y+=5:if(y>350):y=350
    if(key&16):gosub *TamaBirth
    color $00,$00,$00:boxf
    color $00,$FF,$00:pos x,y:mes "山"
    return
```

追加

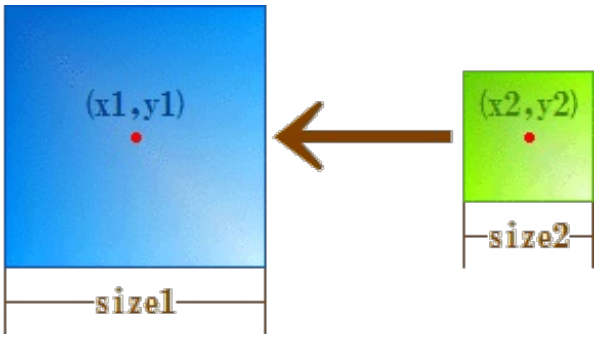
追加

5.4 当たり判定の考え

当たり判定には、色々なタイプがあります。

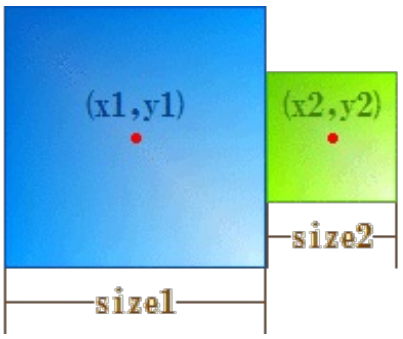
1. 矩形＋矩形
2. 矩形＋点
3. 円＋円
4. 円＋点
5. 点＋点
6. 差分

上記の６つが代表ですが、それ以外も多数ありますね。そして、一般的な当たり判定は、「矩形＋矩形」だと思います。しかし、今回のミニ講座では、理解しやすい「差分」を使います。



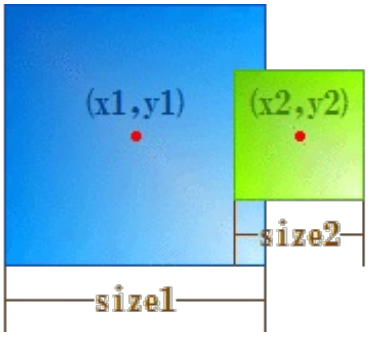
上記の図は、これから衝突しようとしてる２つのキャラクタです。これから緑色が青色に向かって、衝突しようとしてます。このとき、キャラクタの重なり具合を順に追って考えてみましょう。

タイミング(1)



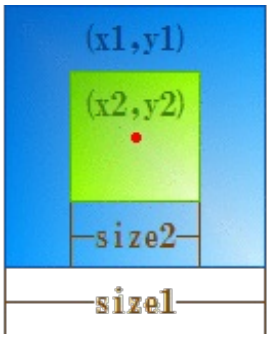
このタイミングでは、２つのキャラクタは衝突してませんね。差分の|x2-x1|の距離が(size1/2)+(size2/2)よりも小さくないため「衝突してない」と判断できます。

タイミング(2)



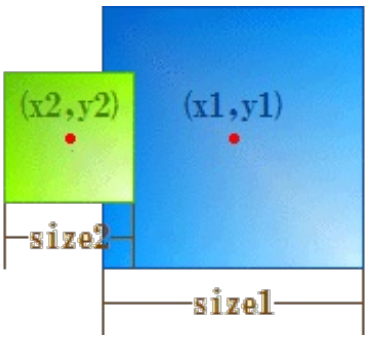
このタイミングでは、２つのキャラクタは衝突してます。差分の|x2-x1|の距離が(size1/2)+(size2/2)よりも小さくなったので「衝突した」と判断できます。

タイミング(3)



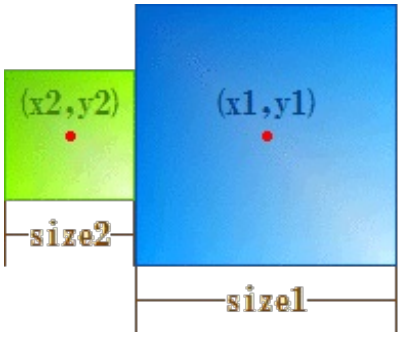
このタイミングでも、２つのキャラクタは衝突してます。差分の|x2-x1|の距離が(size1/2)+(size2/2)よりも小さくなったので「衝突した」と判断できます。

タイミング(4)



このタイミングでも、２つのキャラクタは衝突してます。差分の|x2-x1|の距離が(size1/2)+(size2/2)よりも小さくなったので「衝突した」と判断できます。

タイミング(5)



このタイミングでは、２つのキャラクタは衝突してません。差分の|x2-x1|の距離が(size1/2)+(size2/2)よりも小さくないため「衝突してない」と判断できます。

まとめ

まとめると緑色(x2)から青色(x1)を引いた絶対値を求めます。上記の|x2-x1|と表現されてる部分が、数学上で絶対値を求めることを意味してます。そして青色のサイズ(size1)を２で割った値と緑色のサイズ(size2)を２で割った値を足した距離を求めます。その後に差分の絶対値と距離を比較して、絶対値が距離よりも小さければ「衝突した」と判断できます。また、ミニ講座では size1 も size2 も同じ 50 ドットですから、距離＝(size1/2)+(size2/2)＝(50/2)+(50/2)＝25＋25＝50となります。決して size1 の 50 ドットとか、size2 の 50 ドットが２つのキャラクタの距離になる訳ではありません。ご注意。それでは、当たり判定の条件式を整理します。

- |x2-x1| ＜ 50 … 衝突してる
- |x2-x1| ＝ 50 … 衝突してない
- |x2-x1| ＞ 50 … 衝突してない

差分の絶対値が、距離である 50 ドットよりも小さければ、衝突してると判断すれば良いわけです。それでは、縦軸の当たり判定も載せます。

- |y2-y1| ＜ 50 … 衝突してる
- |y2-y1| ＝ 50 … 衝突してない
- |y2-y1| ＞ 50 … 衝突してない

見れば分かると思いますが、横軸と縦軸は同じ考えで、当たり判定(衝突判定)を行えば良いわけです。それでは、自機の衝突判定(FightCrash)、自機弾の衝突判定(TamaCrash)をそれぞれ作成しましょう。

5.5 自機の衝突判定

リスト(5.5.1)

```
#defcfunc FightCrash int _x_,int _y_
    if(abs(x-_x_)<50)and(abs(y-_y_)<50){
        fight--
        return 1
    }
    return 0
```

上記のユーザ定義関数 FightCrash は、敵機の座標を引数で受け取ります。そして、自機の座標との差分を取り、その絶対値が距離である 50 ドットよりも小さい場合は、衝突したと判定します。その後は、自機の残機数を 1 つ減らして、衝突したことを意味する 1 を返します。衝突してない場合は 0 を返します。

5.6 自機弾の衝突判定

リスト(5.6.1)

