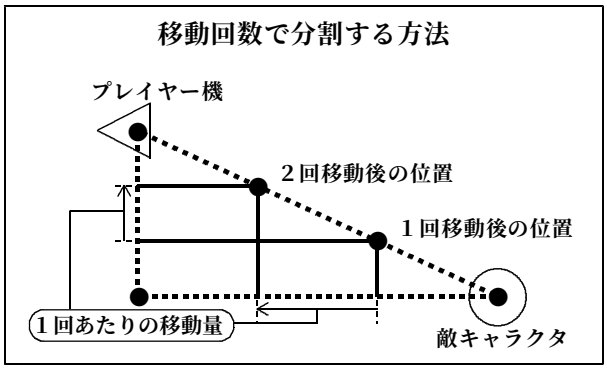
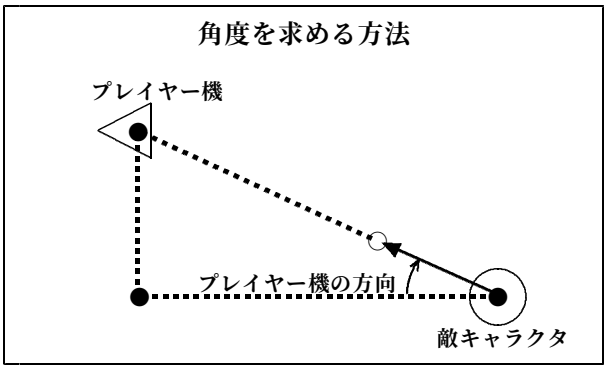
Game Algorithm

▶ 第４回　プレイヤーを狙う弾のアルゴリズム ◀

プレイヤーを狙う弾

プレイヤー機の方向へ弾を撃つ場合、その狙い方は、大きく分けて２とおりあります。
１つは、プレイヤー機と敵キャラクターとの座標の差を移動回数で分割して、速度とするものです。実際にキャラクターを表示するグラフィック画面は、横方向のｘ座標と縦方向のｙ座標を利用しているため、この方式は比較的簡単に使えます。
もう１つは、敵キャラクターからプレイヤー機を見た場合の方向(角度)を求めて、その方向に移動する速度を計算するという方法があります。これは、実際に人間が何かを狙うときの発想と同じなので、考え方としてなじみやすいはずです。しかし、この方式をプログラムへ適用しようとすると、"方向"というものを取り扱わなければならないために、プログラミングがやや難しくなります。ですが、この「方向」の概念を利用すると、最初のアルゴリズムでは難しい弾の移動が実現できるようになります。





"角度"を計算で求める

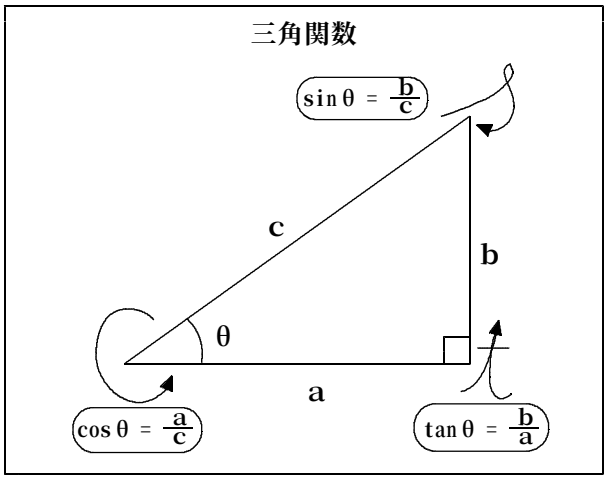
方向の概念を利用してターゲットを狙うアルゴリズムには、「ターゲットがどの方向に見えるか」という要素が欠かせません。
２次元の場合は、方向というのは"角度"のパラメータを利用すれば数値として処理できます。よって、プログラム的には「角度を計算してその方向を狙う」という処理になります。
ここで、敵キャラクタからプレイヤー機を狙う弾の移動速度を計算することを考えてみましょう。２次元のゲームを実際に表現するのは画面座標系なので、最終的にはｘ座標方向への速度とｙ座標方向への速度の２つの数値を計算するのが目的になります。
ほとんどの場合、プレイヤー機と敵キャラクタの座標しかありません。これから角度を計算するためには"三角関数"と呼ばれる関数を利用しなければなりません。