```
#defcfunc TamaCrash int _x_,int _y_
    n=0
    foreach tamaF
        if tamaF(cnt){
            if(abs(tamaX(cnt)-_x_)<50)and(abs(tamaY(cnt)-_y_)<50){
                tamaF(cnt)=0
                n=1
                break
            }
        }
    loop
    return n
```

上記のユーザ定義関数 TamaCrash は、敵機の座標を引数で受け取ります。そして、自機弾の座標との差分を取り、その絶対値が距離である 50 ドットよりも小さい場合は、衝突したと判定します。その後は、自機弾の有無フラグを 0 として、衝突したことを意味する 1 を返します。衝突してない場合は 0 を返します。

5.7 当たり判定

それでは第5章の「当たり判定」ソースを紹介します。

リスト(5.7.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第5章「当たり判定」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Init
    ;弾丸の管理
    dim tamaF,10
    dim tamaX,10
    dim tamaY,10
    ;敵機の管理
    dim enemyF,20
    dim enemyX,20
    dim enemyY,20
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%11111
        gosub *EnemyBirth
        gosub *FightDraw
        gosub *EnemyDraw
        gosub *TamaDraw
        redraw 1
        await (1000/60)
    loop
    stop
//--------------------------------------
// 自機の描画
//--------------------------------------
*FightDraw
    if(key&1):x-=5:if(x<0):x=0
    if(key&2):y-=5:if(y<0):y=0
    if(key&4):x+=5:if(x>550):x=550
    if(key&8):y+=5:if(y>350):y=350
    if(key&16):gosub *TamaBirth
    color $00,$00,$00:boxf
    color $00,$FF,$00:pos x,y:mes "山"
    return
//--------------------------------------
// 弾丸の発生
//--------------------------------------
*TamaBirth
    if(tamaTrigg):tamaTrigg--:return
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    tamaTrigg=8
    return
```

```
//-------------------------------------
// 弾丸の描画
//-------------------------------------
*TamaDraw
    foreach tamaF
        if tamaF(cnt){
            tamaY(cnt)-=8:if(tamaY(cnt)<-50):tamaF(cnt)=0:continue
            pos tamaX(cnt),tamaY(cnt)
            color $FF,$FF,$00:mes ":"
        }
    loop
    return
//-------------------------------------
// 敵機の発生
//-------------------------------------
*EnemyBirth
    if(enemyCycle):enemyCycle--:return
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            break
        }
    loop
    enemyCycle=30
    return
//-------------------------------------
// 敵機の描画
//-------------------------------------
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            enemyY(cnt)+=3:if(enemyY(cnt)>400)      :enemyF(cnt)=0:continue
            if FightCrash(enemyX(cnt),enemyY(cnt))  :enemyF(cnt)=0:continue
            if  TamaCrash(enemyX(cnt),enemyY(cnt))  :enemyF(cnt)=0:continue
            pos enemyX(cnt),enemyY(cnt)
            color $00,$FF,$FF:mes "Ж"
        }
    loop
    return
//-------------------------------------
// 自機の衝突判定
//-------------------------------------
#defcfunc FightCrash int _x_,int _y_
    if(abs(x-_x_)<50)and(abs(y-_y_)<50){
        fight--
        return 1
    }
    return 0
//-------------------------------------
// 自機弾の衝突判定
//-------------------------------------
#defcfunc TamaCrash int _x_,int _y_
    n=0
    foreach tamaF
        if tamaF(cnt){
            if(abs(tamaX(cnt)-_x_)<50)and(abs(tamaY(cnt)-_y_)<50){
                tamaF(cnt)=0
                n=1
                break
            }
        }
    loop
    return n
//-------------------------------------------------------------------------
// End of lesson-5.hsp
//-------------------------------------------------------------------------
```

これで「当たり判定」は終わります。次章は敵機を破壊した時の「爆発アニメ」を紹介します。

◆◆◆ 第6章 爆発アニメ ◆◆◆

1. 爆発のアニメ
2. 爆発カウンタ
3. 爆発処理の管理
4. 自機の爆発アニメ
5. 敵機の爆発アニメ
6. 爆発アニメ

6.1 爆発のアニメ

爆発のアニメーション処理は、自機の破壊、敵機の破壊などで使います。まずは次のソースを試してみましょう。

リスト(6.1.1)

```
screen 0,600,400,SCREEN_FIXEDSIZE
font MSGOTHIC,50
repeat
    x=rnd(600/50)*50
    y=rnd(400/50)*50
    repeat 60
        color $00,$00,$00:boxf
        color $FF,$00,$00:pos x,y
        if(cnt\10<5):mes "※":else:mes "＊"
        await (1000/60)
    loop
    color $00,$00,$00:boxf
    await 1000
loop
```

上記のサンプルを実行すると画面の任意の場所で「※」文字と「＊」文字が交互に表示されます。すごく単純な方法ですが、爆発アニメーションに使えませんか？今回は、この文字による爆発アニメーションで、自機の破壊、敵機の破壊を表現しましょう。

6.2 爆発カウンタ

リスト(6.2.1)

```
if(cnt\10<5):mes "※":else:mes "＊"
```

上記のソースで cnt が爆発カウンタとして、爆発アニメーションを演出してます。この爆発カウンタが 60 回ループすることで、１秒間の爆発アニメーションを行わせてます。なお、２秒間ならば 120 回ループするようにすると良いでしょう。この爆発カウンタを 10 で割った余りの 0 ～ 9 が点滅周期となります。そして、この点滅周期の半分で「※」文字と「＊」文字を交互に表示させてます。なお、この点滅周期は 1/6 秒間ですから１秒間に６回の点滅(６Hz)となります。

6.3 爆発処理の管理

自機の破壊、敵機の破壊を爆発アニメーションで演出する場合、爆発カウンタを 1 つ用意すれば処理できそうですね。爆発カウンタに爆発時間をフレーム数でセットして、徐々に減らして 0 になったら自機(敵機)が消滅するようにします。なお、自機の場合は、残機数を減らして再登場させる必要があると思います。それでは、自機と敵機の管理変数を見てみましょう。

リスト(6.3.1)

```
;自機の管理
dim x                   ;自機の横軸
dim y                   ;自機の縦軸
dim fight               ;自機の残機数
dim blast               ;自機の爆発カウンタ ←追加
```

リスト(6.3.2)

```
;敵機の管理
dim enemyF,20           ;有無フラグ
dim enemyX,20           ;敵機の横軸
dim enemyY,20           ;敵機の縦軸
dim enemyZ,20           ;敵機の爆発カウンタ ←追加
```

上記の blast 変数、enemyZ 配列が、爆発アニメを行うための爆発カウンタとなります。この爆発カウンタが 0 のときは、自機も敵機も生きてる状態と考え、0 以外の値がセットされてたら爆発アニメ処理中と考えましょう。そして、この爆発カウンタは、爆発アニメの時間と共に、点滅するためのサイクルにも使います。

リスト(6.3.3)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%11111
        gosub *EnemyBirth
        gosub *FightDraw
        gosub *EnemyDraw
        gosub *TamaDraw
        redraw 1
        await (1000/60)
    loop
    stop
```

上記のゲームループで await 命令で (1000/60)とありますね。これは、 1 秒間(1000ミリ秒間)で 60 回のループを行う事を意味してます。この 60 はゲームを処理するフレーム数のことで、一般的に 60 FPSと呼ばれてる数です。この 60 フレームを元に爆発カウンタにセットする値、点滅周期を考えると次のようになります。

- 爆発カウンタに 60 をセットすると約 1 秒間の爆発アニメになる
- 爆発カウンタの 60 を点滅回数の 6 で割ると点滅周期は 10 フレームとなる
- 点滅周期の 10 フレームの半分の 5 フレーム間隔で点滅処理を行う

上記の考え方で、自機、敵機の爆発アニメ(点滅)を行ってみたいと思います。

6.4 自機の爆発アニメ

自機の爆発アニメで追加変更する部分は、自機の描画部、衝突判定の２カ所です。この２カ所は、次のような追加変更すれば良いと思います。

1. 自機の描画部は blast変数の爆発カウンタの初期化と点滅処理を行う
2. 自機の衝突判定は blast 変数の爆発カウンタで衝突判定しない

それでは、自機の描画部(FightDraw)、自機の衝突判定(FightCrash)を改良しましょう。

自機の描画部

自機の描画部は blast 変数の爆発カウンタの状態で、４つの処理に分岐します。４つの処理とは、自機の移動・発射・描画、爆発アニメ、残機数、ゲームオーバーです。この仕組みを分かりやすく擬似言語で表してみます。

リスト(6.4.1)

```
if(blast==0){
    自機の移動
    自機の弾発射
    自機の描画
}
else:if(blast>1){
    blast--
    爆発アニメ
}
else:if(残機数){
    残機数--
    自機の誕生
}
else{
    ゲームオーバー
}
```

1. 爆発カウンタの blastが 0 のとき、自機は生きてるので移動・弾発射・描画を行います。
2. 爆発カウンタの blastが 1 以上は、自機は破壊されてるので、爆発アニメを処理します。
3. 爆発カウンタの blastが 1 のとき、残機数を減らして、自機を発生させます。
4. 爆発カウンタの blast が 1 のときで、残機数も 0 の場合は、ゲームオーバーとなります。

それでは、この疑似言語を元に自機の描画部(FightDraw)を改良してみましょう。

リスト(6.4.2)

```
*FightDraw
    if(blast==0){
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
    }
    else:if(blast>1){
```