三角関数

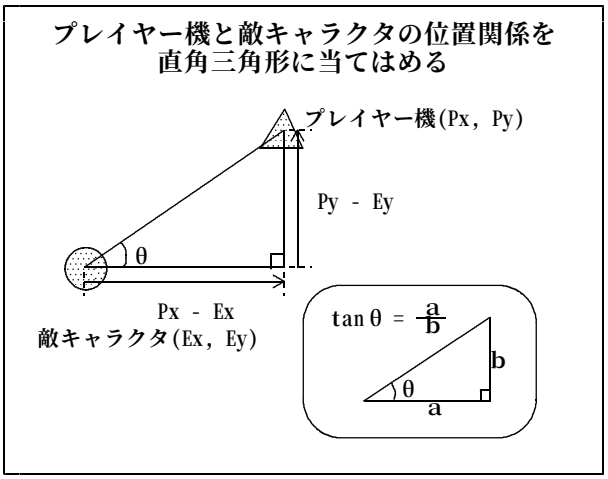
三角関数には、

・sin（サイン）
・cos（コサイン）
・tan（タンジェント）

の３種類があります。これらはすべて「直角三角形の特定の頂点の角度から、辺の長さの比を計算するもの」です。



▲角度θに対して、sin/cos/tanは
このような辺の比率になります



▲位置関係に直角三角形を当てはめて、
三角関数から角度を求めましょう

この中で、今回使えそうなものは、tanです。ｘ座標の差とｙ座標の差、そして角度θが、

$$\tan\theta = \frac{\text{プレーヤー機のｙ座標} - \text{敵キャラクターのｙ座標}}{\text{プレーヤー機のｘ座標} - \text{敵キャラクターのｘ座標}}$$

というようにtanによって関連づけられます。

ただしtanは、あくまでも「角度を入力すると辺の長さの比が出力される」という関数です。プレイヤー機の方向を調べるためには、座標の差から計算した「辺の長さの比」から角度を計算しなくてはならないため、逆の動作が必要です。

このために、tanの"逆関数"を利用します。逆関数というのは入力と出力の関係をひっくり返した関数です。

tan自体は、角度を入力すると辺の長さの比が出力されます。tanの逆関数は、辺の長さの比を入力すると、角度が出力されます。このような逆関数があれば、

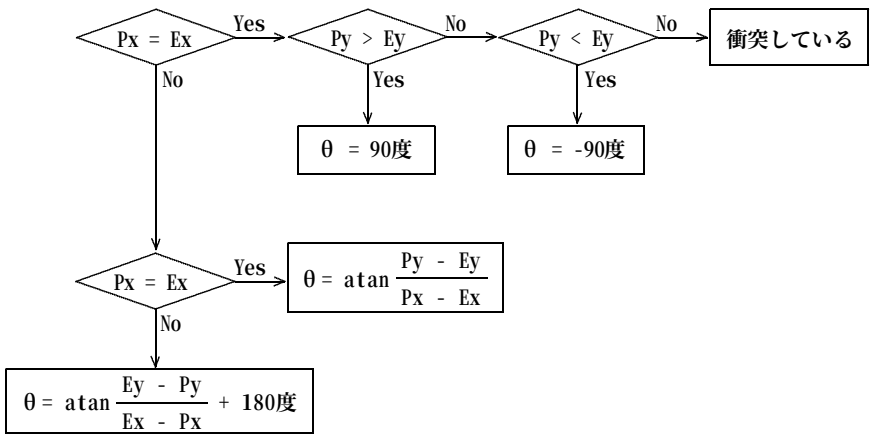
$$\theta = \text{tanの逆関数}\left[\frac{\text{プレーヤー機のｙ座標} - \text{敵キャラクターのｙ座標}}{\text{プレーヤー機のｘ座標} - \text{敵キャラクターのｘ座標}}\right] \cdots ①$$

というように、ｙ座標の差をｘ座標の差で割った数値から、角度θを計算できます。つまり、プレイヤー機の方向が角度θとして得られるわけです。

Ｃ言語では、tanの逆関数がatan関数(arctan:アークタンジェント)、atan2関数として用意されているので、これをそのまま利用できます。これらの関数の返値の単位はラジアンです。範囲は、atan関数が「π／２ ～ －π／２」(９０度～－９０度)、atan2関数が「π ～ －π」(１８０度～－１８０度)となります。