```
        blast--
        color $00,$00,$00:boxf
        color $FF,$00,$00:pos x,y
        if(blast\10<5):mes "※":else:mes "＊"
    }
    else:if(fight){
        fight--
        blast=0
        x=(600-50)/2
        y=(400-50)-16
    }
    else{
        dialog "ゲームオーバーです。",1,"確認"
        end
    }
    return
```

自機の衝突判定

自機の衝突判定で blast の爆発カウンタが 0 のとき、当たり判定を行わないようにします。この処理をなくすと爆発アニメに敵機が衝突しても敵機が破壊されます。この動作が意図的なものであれば良いのですが、一般に爆発アニメに敵機が突っ込んでもダメージは受けないモノとします。その方がゲームらしく見えます。

リスト(6.4.3)

```
#defcfunc FightCrash int _x_,int _y_
    if(blast==0)and(abs(x-_x_)<50)and(abs(y-_y_)<50){ ←修正
        blast=60 ←追加
        return 1
    }
    return 0
```

6.5 敵機の爆発アニメ

敵機の爆発アニメで追加変更する部分は、敵機の発生部、描画部、の２カ所です。この２カ所は、次のような追加変更すれば良いと思います。

1. 敵機の発生部は enemyZ配列の爆発カウンタを初期化する
2. 敵機の描画部は enemyZ 配列の爆発カウンタで点滅処理を行う

それでは、敵機の発生部(EnemyBirth)、敵機の描画部(EnemyDraw)を改良しましょう。

敵機の発生部

敵機の発生部は enemyZ 配列の爆発カウンタを初期化するだけです。この爆発カウンタが 0 のとき、敵機は生きてる事を意味してます。そのため敵機の発生で必ず 0 で初期化しておきましょう。

リスト(6.5.1)

```
*EnemyBirth
    if(enemyCycle):enemyCycle--:return
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            enemyZ(cnt)=0 ←追加
            break
        }
    loop
    enemyCycle=30
    return
```

敵機の描画部

敵機の描画部は enemyZ 配列の爆発カウンタの状態で、３つの処理に分岐します。３つの処理とは、敵機の移動・衝突判定・描画、爆発アニメ、敵機の消滅です。この仕組みを分かりやすく擬似言語で表してみます。

リスト(6.5.2)

```
if(enemyZ(cnt)==0){
    敵機の移動
    自機との衝突判定
    自機弾との衝突判定
    敵機の描画
}
else:if(enemyZ(cnt)>1){
    enemyZ(cnt)--
    爆発アニメ
}
else{
    敵機の消滅
}
```

1. 爆発カウンタの enemyZ(cnt)が 0 のとき、敵機は生きてるので移動・衝突判定・描画を行います。
2. 爆発カウンタの enemyZ(cnt)が 1 以上は、敵機は破壊されてるので、爆発アニメを処理します。
3. 爆発カウンタの enemyZ(cnt) が 1 のとき、爆発アニメを終了して、敵機の消滅を行います。

それでは、この疑似言語を元に敵機の描画部(EnemyDraw)を改良してみましょう。

リスト(6.5.3)

```
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            if(enemyZ(cnt)==0){
                enemyY(cnt)+=3:if(enemyY(cnt)>400)       :enemyF(cnt)=0:continue
                if FightCrash(enemyX(cnt),enemyY(cnt))   :enemyZ(cnt)=60:continue
                if  TamaCrash(enemyX(cnt),enemyY(cnt))   :enemyZ(cnt)=60:continue
                pos enemyX(cnt),enemyY(cnt)
                color $00,$FF,$FF:mes "Ж"
            }
            else:if(enemyZ(cnt)>1){
                enemyZ(cnt)--
                color $FF,$00,$00:pos enemyX(cnt),enemyY(cnt)
                if(enemyZ(cnt)\10<5):mes "※":else:mes "＊"
            }
            else{
                enemyF(cnt)=0
            }
        }
    loop
    return
```

6.6 爆発アニメ

それでは第6章の「爆発アニメ」ソースを紹介します。

リスト(6.6.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第6章「爆発アニメ」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Init
    ;自機の管理
    dim x               ;自機の横軸
    dim y               ;自機の縦軸
    dim fight           ;自機の残機数
    dim blast           ;自機の爆発カウンタ
    ;弾丸の管理
    dim tamaF,10        ;有無フラグ
    dim tamaX,10        ;弾丸の横軸
    dim tamaY,10        ;弾丸の縦軸
    ;敵機の管理
    dim enemyF,20       ;有無フラグ
    dim enemyX,20       ;敵機の横軸
    dim enemyY,20       ;敵機の縦軸
    dim enemyZ,20       ;敵機の爆発カウンタ
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%11111
        gosub *EnemyBirth
        gosub *FightDraw
        gosub *EnemyDraw
        gosub *TamaDraw
        redraw 1
        await (1000/60)
    loop
    stop
//--------------------------------------
// 自機の描画
//--------------------------------------
*FightDraw
    if(blast==0){
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth
        color $00,$00,$00:boxf
        color $00,$FF,$00:pos x,y:mes "山"
    }
    else:if(blast>1){
        blast--
        color $00,$00,$00:boxf
        color $FF,$00,$00:pos x,y
        if(blast\10<5):mes "※":else:mes "＊"
    }
    else:if(fight){
        fight--
```

```
        blast=0
        x=(600-50)/2
        y=(400-50)-16
    }
    else{
        dialog "ゲームオーバーです。",1,"確認"
        end
    }
    return
//-------------------------------------
// 弾丸の発生
//-------------------------------------
*TamaBirth
    if(tamaTrigg):tamaTrigg--:return
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    tamaTrigg=8
    return
//-------------------------------------
// 弾丸の描画
//-------------------------------------
*TamaDraw
    foreach tamaF
        if tamaF(cnt){
            tamaY(cnt)-=8:if(tamaY(cnt)<-50):tamaF(cnt)=0:continue
            pos tamaX(cnt),tamaY(cnt)
            color $FF,$FF,$00:mes ":"
        }
    loop
    return
//-------------------------------------
// 敵機の発生
//-------------------------------------
*EnemyBirth
    if(enemyCycle):enemyCycle--:return
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            enemyZ(cnt)=0
            break
        }
    loop
    enemyCycle=30
    return
//-------------------------------------
// 敵機の描画
//-------------------------------------
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            if(enemyZ(cnt)==0){
                enemyY(cnt)+=3:if(enemyY(cnt)>400)      :enemyF(cnt)=0:continue
                if FightCrash(enemyX(cnt),enemyY(cnt))  :enemyZ(cnt)=60:continue
                if  TamaCrash(enemyX(cnt),enemyY(cnt))  :enemyZ(cnt)=60:continue
                pos enemyX(cnt),enemyY(cnt)
                color $00,$FF,$FF:mes "Ж"
            }
            else:if(enemyZ(cnt)>1){
                enemyZ(cnt)--
                color $FF,$00,$00:pos enemyX(cnt),enemyY(cnt)
```

```
                if(enemyZ(cnt)\10<5):mes "※":else:mes "＊"
            }
            else{
                enemyF(cnt)=0
            }
        }
    loop
    return
//-------------------------------------
// 自機の衝突判定
//-------------------------------------
#defcfunc FightCrash int _x_,int _y_
    if(blast==0)and(abs(x-_x_)<50)and(abs(y-_y_)<50){
        blast=60
        return 1
    }
    return 0
//-------------------------------------
// 自機弾の衝突判定
//-------------------------------------
#defcfunc TamaCrash int _x_,int _y_
    n=0
    foreach tamaF
        if tamaF(cnt){
            if(abs(tamaX(cnt)-_x_)<50)and(abs(tamaY(cnt)-_y_)<50){
                tamaF(cnt)=0
                n=1
                break
            }
        }
    loop
    return n
//-------------------------------------------------------------------------------
// End of lesson-6.hsp
//-------------------------------------------------------------------------------
```

これで「爆発アニメ」は終わります。次章は背景を演出するための「流星の表示」を紹介します

。

◆◆◆ 第7章 流星の表示 ◆◆◆

1. 流星のアニメ
2. 流星の管理
3. 流星の処理
4. 流星の発生
5. 流星の描画
6. 発生の間隔
7. 流星の表示

7.1 流星のアニメ

流星のアニメーション処理は、簡単な方法で背景がスクロールしてるように見えます。そのため簡単なミニ・ゲームや、サンプル・ゲームには良く登場します。また、実行ファイルをコンパクトにするためにも、背景画像を用意しないでスクロールを演出できるのは便利でしょう。今回は、この流星による背景スクロールを考えてみます。