また、arctanの計算上、分母(ｘ座標の差)が０になる場合があります。当然、この場合は計算できないのでエラーになってしまいます。

このような場合に対応するため、以下のようにあらかじめ場合わけをして、専用の処理を行わせます。



プレイヤー機と敵キャラクタのｘ座標が等しい場合は、プレイヤー機が敵キャラクタの真上か真下のどちらかに存在することになります。そこで、ｙ座標の大小関係を調べて、「＋９０度」か「－９０度」かを判断します。
プレイヤー機のｘ座標が敵キャラクタのｘ座標より小さい条件では、まず基準となるキャラクタを敵キャラクタからプレイヤー機に変更し、プレイヤー機から見た敵キャラクタの方向を式①で計算します。プレイヤー機から見た敵キャラクタの方向と敵キャラクタから見たプレイヤー機の方向は、ちょうど反対になっているので、１８０度ずらします。

なお、ほとんどのプログラミング言語の三角関数は、角度をすべて"ラジアン"という単位で処理します。ここでの説明や図では"度"の単位になっていますが、実際にプログラムを作成する場合にはラジアンに対応させる必要があります。
ラジアンは、１回転を「２π」で表します。度は「３６０度」で１回転ですが、この３６０という数値には、実は根拠がありません。これに対してラジアンは、半径が１の円の円周と同じ２πが１回転なので、明確な根拠があります。この特徴を利用した近似計算が、いたるところで使われています。

ｘ／ｙ方向の移動速度を計算する

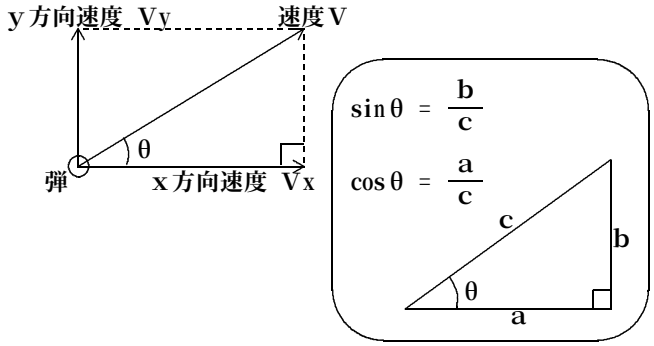
これまで説明したように、プレイヤー機の方向は角度という形で得られます。次は、この角度から実際に画面上をキャラクタが移動するのに必要な、

・ｘ方向の速度　Vx
・ｙ方向の速度　Vy

の２つを計算してみましょう。
このような処理には、三角関数のうちsinとcosを利用します。sinとcosともに、前述のように「角度から辺の長さの比を計算する」関数です。以下の図のように当てはめてやると、速度を１とした場合のｘ方向の速度はcosで、ｙ方向の速度はsinでそれぞれ計算できまることがわかります。
計算結果は、進行方向の速度を１としたものなので、これらの結果に実際の弾の速度Ｖを掛けてやれば、それぞれの座標軸方向への実際の速度が求まります。
よって、

$$Vx = V \cdot \cos\theta$$
$$Vy = V \cdot \sin\theta$$

とすることで、各座標軸方向への移動速度が計算できるわけです。



▲ｘ方向／ｙ方向への移動量

放射状に拡散する弾を実現しよう

方向の概念を使って実現しやすくなる弾の移動表現として、放射状に広がっていく弾があります。プレイヤー機を直接狙う弾の角度θをもとに、プラス方向とマイナス方向に角度αだけずらした弾を同時に発生させることによって、分裂して拡散する弾が作れます。
このような表現を「移動回数で分割する」アルゴリズムで実現しようとしても、ほとんど不可能です。「直接狙う弾をひとつだけ用意すればよい」という条件では、このアルゴリズムのほうが簡単です。しかし、ちょっとしたアレンジを付け加える場合には、角度を利用して取り扱うほうがはるかに有利になります。弾の個性に合わせて、アルゴリズムの選択をすればよいでしょう。

§ 課題 §

以下のプログラムは、砲台から発射される弾をよける「エスケープゲーム」です。左下の砲台からは、プレイヤー機を追ってくる「追尾弾」が発射されますが、それを処理するためのプログラムがないので直進するだけになっています。追尾弾のプログラムを追加しましょう。

```
/*
================================================================================================
                                        エスケープゲーム
    Programmed by Hibikino software. Copyright (c) 2003 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
    Microsoft Windows98/2000以降

  【コンパイラ】
    Microsoft VisualC++ 6.0J ServicePack5

  【プログラム】
    EscapeGame.cpp
                エスケープゲーム

  【履歴】
    * Version    1.00       2003/08/26  00:00:00  初版

================================================================================================
*/

/************************************************************************************************/
/*                                        インクルードファイル                                  */
/************************************************************************************************/
#include "GameFunc.h"
#include "DDUtils.h"
#include "DIUtils.h"
#include "DXAUtils.h"

#include <math.h>
#include <list>

/************************************************************************************************/
/*                                          列挙体定義                                          */
/************************************************************************************************/
enum SHOT_KIND {
    NONE,       // なし
    NORMAL,     // 通常弾
    HISP,       // 高速弾
    SPLIT,      // 分裂弾
    TRACKING    // 追尾弾
};

/************************************************************************************************/
/*                                          構造体定義                                          */
/************************************************************************************************/
// 座標(double)
struct POINTD {
    double   x;
    double   y;
};

// 領域(double)
struct RECTD {
    double   left;
    double   top;
    double   right;
    double   bottom;
};

// プレイヤー
struct PLAYER {
    RECTD    rect;      // 領域
};

// 弾
```

```
struct SHOT {
    SHOT_KIND   kind;   // 種類
    RECTD       rect;   // 領域
    double      add_x;  // ｘ方向増分
    double      add_y;  // ｙ方向増分
};

// 砲台
struct BATTERY {
    RECTD       rect;   // 領域
    SHOT_KIND   kind;   // 発射する弾の種類
    DWORD       aves;   // 弾を発射する確率
};

/*****************************************************************************************************/
/*                                          型定義型                                                 */
/*****************************************************************************************************/
typedef std::list<SHOT>             shot_list;
typedef std::list<SHOT>::iterator   shot_list_it;

/*****************************************************************************************************/
/*                                          外部参照変数                                             */
/*****************************************************************************************************/
extern GameFuncPtr   GameFunc;

/*****************************************************************************************************/
/*                                          グローバル変数                                           */
/*****************************************************************************************************/
static int          g_nCnt        = 0;      // 汎用カウンタ
static int          g_nScore      = 0;      // スコア
static int          g_nHighScore  = 0;      // ハイスコア

static PLAYER       g_Player;               // プレイヤー
static shot_list    g_Shot;                 // ショット
static BATTERY      g_Battery[4];           // 砲台

/*****************************************************************************************************/
/*                                          定　　数                                                 */
/*****************************************************************************************************/
static const double   PI           = 3.14159265358979323846264338327950;
static const RECT     MOVENABLE    = {0, 32, 640, 480};
static const double   SHOT_SPEED[] = {0.0, 1.0, 4.0, 0.5, 1.0};

/*****************************************************************************************************/
/*                                          インライン関数                                           */
/*****************************************************************************************************/
// 度をラジアンに変換
inline double ToRadian(const double degree)
{
    return degree * PI / 180.0;
}

// ラジアンを度に変換
inline double ToDegree(const double radian)
{
    return radian * 180.0 / PI;
}

// 領域の移動
inline void OffsetRect(RECTD& rect, const double x, const double y)