7.2 流星の管理

流星は、最大50個発生できることを考えてみます。まずは流星の横軸(x)、縦軸(y)は絶対に必要なので、この２つを配列で管理します。さらに流星の配列が、画面内を移動中(有効)か、画面外に外れた(無効)かを判定するフラグ(flag)も必要になりますね。それでは３つを配列で管理すると次のようになります。

リスト(7.2.1)

```
dim starF,50
dim starX,50
dim starY,50
```

上記の３つの配列は次の意味があります。

- starF は、流星の有無フラグで 0 なら無効、1 なら有効を意味
- starX は、流星の横軸で 0 ～ 599 の値を管理
- starY は、流星の縦軸で 0 ～ 399 の値を管理

なお、流星は 1 ドットの点として描画します。

7.3 流星の処理

流星を処理するには、流星の発生、移動、描画の最低でも３つの処理が必要です。今回は、流星の発生部(StarBirth)、流星の描画部(StarDraw)の２つをサブルーチンにします。なお、流星の移動は描画部(StarDraw)処理と一緒に記述して、サブルーチンの数を減らしてみます。それでは、次のサンプルを見てみましょう。

リスト(7.3.1)

```
*Init            ┐
    dim starF,50  ├追加
    dim starX,50  │
    dim starY,50 ┘
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        color $00,$00,$00:boxf
        gosub *StarBirth ←追加
        gosub *StarDraw  ←追加
        gosub *FightDraw
        redraw 1
        await (1000/60)
    loop
    stop
*FightDraw
    x=(600-50)/2
    y=(400-50)-16
    color $00,$FF,$00:pos x,y:mes "山"
    return
*StarBirth ┐
    :       │
    return  ├追加
*StarDraw   │
    :       │
    return ┘
```

とりあえず、上記の４カ所が流星スクロールに関係する部分です。

7.4 流星の発生

それでは、流星の発生ルーチン(StarBirth)を考えてみましょう。流星の発生手順は、次のようになると思います。

1. StarF 配列から流星情報が無効である場所を探す
2. 無効領域が見つかったら流星を発生(a)～(c)を行う
    a. 流星の有無フラグに 1 をセット
    b. 流星の横軸(x)に 0 ～ 599 をセット(乱数で決定)
    c. 流星の縦軸(y)に 0 をセット
3. 繰り返しを抜ける

リスト(7.4.1)

```
*StarBirth
    foreach starF
        if(starF(cnt)==0){
            starF(cnt)=1
            starX(cnt)=rnd(600)
            starY(cnt)=0
            break
        }
    loop
    return
```

上記のように starF 配列の分だけ繰り返す場合は repeat 命令よりも foreach 命令が便利です。たったのこれだけで、無効領域に有効な流星情報をセットして、新しい流星を発生できます。どうですか？以前に紹介した弾丸の発生、敵機の発生とほぼ同じですね。つまり、弾丸の発生、敵機の発生、流星の発生などは、同じ要領で管理できるのです。これで流星の発生は一通り終わりです。

7.5 流星の描画

続いて、流星の描画ルーチン(StarDraw)を考えてみましょう。こちらは、流星の移動も含まれますので、先に移動処理、続いて描画処理を行います。

1. starF 配列から流星情報が有効である場所を探す
2. 有効領域が見つかったら流星の移動(a)〜(d)を行う
    a. 流星の縦軸(y)に 2 ドット足す
    b. 流星の縦軸(y)が画面外に外れたかチェック
    c. 画面外に外れた場合は、流星の有無フラグに 0 をセット
    d. 次の繰り返しへジャンプ(continue)
3. 流星の描画色を設定(color)
4. 流星の点を描画(pset)

リスト(7.5.1)

```
*StarDraw
    foreach starF
        if starF(cnt){
            starY(cnt)+=2:if(starY(cnt)>=400):starF(cnt)=0:continue
            color $FF,$FF,$00
            pset starX(cnt),starY(cnt)
        }
    loop
    return
```

上記のソースで continue 命令がありますが、これを実行すると次の繰り返し処理にジャンプします。そのため次の行の color 命令、pset 命令は実行されません。これで流星の描画は一通り終わりです。

7.6 発生の間隔

上記のままでは、一度に50個の流星が発生してしまいます。これでは、画面にまばらに流星が発生しないので不自然ですね。この不自然さを解決する方法は２つあります。

1. 流星の発生個数を 50個 から 200 個などに増やす
2. 敵機の発生部のように starCycle カウンタを用意する

一番簡単な解決策は、発生個数を増やすことでしょう。これで単純ではありますが、画面一杯に流星が「まばら」に発生します。今回は starCycle カウンタを用意して、画面一杯に流星が「まばら」に発生するようにします。改良する部分は StarBirth のサブルーチンです。

リスト(7.6.1)

```
*StarBirth
    if(starCycle):starCycle--:return ←追加
    foreach starF
        if(starF(cnt)==0){
            starF(cnt)=1
            starX(cnt)=rnd(600)
            starY(cnt)=0
            break
        }
    loop
    starCycle=3 ←追加
    return
```

上記の２カ所だけ追加します。ここでは３サイクルで１個の流星が発生するようにしてます。このサイクル数を変更すると流星の発生間隔を制御できます。

7.7 流星の表示

画面の消去処理を、今までは FightDraw 内に記述してましたが、今回から StarDraw に記述します。記述を移動した理由は、画面消去→流星→自機→敵機→弾丸の順に描画した方が良いからです。移動しないと画面消去→自機→流星→敵機→弾丸の順に描画されて、自機の上に流星が移動して不自然になります。これが意図的なものなら良いのですが、一般に流星は自機の背景をスクロールしますよね。だから画面消去の部分を FightDraw から StarDraw に記述を移したのです。それでは第7章の「流星の表示」ソースを紹介します。

リスト(7.7.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第7章「流星の表示」by 科学太郎
//-------------------------------------------------------------------------------

//---------------------------------------
// メイン部
//---------------------------------------
*Init
    ;自機の管理
    dim x               ;自機の横軸
    dim y               ;自機の縦軸
    dim fight           ;自機の残機数
    dim blast           ;自機の爆発カウンタ
    ;弾丸の管理
    dim tamaF,10        ;有無フラグ
    dim tamaX,10        ;弾丸の横軸
    dim tamaY,10        ;弾丸の縦軸
    ;敵機の管理
    dim enemyF,20       ;有無フラグ
    dim enemyX,20       ;敵機の横軸
    dim enemyY,20       ;敵機の縦軸
    dim enemyZ,20       ;敵機の爆発カウンタ
    ;流星の管理
    dim starF,50        ;有無フラグ
    dim starX,50        ;流星の横軸
    dim starY,50        ;流星の縦軸
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%11111
        gosub *EnemyBirth
        gosub *StarBirth
        gosub *StarDraw
        gosub *FightDraw
        gosub *EnemyDraw
        gosub *TamaDraw
        redraw 1
        await (1000/60)
    loop
    stop
//---------------------------------------
// 自機の描画
//---------------------------------------
*FightDraw
    if(blast==0){
        if(key&1):x-=5:if(x<0):x=0
```

```
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth
        color $00,$FF,$00:pos x,y:mes "山"
    }
    else:if(blast>1){
        blast--
        color $FF,$00,$00:pos x,y
        if(blast\10<5):mes "※":else:mes "＊"
    }
    else:if(fight){
        fight--
        blast=0
        x=(600-50)/2
        y=(400-50)-16
    }
    else{
        dialog "ゲームオーバーです。",1,"確認"
        end
    }
    return
//-------------------------------------
// 流星の発生
//-------------------------------------
*StarBirth
    if(starCycle):starCycle--:return
    foreach starF
        if(starF(cnt)==0){
            starF(cnt)=1
            starX(cnt)=rnd(600)
            starY(cnt)=0
            break
        }
    loop
    starCycle=3
    return
//-------------------------------------
// 流星の描画
//-------------------------------------
*StarDraw
    color $00,$00,$00:boxf
    foreach starF
        if starF(cnt){
            starY(cnt)+=2:if(starY(cnt)>=400):starF(cnt)=0:continue
            color $FF,$FF,$00
            pset starX(cnt),starY(cnt)
        }
    loop
    return
//-------------------------------------
// 弾丸の発生
//-------------------------------------
*TamaBirth
    if(tamaTrigg):tamaTrigg--:return
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    tamaTrigg=8
    return
//-------------------------------------
// 弾丸の描画
//-------------------------------------
```