{
    rect.left += x;     rect.right  += x;
    rect.top  += y;     rect.bottom += y;
}

/*****************************************************************************************************/
/*                                          プロトタイプ(プライベート)                               */
/*****************************************************************************************************/
static void EscapeMain();

static void PlayerProc(PLAYER& player, const BYTE byKeyState[]);
static void ShotProc(shot_list& shot, const PLAYER& player);
static void BatteryProc(BATTERY battery[], shot_list& shot, const PLAYER& player);
```

```
static bool CollisionDetection(PLAYER& player, shot_list& shot, BATTERY battery[]);

static void DrawEscape (const HDC hDC);
static void DrawBG     (const HDC hDC);
static void DrawPlayer (const HDC hDC, PLAYER& player);
static void DrawShot   (const HDC hDC, shot_list& shot);
static void DrawBattery(const HDC hDC, BATTERY battery[]);

static void GameOver();
static void GameOverProc();

/*****************************************************************************************************/
/*                                          エスケープゲーム                                         */
/*****************************************************************************************************/
void Escape()
{
    srand(GetTickCount());

    g_nCnt   = 0;
    g_nScore = 0;

    // プレイヤー初期化
    g_Player.rect.left   = 312.0;
    g_Player.rect.top    = 232.0;
    g_Player.rect.right  = g_Player.rect.left + 16.0;
    g_Player.rect.bottom = g_Player.rect.top  + 16.0;

    // ショット初期化
    g_Shot.clear();

    // 砲台初期化
    g_Battery[0].rect.left   = MOVENABLE.left;
    g_Battery[0].rect.top    = MOVENABLE.top;
    g_Battery[0].rect.right  = g_Battery[0].rect.left + 16.0;
    g_Battery[0].rect.bottom = g_Battery[0].rect.top  + 16.0;
    g_Battery[0].kind        = NORMAL;
    g_Battery[0].aves        = FPS;

    g_Battery[1].rect.left   = MOVENABLE.right - 16.0;
    g_Battery[1].rect.top    = MOVENABLE.top;
    g_Battery[1].rect.right  = g_Battery[1].rect.left + 16.0;
    g_Battery[1].rect.bottom = g_Battery[1].rect.top  + 16.0;
    g_Battery[1].kind        = SPLIT;
    g_Battery[1].aves        = FPS * 3;

    g_Battery[2].rect.left   = MOVENABLE.right  - 16.0;
    g_Battery[2].rect.top    = MOVENABLE.bottom - 16.0;
    g_Battery[2].rect.right  = g_Battery[2].rect.left + 16.0;
    g_Battery[2].rect.bottom = g_Battery[2].rect.top  + 16.0;
    g_Battery[2].kind        = HISP;
    g_Battery[2].aves        = FPS * 1.5;

    g_Battery[3].rect.left   = MOVENABLE.left;
    g_Battery[3].rect.top    = MOVENABLE.bottom - 16.0;
    g_Battery[3].rect.right  = g_Battery[3].rect.left + 16.0;
    g_Battery[3].rect.bottom = g_Battery[3].rect.top  + 16.0;
    g_Battery[3].kind        = TRACKING;
    g_Battery[3].aves        = FPS * 20;

    GameFunc = EscapeMain;  // 初期化が終わったら、メインループ関数へ
}

/*****************************************************************************************************/
/*                                      エスケープゲームメイン処理                                   */
/*****************************************************************************************************/
void EscapeMain()
{
    g_nCnt = (g_nCnt + 1) % FPS;
    if(0 == g_nCnt % FPS)
        g_nScore++;

    // キー入力
    BYTE   byKeyState[256];
    DIGetKeyboardState(byKeyState);
```

```
    PlayerProc(g_Player, byKeyState);           // プレイヤー処理
    ShotProc(g_Shot, g_Player);                 // ショット処理
    BatteryProc(g_Battery, g_Shot, g_Player);   // 砲台処理

    // 衝突判定
    const bool   bGameOver = CollisionDetection(g_Player, g_Shot, g_Battery);

    // 画面構築
    DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));
    const HDC   hDC = DDGetDC(DDS_BACKBUF);
    DrawEscape(hDC);
    DDReleaseDC(DDS_BACKBUF);

    DDFlip();
    WaitFrame();

    // ゲームオーバー判定
    if(true == bGameOver)
        GameFunc = GameOver;
}

/*****************************************************************************************************/
/*                                          プレイヤー処理                                           */
/*****************************************************************************************************/
void PlayerProc(PLAYER& player, const BYTE byKeyState[])
{
    // キー状態設定
    const int   LR = (byKeyState[DIK_RIGHT] & 0x80) - (byKeyState[DIK_LEFT] & 0x80);
    const int   UD = (byKeyState[DIK_DOWN ] & 0x80) - (byKeyState[DIK_UP  ] & 0x80);

    // 角度設定
    double   dAngle = -1.0;
    if(0 == LR) {
        if(0 == UD)     return;
        else if(0 < UD) dAngle =  90.0;
        else            dAngle = 270.0;
    } else if(0 < LR) {
        if(0 == UD)     dAngle =   0.0;