```
*TamaDraw
    foreach tamaF
        if tamaF(cnt){
            tamaY(cnt)-=8:if(tamaY(cnt)<-50):tamaF(cnt)=0:continue
            pos tamaX(cnt),tamaY(cnt)
            color $FF,$FF,$00:mes ":"
        }
    loop
    return
//-------------------------------------
// 敵機の発生
//-------------------------------------
*EnemyBirth
    if(enemyCycle):enemyCycle--:return
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            enemyZ(cnt)=0
            break
        }
    loop
    enemyCycle=30
    return
//-------------------------------------
// 敵機の描画
//-------------------------------------
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            if(enemyZ(cnt)==0){
                enemyY(cnt)+=3:if(enemyY(cnt)>400)      :enemyF(cnt)=0:continue
                if FightCrash(enemyX(cnt),enemyY(cnt))  :enemyZ(cnt)=60:continue
                if  TamaCrash(enemyX(cnt),enemyY(cnt))  :enemyZ(cnt)=60:continue
                pos enemyX(cnt),enemyY(cnt)
                color $00,$FF,$FF:mes "Ж"
            }
            else:if(enemyZ(cnt)>1){
                enemyZ(cnt)--
                color $FF,$00,$00:pos enemyX(cnt),enemyY(cnt)
                if(enemyZ(cnt)\10<5):mes "※":else:mes "＊"
            }
            else{
                enemyF(cnt)=0
            }
        }
    loop
    return
//-------------------------------------
// 自機の衝突判定
//-------------------------------------
#defcfunc FightCrash int _x_,int _y_
    if(blast==0)and(abs(x-_x_)<50)and(abs(y-_y_)<50){
        blast=60
        return 1
    }
    return 0
//-------------------------------------
// 自機弾の衝突判定
//-------------------------------------
#defcfunc TamaCrash int _x_,int _y_
    n=0
    foreach tamaF
        if tamaF(cnt){
            if(abs(tamaX(cnt)-_x_)<50)and(abs(tamaY(cnt)-_y_)<50){
                tamaF(cnt)=0
                n=1
```

```
                break
            }
        }
    loop
    return n
//-----------------------------------------------------------------------------
// End of lesson-7.hsp
//-----------------------------------------------------------------------------
```

これで「流星の表示」は終わります。次章は自機のゲームオーバーとコンティニューを含む「残機の管理」を紹介します。

◆◆◆ 第8章 残機の管理 ◆◆◆

1. 残機の管理
2. 自機の初期化
3. 残機の表示
4. スコアの管理
5. スコアの加算
6. スコアの表示
7. コンティニューの処理
8. 残機の管理

8.1 残機の管理

残機の管理とは、自機が敵機と衝突して、破壊されたときに 1 つ残機を失います。そして、残機数が 0 のときに敵機と衝突して破壊されるとゲームオーバーになるのが一般的でしょう。まずは、自機の管理変数を見てみましょう。

リスト(8.1.1)

```
dim x
dim y
dim fight
dim blast
```

上記の 4 つの変数は次の意味があります。

- x は、自機の横軸
- y は、自機の縦軸
- fight は、自機の残機数
- blast は、自機の爆発カウンタ

残機数の fight は、既に FightDraw 内で記述してます。しかし、初期化してないためゲーム開始時は、残機数の fight は 0 ですね。このままだと自機が敵機と衝突して、破壊されると直ぐにゲームオーバーになってました。そこで、まず最初にゲーム開始時の自機の初期化を追加しましょう。

8.2 自機の初期化

自機の初期化は、次のようにすれば良いでしょう。

リスト(8.2.1)

```
*Init
    ;自機の管理
    ;
    ;省略
    ;
    ;自機の初期化 ┐
    x=(600-50)/2  │
    y=(400-50)-16 ├追加
    fight=3       │
    blast=0       ┘
*Main
```

上記の x、y にゲーム開始時の自機の座標をセットしてます。今までは、両方とも 0 だったので左上隅に自機が表示されました。そのため場合によっては、敵機と衝突して直ぐにゲームオーバーになってましたね。これを防ぐには、ゲーム開始時に画面下部の中央に表示されるようにセットします。画面の横幅である 600 ドットから自機の横幅の 50 ドットを引いて、２で割れば画面中央の座標が求まります。画面の縦幅である 400 ドットから自機の縦幅の 50 ドットを引いて、さらに下部の隙間である 16 ドットを引いてます。これにより、画面下部から 16 ドット上に自機が表示されます。あとは残機数の fight に 3 をセットして、爆発カウンタの blast は 0 をセットします。これで「自機の初期化」は終わりですね。

8.3 残機の表示

続いて、残機の表示を行いましょう。この「残機の表示」は数字で表すか、マークで表すかの２通りありますね。今回は、マークで残機を表すことにしましょう。また、マークといっても色々あります。自機マークとか、♥マークとかがあります。今回は、自機マークの「山」文字で表すことにしました。それでは、残機数を描画する TelopDraw を作成します。

リスト(8.3.1)

```
*TelopDraw
    msg=""
    repeat fight
        msg+="山"
    loop
    font MSGOTHIC,20,1
    color $00,$FF,$00:pos 0,(400-20):mes msg
    font MSGOTHIC,50
    return
```

上記のように msg 変数に、自機マークである「山」文字を fight 個数分だけ連結します。この自機マーク文字列を画面左下隅の描画します。このとき、文字のフォント・サイズを変更してます。さらにゲームループでも TelopDraw を呼び出す必要があります。

リスト(8.3.2)

```
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%11111
        gosub *EnemyBirth
        gosub *StarBirth
        gosub *StarDraw
        gosub *FightDraw
        gosub *EnemyDraw
        gosub *TamaDraw
        gosub *TelopDraw ←追加
        redraw 1
        await (1000/60)
    loop
    stop
*TelopDraw ┐
    :      ├追加
    return ┘
```

上記のように TelopDraw を呼び出す場所は、描画ルーチンの最後です。このようにすると、画面消去→流星→自機→敵機→弾丸→テロップの順に描画されて、テロップが一番上に表示されます。この順番を間違うとテロップ文字の上に自機や敵機が移動してしまい不自然になると思います。これで「残機の描画」は終わりです。

8.4 スコアの管理

スコアの管理は、プレイヤーの現在スコアと最高スコアの２つを用意すれば良いでしょう。まずはスコアの管理変数を見てみましょう。

リスト(8.4.1)

```
dim Score
dim HiScore
```

何とも分かりやすい変数名です。ちょっと当たり前すぎましたね。(笑)これを初期化部に追加しますよ。

リスト(8.4.2)

```
*Init
    ;スコアの管理 ┐
    dim Score      ├追加
    dim HiScore    ┘
    ;自機の管理
    ;
    ;省略
    ;
    ;スコアの初期化 ┐
    Score=0          ├追加
    HiScore=9900     ┘
    ;自機の初期化
    x=(600-50)/2
    y=(400-50)-16
    fight=3
    blast=0
*Main
```

8.5 スコアの加算

続いて、スコアを加算する場所を考えます。今回は、敵機が 1 種類しか登場しませんので、敵機が破壊された場所に直接スコア加算しましょう。改良する部分は EnemyDraw サブルーチンです。

リスト(8.5.1)

```
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            if(enemyZ(cnt)==0){
                enemyY(cnt)+=3:if(enemyY(cnt)>400)       :enemyF(cnt)=0:continue
                if FightCrash(enemyX(cnt),enemyY(cnt))   :enemyZ(cnt)=60:ScoreCalc 100:continue ←修正
                if  TamaCrash(enemyX(cnt),enemyY(cnt))   :enemyZ(cnt)=60:ScoreCalc 100:continue ←修正
                pos enemyX(cnt),enemyY(cnt)
                color $00,$FF,$FF:mes "Ж"
            }
            else:if(enemyZ(cnt)>1){
                enemyZ(cnt)--
                color $FF,$00,$00:pos enemyX(cnt),enemyY(cnt)
                if(enemyZ(cnt)\10<5):mes "※":else:mes "＊"
            }
            else{
                enemyF(cnt)=0
            }
        }
    loop
    return
```

変更箇所は敵機が破壊された部分で、ユーザ定義命令の ScoreCalc 命令を引数付きで呼び出してます。また、今回は敵機 1 匹当たり 100 ポイントのスコアを加算します。それからスコア加算のユーザ定義命令も作成しましょう。

リスト(8.5.2)

```
#deffunc ScoreCalc int _score_
    Score+=_score_
    if(Score>HiScore):HiScore=Score
    return
```

上記のように現在スコアが最高スコアを上回った場合は、最高スコアに現在スコアを代入してます。これで「スコアの加算」は終わりです。

8.6 スコアの表示

続いて、スコアの表示を追加しましょう。この「スコアの表示」も数字で表すか、グラフィックで表すかの２通りありますね。今回は、数字でスコアを表すことにしましょう。それでは、残機数を描画する TelopDraw に、現在スコア、最高スコアも一緒に描画します。

リスト(8.6.1)

```
*TelopDraw
    msg=""
    repeat fight
        msg+="山"
    loop
    y(1)=0:x(1)=(20)                                    ┐
    y(2)=0:x(2)=(600-20)-(16*11)                        ├追加
    x(3)=0:y(3)=(400-20)                                ┘
    font MSGOTHIC,20,1
    color $FF,$FF,$00:pos x(1),y(1):mes strf("Score:%08d",Score)      ┐
    color $FF,$FF,$00:pos x(2),y(2):mes strf("HiScore:%08d",HiScore)  ├追加
    color $00,$FF,$00:pos x(3),y(3):mes msg                           ┘
    font MSGOTHIC,50
    return
```

どうですか？このテロップ機能(残機数・スコア・最高スコア)を追加しただけで、以外にもゲームらしくなりましたね。これで「スコアの表示」は終わりです。

8.7 コンティニューの処理

自機の残機が 0 のときに、自機が破壊されるとゲームオーバーとなり、今までは強制的に終了してました。そこで、ゲームオーバーになった後、ゲームを終了するか、もう一度ゲームをするかのコンティニュー機能を付けてみましょう。考え方は、次のようにすれば良いでしょう。

1. 「はい」「いいえ」のダイアログを表示する
2. 「はい」を選択した場合は、コンティニュー処理(a)～(c)を行う
    a. スコアの初期化(現在スコア)
    b. 自機の初期化(横軸、縦軸、残機数、爆発カウンタ)
    c. 配列の初期化(弾丸、敵機、流星)
3. 「いいえ」を選択した場合は、プログラムを終了する

それでは、上記のコンティニュー処理をもとに、自機の描画部(FightDraw)を改良してみましょう。

リスト(8.7.1)

```
*FightDraw
    if(blast==0){
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth
        color $00,$FF,$00:pos x,y:mes "山"
    }
    else:if(blast>1){
        blast--
        color $FF,$00,$00:pos x,y
        if(blast\10<5):mes "※":else:mes "＊"
    }
    else:if(fight){
        fight--
        blast=0
        x=(600-50)/2
        y=(400-50)-16
    }
    else{
        dialog "もう一度、ゲームを行いますか？",3,"ゲームオーバー"
        if(stat==7):end
        ;スコアの初期化
        Score=0
        ;自機の初期化
        x=(600-50)/2
        y=(400-50)-16
        fight=3
        blast=0
        ;配列の初期化
        dim starF,50
        dim tamaF,10
        dim enemyF,20
    }
    return
```

追加

これで「コンティニュー機能」は一通り終わりです。

8.8 残機の管理

それでは第8回目の「残機の管理」ソースを紹介します。

リスト(8.8.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第8回「残機の管理」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Init
    ;スコアの管理
    dim Score           ;現在スコア
    dim HiScore         ;最高スコア
    ;自機の管理
    dim x               ;自機の横軸
    dim y               ;自機の縦軸
    dim fight           ;自機の残機数
    dim blast           ;自機の爆発カウンタ
    ;弾丸の管理
    dim tamaF,10        ;有無フラグ
    dim tamaX,10        ;弾丸の横軸
    dim tamaY,10        ;弾丸の縦軸
    ;敵機の管理
    dim enemyF,20       ;有無フラグ
    dim enemyX,20       ;敵機の横軸
    dim enemyY,20       ;敵機の縦軸
    dim enemyZ,20       ;敵機の爆発カウンタ
    ;流星の管理
    dim starF,50        ;有無フラグ
    dim starX,50        ;流星の横軸
    dim starY,50        ;流星の縦軸
    ;スコアの初期化
    Score=0
    HiScore=9900
    ;自機の初期化
    x=(600-50)/2
    y=(400-50)-16
    fight=3
    blast=0
*Main
    screen 0,600,400,SCREEN_FIXEDSIZE
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
        stick key,%11111
        gosub *EnemyBirth
        gosub *StarBirth
        gosub *StarDraw
        gosub *FightDraw
        gosub *EnemyDraw
        gosub *TamaDraw
        gosub *TelopDraw
        redraw 1
        await (1000/60)
    loop
    stop
//--------------------------------------
// 自機の描画
//--------------------------------------
```

```
*FightDraw
    if(blast==0){
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth
        color $00,$FF,$00:pos x,y:mes "山"
    }
    else:if(blast>1){
        blast--
        color $FF,$00,$00:pos x,y
        if(blast\10<5):mes "※":else:mes "＊"
    }
    else:if(fight){
        fight--
        blast=0
        x=(600-50)/2
        y=(400-50)-16
    }
    else{
        dialog "もう一度、ゲームを行いますか？",3,"ゲームオーバー"
        if(stat==7):end
        ;スコアの初期化
        Score=0
        ;自機の初期化
        x=(600-50)/2
        y=(400-50)-16
        fight=3
        blast=0
        ;配列の初期化
        dim starF,50
        dim tamaF,10
        dim enemyF,20
    }
    return
//-------------------------------------
// 流星の発生
//-------------------------------------
*StarBirth
    if(starCycle):starCycle--:return
    foreach starF
        if(starF(cnt)==0){
            starF(cnt)=1
            starX(cnt)=rnd(600)
            starY(cnt)=0
            break
        }
    loop
    starCycle=3
    return
//-------------------------------------
// 流星の描画
//-------------------------------------
*StarDraw
    color $00,$00,$00:boxf
    foreach starF
        if starF(cnt){
            starY(cnt)+=2:if(starY(cnt)>=400):starF(cnt)=0:continue
            color $FF,$FF,$00
            pset starX(cnt),starY(cnt)
        }
    loop
    return
//-------------------------------------
// 弾丸の発生
//-------------------------------------
*TamaBirth
```

```
    if(tamaTrigg):tamaTrigg--:return
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    tamaTrigg=8
    return
//-------------------------------------
// 弾丸の描画
//-------------------------------------
*TamaDraw
    foreach tamaF
        if tamaF(cnt){
            tamaY(cnt)-=8:if(tamaY(cnt)<-50):tamaF(cnt)=0:continue
            pos tamaX(cnt),tamaY(cnt)
            color $FF,$FF,$00:mes ":"
        }
    loop
    return
//-------------------------------------
// 敵機の発生
//-------------------------------------
*EnemyBirth
    if(enemyCycle):enemyCycle--:return
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            enemyZ(cnt)=0
            break
        }
    loop
    enemyCycle=30
    return
//-------------------------------------
// 敵機の描画
//-------------------------------------
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            if(enemyZ(cnt)==0){
                enemyY(cnt)+=3:if(enemyY(cnt)>400)      :enemyF(cnt)=0:continue
                if FightCrash(enemyX(cnt),enemyY(cnt))  :enemyZ(cnt)=60:ScoreCalc 100:continue
                if  TamaCrash(enemyX(cnt),enemyY(cnt))  :enemyZ(cnt)=60:ScoreCalc 100:continue
                pos enemyX(cnt),enemyY(cnt)
                color $00,$FF,$FF:mes "Ж"
            }
            else:if(enemyZ(cnt)>1){
                enemyZ(cnt)--
                color $FF,$00,$00:pos enemyX(cnt),enemyY(cnt)
                if(enemyZ(cnt)\10<5):mes "※":else:mes "＊"
            }
            else{
                enemyF(cnt)=0
            }
        }
    loop
    return
//-------------------------------------
// テロップの描画
//-------------------------------------
*TelopDraw
    msg=""
```

```
    repeat fight
        msg+="山"
    loop
    y(1)=0:x(1)=(20)
    y(2)=0:x(2)=(600-20)-(16*11)
    x(3)=0:y(3)=(400-20)
    font MSGOTHIC,20,1
    color $FF,$FF,$00:pos x(1),y(1):mes strf("Score:%08d",Score)
    color $FF,$FF,$00:pos x(2),y(2):mes strf("HiScore:%08d",HiScore)
    color $00,$FF,$00:pos x(3),y(3):mes msg
    font MSGOTHIC,50
    return
//-------------------------------------
// スコアの加算
//-------------------------------------
#deffunc ScoreCalc int _score_
    Score+=_score_
    if(Score>HiScore):HiScore=Score
    return
//-------------------------------------
// 自機の衝突判定
//-------------------------------------
#defcfunc FightCrash int _x_,int _y_
    if(blast==0)and(abs(x-_x_)<50)and(abs(y-_y_)<50){
        blast=60
        return 1
    }
    return 0
//-------------------------------------
// 自機弾の衝突判定
//-------------------------------------
#defcfunc TamaCrash int _x_,int _y_
    n=0
    foreach tamaF
        if tamaF(cnt){
            if(abs(tamaX(cnt)-_x_)<50)and(abs(tamaY(cnt)-_y_)<50){
                tamaF(cnt)=0
                n=1
                break
            }
        }
    loop
    return n
//-------------------------------------------------------------------------------
// End of lesson-8.hsp
//-------------------------------------------------------------------------------
```

これで「残機の管理」は終わります。次はミニ講座の最終回なのでスタート画面を付けた「完成ソース」を紹介します。

◆◆◆ 第9章 完成ソース ◆◆◆

1. スタート画面
2. 画面デザイン
3. 完成ソース
4. 変数・配列の一覧
5. サブルーチン(命令,関数)の一覧
6. おわりに

9.1 スタート画面

スタート画面の役割は、次の項目を選択するのが一般的でしょう。

- ゲームのタイトル表示
- ゲームの開始
- ゲームの設定
- ゲームの説明
- ゲームの終了

今までゲームを起動すると直ぐにゲームが開始されました。心の準備もなしにゲームがスタートするのは、ちょっとビックリしますね。そこで、簡単ながらゲームのスタート画面を付けて完成させましょう。なお、今回がシューティング・ゲームのミニ講座シリーズの最終回になります。

9.2 画面デザイン

今回は簡単なスタート画面を目指してます。そこで、ゲーム・タイトル、ゲーム開始、ゲーム終了の３要素を付けたいと思います。さらに[SPC]キーが押されると「ゲームの開始」、[ESC]キーが押されると「ゲームの終了」の処理も行います。このスタート画面は、次のようになると思います。

1. スタート画面を表示する(ゲーム・タイトル、ゲーム開始、ゲーム終了)
2. キー入力のループ待ちで(a)～(c)を行う
    a. stick 命令を実行
    b. [SPC]キーが押されてたら break 命令を実行して、ループを抜ける
    c. [ESC]キーが押されてたら end 命令を実行して、ゲーム終了
3. 繰り返しを抜ける
4. ゲームのメインに進む

上記のフローチャートをもとにスタート画面を作成すると次のようになると思います。

リスト(9.2.1)

```
*Start
    screen 0,600,400,SCREEN_FIXEDSIZE
    title "シューティング・ゲームのミニ講座"
    redraw 0
    color:boxf
    color $00,$FF,$FF:font "HG明朝E",50:pos (600-50*11)/2,100:mes "シューティング・ゲーム"
    color $FF,$FF,$00:font "HG明朝E",30:pos (600-30* 6)/2,180:mes "～ミニ講座～"
    color $FF,$FF,$FF:font "HG明朝E",16:pos (600- 8*15)/2,250:mes "[SPC]キーで開始"
    color $FF,$FF,$FF:font "HG明朝E",16:pos (600- 8*15)/2,282:mes "[ESC]キーで終了"
    redraw 1
    repeat
        stick key
        if(key & $10):break
        if(key & $80):end
        await 20
    loop
*Main
```

どうですか？すごくシンプルなスタート画面ですが、よりゲームらしくなりましたね。上記の (key & $10) が[SPC]キーが押されてるかのチェックです。押されていたら break 命令を実行して繰り返しを抜けます。その後は *Main ラベルに進んでゲームが開始されます。上記の (key & $20) は[ESC]キーが押されてるかのチェックです。押されていたら end 命令を実行してウインドウが閉じます。つまり、ゲームの終了を選択したことになります。

9.3 完成ソース

それでは第9回目の「完成ソース」を紹介します。

リスト(9.3.1)

```
//-------------------------------------------------------------------------------
// シューティング・ゲームのミニ講座 for HSP(Ver.3.3.2)
//===============================================================================
// 第9回「完成ソース」by 科学太郎
//-------------------------------------------------------------------------------

//--------------------------------------
// メイン部
//--------------------------------------
*Init
    ;スコアの管理
    dim Score           ;現在スコア
    dim HiScore         ;最高スコア
    ;自機の管理
    dim x               ;自機の横軸
    dim y               ;自機の縦軸
    dim fight           ;自機の残機数
    dim blast           ;自機の爆発カウンタ
    ;弾丸の管理
    dim tamaF,10        ;有無フラグ
    dim tamaX,10        ;弾丸の横軸
    dim tamaY,10        ;弾丸の縦軸
    ;敵機の管理
    dim enemyF,20       ;有無フラグ
    dim enemyX,20       ;敵機の横軸
    dim enemyY,20       ;敵機の縦軸
    dim enemyZ,20       ;敵機の爆発カウンタ
    ;流星の管理
    dim starF,50        ;有無フラグ
    dim starX,50        ;流星の横軸
    dim starY,50        ;流星の縦軸
    ;スコアの初期化
    Score=0
    HiScore=9900
    ;自機の初期化
    x=(600-50)/2
    y=(400-50)-16
    fight=3
    blast=0
*Start
    screen 0,600,400,SCREEN_FIXEDSIZE
    title "シューティング・ゲームのミニ講座"
    redraw 0
    color:boxf
    color $00,$FF,$FF:font "HG明朝E",50:pos (600-50*11)/2,100:mes "シューティング・ゲーム"
    color $FF,$FF,$00:font "HG明朝E",30:pos (600-30* 6)/2,180:mes "〜ミニ講座〜"
    color $FF,$FF,$FF:font "HG明朝E",16:pos (600- 8*15)/2,250:mes "[SPC]キーで開始"
    color $FF,$FF,$FF:font "HG明朝E",16:pos (600- 8*15)/2,282:mes "[ESC]キーで終了"
    redraw 1
    repeat
        stick key
        if(key & $10):break
        if(key & $80):end
        await 20
    loop
*Main
    font MSGOTHIC,50
    randomize
    repeat
        redraw 0
```

```
        stick key,%11111
        gosub *EnemyBirth
        gosub *StarBirth
        gosub *StarDraw
        gosub *FightDraw
        gosub *EnemyDraw
        gosub *TamaDraw
        gosub *TelopDraw
        redraw 1
        await (1000/60)
    loop
    stop
//-------------------------------------
// 自機の描画
//-------------------------------------
*FightDraw
    if(blast==0){
        if(key&1):x-=5:if(x<0):x=0
        if(key&2):y-=5:if(y<0):y=0
        if(key&4):x+=5:if(x>550):x=550
        if(key&8):y+=5:if(y>350):y=350
        if(key&16):gosub *TamaBirth
        color $00,$FF,$00:pos x,y:mes "山"
    }
    else:if(blast>1){
        blast--
        color $FF,$00,$00:pos x,y
        if(blast\10<5):mes "※":else:mes "＊"
    }
    else:if(fight){
        fight--
        blast=0
        x=(600-50)/2
        y=(400-50)-16
    }
    else{
        dialog "もう一度、ゲームを行いますか？",3,"ゲームオーバー"
        if(stat==7):end
        ;スコアの初期化
        Score=0
        ;自機の初期化
        x=(600-50)/2
        y=(400-50)-16
        fight=3
        blast=0
        ;配列の初期化
        dim starF,50
        dim tamaF,10
        dim enemyF,20
    }
    return
//-------------------------------------
// 流星の発生
//-------------------------------------
*StarBirth
    if(starCycle):starCycle--:return
    foreach starF
        if(starF(cnt)==0){
            starF(cnt)=1
            starX(cnt)=rnd(600)
            starY(cnt)=0
            break
        }
    loop
    starCycle=3
    return
//-------------------------------------
// 流星の描画
```

```
//-------------------------------------
*StarDraw
    color $00,$00,$00:boxf
    foreach starF
        if starF(cnt){
            starY(cnt)+=2:if(starY(cnt)>=400):starF(cnt)=0:continue
            color $FF,$FF,$00
            pset starX(cnt),starY(cnt)
        }
    loop
    return
//-------------------------------------
// 弾丸の発生
//-------------------------------------
*TamaBirth
    if(tamaTrigg):tamaTrigg--:return
    foreach tamaF
        if(tamaF(cnt)==0){
            tamaF(cnt)=1
            tamaX(cnt)=x
            tamaY(cnt)=y
            break
        }
    loop
    tamaTrigg=8
    return
//-------------------------------------
// 弾丸の描画
//-------------------------------------
*TamaDraw
    foreach tamaF
        if tamaF(cnt){
            tamaY(cnt)-=8:if(tamaY(cnt)<-50):tamaF(cnt)=0:continue
            pos tamaX(cnt),tamaY(cnt)
            color $FF,$FF,$00:mes ":"
        }
    loop
    return
//-------------------------------------
// 敵機の発生
//-------------------------------------
*EnemyBirth
    if(enemyCycle):enemyCycle--:return
    foreach enemyF
        if(enemyF(cnt)==0){
            enemyF(cnt)=1
            enemyX(cnt)=rnd(600/50)*50
            enemyY(cnt)=-50
            enemyZ(cnt)=0
            break
        }
    loop
    enemyCycle=30
    return
//-------------------------------------
// 敵機の描画
//-------------------------------------
*EnemyDraw
    foreach enemyF
        if enemyF(cnt){
            if(enemyZ(cnt)==0){
                enemyY(cnt)+=3:if(enemyY(cnt)>400)      :enemyF(cnt)=0:continue
                if FightCrash(enemyX(cnt),enemyY(cnt))  :enemyZ(cnt)=60:ScoreCalc 100:continue
                if  TamaCrash(enemyX(cnt),enemyY(cnt))  :enemyZ(cnt)=60:ScoreCalc 100:continue
                pos enemyX(cnt),enemyY(cnt)
                color $00,$FF,$FF:mes "Ж"
            }
            else:if(enemyZ(cnt)>1){
```

```
                enemyZ(cnt)--
                color $FF,$00,$00:pos enemyX(cnt),enemyY(cnt)
                if(enemyZ(cnt)\10<5):mes "※":else:mes "＊"
            }
            else{
                enemyF(cnt)=0
            }
        }
    loop
    return
//-------------------------------------
// テロップの描画
//-------------------------------------
*TelopDraw
    msg=""
    repeat fight
        msg+="山"
    loop
    y(1)=0:x(1)=(20)
    y(2)=0:x(2)=(600-20)-(16*11)
    x(3)=0:y(3)=(400-20)
    font MSGOTHIC,20,1
    color $FF,$FF,$00:pos x(1),y(1):mes strf("Score:%08d",Score)
    color $FF,$FF,$00:pos x(2),y(2):mes strf("HiScore:%08d",HiScore)
    color $00,$FF,$00:pos x(3),y(3):mes msg
    font MSGOTHIC,50
    return
//-------------------------------------
// スコアの加算
//-------------------------------------
#deffunc ScoreCalc int _score_
    Score+=_score_
    if(Score>HiScore):HiScore=Score
    return
//-------------------------------------
// 自機の衝突判定
//-------------------------------------
#defcfunc FightCrash int _x_,int _y_
    if(blast==0)and(abs(x-_x_)<50)and(abs(y-_y_)<50){
        blast=60
        return 1
    }
    return 0
//-------------------------------------
// 自機弾の衝突判定
//-------------------------------------
#defcfunc TamaCrash int _x_,int _y_
    n=0
    foreach tamaF
        if tamaF(cnt){
            if(abs(tamaX(cnt)-_x_)<50)and(abs(tamaY(cnt)-_y_)<50){
                tamaF(cnt)=0
                n=1
                break
            }
        }
    loop
    return n
//-------------------------------------------------------------------------
// End of lesson-9.hsp
//-------------------------------------------------------------------------
```

どうですか？全部で 257 行になりました。コメント行が多いので実際には 202 行ですが...。

9.4 変数・配列の一覧

以下にミニ講座で使用してる変数と配列の一覧を表示します。変数が 12 個、配列が 10 個の合計で 22 個です。簡単なシューティング・ゲームですが 22 個も変数を使ってますね。これは、多い方なのでしょうか、それとも少ない方なのでしょうか。２次元配列にすれば、配列の個数を減らせるという噂もあります。この方法は、また別の機会にでも紹介しましょう。

表(9.4.1)

名前	意味
スコアの管理	
dim Score	現在スコア
dim HiScore	最高スコア
自機の管理	
dim x	自機の横軸
dim y	自機の縦軸
dim fight	自機の残機数
dim blast	自機の爆発カウンタ
弾丸の管理	
dim tamaF,10	有無フラグ
dim tamaX,10	弾丸の横軸
dim tamaY,10	弾丸の縦軸
dim tamaTrigg	弾丸の発生間隔
敵機の管理	
dim enemyF,20	有無フラグ
dim enemyX,20	敵機の横軸
dim enemyY,20	敵機の縦軸
dim enemyZ,20	敵機の爆発カウンタ
dim enemyCycle	敵機の発生間隔
流星の管理	
dim starF,50	有無フラグ
dim starX,50	流星の横軸
dim starY,50	流星の縦軸
dim starCycle	流星の発生間隔
作業用の変数	
dim key	キー入力
dim n	整数型の一時変数
sdim msg	文字型の一時変数

9.5 サブルーチン(命令,関数)の一覧

続いてミニ講座で使用してるサブルーチン(命令,関数)の一覧を表示します。サブルーチンが 11 個、ユーザ定義命令が 1 個、ユーザ定義関数が 2 個の合計 14 個です。簡単なシューティング・ゲームですが 14 個もサブルーチンなどを使ってます。これは、多い方なのでしょうか、それとも少ない方なのでしょうか。今回、移動と描画を一体化してるので、サブルーチンの数は少ない方だと思います。なお、サブルーチンの個数は、分かりやすい程度に分割して決めます。多くても少なくてもシューティング・ゲームのソース・コードを管理しづらくなります。また、ミニ講座なので 1 つの巨大なファイルにしました。本格的なシューティング・ゲーム(パワーアップ型)にする場合は、複数のファイルに分割して 1 本のゲームを作成します。この方法は、また別の機会にでも紹介しましょう。

表(9.5.1)

名前	処理
メイン部	
*Init	配列の確保、スコアの初期化、自機の初期化
*Start	スタート画面の処理
*Main	ゲームループの処理
自機の処理	
*FightDraw	自機の移動、描画、爆発、ゲームオーバー、コンティニュー処理
FightCrash(x,y)	自機の衝突判定
流星の処理	
*StarBirth	流星の発生
*StarDraw	流星の移動、描画
弾丸の処理	
*TamaBirth	弾丸の発生
*TamaDraw	弾丸の移動、描画
TamaCrash(x,y)	弾丸の衝突判定
敵機の処理	
*EnemyBirth	敵機の発生
*EnemyDraw	敵機の移動、描画、爆発、当たり判定
その他	
*TelopDraw	スコア、最高スコア、残機の描画
ScoreCalc score	スコアの加算

上記のサブルーチン(命令,関数)の見分け方は、次のようになります。

- 「*」マークがあるのがサブルーチン
- 引数が付いてるのがユーザ定義命令(ScoreCalc)
- カッコがあるのがユーザ定義関数(FightCrash、TamaCrash)

9.6 おわりに

今回は、シューティング・ゲームのミニ講座なので、敵機弾、アイテムは登場しませんでした。また、文字ベースで自機、弾丸、敵機、爆風などを描いてるため見栄えは良くありませんでしたね。しかし、シューティング・ゲームの全体的な仕組みは理解できたと思います。最初から見た目のデザイン、ゲームの操作性、ボス・キャラ、ステージ・クリアを考えると「頭」がパンクすると思います。そこでワザと文字ベースで自機、弾丸、敵機、爆風だけを登場させました。