        else if(0 < UD) dAngle =  45.0;
        else            dAngle = 315.0;
    } else {
        if(0 == UD)     dAngle = 180.0;
        else if(0 < UD) dAngle = 135.0;
        else            dAngle = 225.0;
    }
    dAngle = ToRadian(dAngle);

    const double   SPEED = 3.0;
    OffsetRect(player.rect, SPEED * cos(dAngle), SPEED * sin(dAngle));  // 移動

    // 移動限界処理
    const double   WIDTH = player.rect.right - player.rect.left;
    if(MOVENABLE.left > player.rect.left) {
        player.rect.left  = MOVENABLE.left;
        player.rect.right = player.rect.left + WIDTH;
    } else if(MOVENABLE.right < player.rect.right) {
        player.rect.right = MOVENABLE.right;
        player.rect.left  = player.rect.right - WIDTH;
    }

    const double   HEIGHT = player.rect.bottom - player.rect.top;
    if(MOVENABLE.top > player.rect.top) {
        player.rect.top    = MOVENABLE.top;
        player.rect.bottom = player.rect.top + HEIGHT;
    } else if(MOVENABLE.bottom < player.rect.bottom) {
        player.rect.bottom = MOVENABLE.bottom;
        player.rect.top    = player.rect.bottom - HEIGHT;
    }
}

/*****************************************************************************************************/
/*                                            弾処理                                                 */
/*****************************************************************************************************/
```

```
void ShotProc(shot_list& shot, const PLAYER& player)
{
    shot_list_it   it = shot.begin();
    while(it != shot.end()) {
        // 分裂弾処理
        if(SPLIT == (*it).kind) {
            // 弾とプレイヤーの距離を求める
            const double   SHOT_X     = ((*it).rect.right   + (*it).rect.left ) / 2.0;
            const double   SHOT_Y     = ((*it).rect.bottom  + (*it).rect.top  ) / 2.0;

            const double   PLAYER_X   = (player.rect.right  + player.rect.left) / 2.0;
            const double   PLAYER_Y   = (player.rect.bottom + player.rect.top ) / 2.0;

            const double   DISTANCE_X = PLAYER_X - SHOT_X;
            const double   DISTANCE_Y = PLAYER_Y - SHOT_Y;
            const double   DISTANCE   = sqrt(DISTANCE_X * DISTANCE_X + DISTANCE_Y * DISTANCE_Y);

            // 分裂判定
            if(150.0 > DISTANCE) {
                it = shot.erase(it);    // 分裂弾を消す

                // 分裂弾とプレイヤーの角度を求める
                double   dAngle;
                if(0.0 != DISTANCE_Y)
                    dAngle = ToDegree(atan2(DISTANCE_Y, DISTANCE_X));
                else
                    dAngle = 0.0;
                dAngle -= 20.0;

                // 分裂した弾を追加する
                SHOT   new_shot;
                new_shot.kind        = NORMAL;
                new_shot.rect.left   = SHOT_X;
                new_shot.rect.top    = SHOT_Y;
                new_shot.rect.right  = new_shot.rect.left + 4.0;
                new_shot.rect.bottom = new_shot.rect.top  + 4.0;
                for(int i = 0; i < 5; i++) {
                    const double   ANGLE = ToRadian(dAngle + i * 10.0);
                    new_shot.add_x = 1.0 * cos(ANGLE);
                    new_shot.add_y = 1.0 * sin(ANGLE);
                    shot.push_back(new_shot);
                }
                continue;
            }
        }

        OffsetRect((*it).rect, (*it).add_x, (*it).add_y);   // 移動

        // 移動可能領域外に出たら消す
        RECT   rcIntersect, rcShot;
        rcShot.left   = (int)(*it).rect.left;
        rcShot.top    = (int)(*it).rect.top;
        rcShot.right  = (int)(*it).rect.right;
        rcShot.bottom = (int)(*it).rect.bottom;
        if(0 == IntersectRect(&rcIntersect, &MOVENABLE, &rcShot))
            it = shot.erase(it);
        else
            it++;
    }
}

/*********************************************************************************************/
/*                                          砲台処理                                         */
/*********************************************************************************************/
void BatteryProc(BATTERY battery[], shot_list& shot, const PLAYER& player)
{
    const double   PLAYER_X = (player.rect.right  + player.rect.left) / 2.0;
    const double   PLAYER_Y = (player.rect.bottom + player.rect.top ) / 2.0;

    for(int i = 0; i < 4; i++) {
        if(0 == rand() % battery[i].aves) {
            // 発射角度設定
            const double   BATTERY_X  = (battery[i].rect.right  + battery[i].rect.left) / 2.0;
            const double   BATTERY_Y  = (battery[i].rect.bottom + battery[i].rect.top)  / 2.0;
```

```
            const double   DISTANCE_X = PLAYER_X - BATTERY_X;
            const double   DISTANCE_Y = PLAYER_Y - BATTERY_Y;

            double   dAngle;
            if(0 != DISTANCE_Y)
                dAngle = atan2(DISTANCE_Y, DISTANCE_X);
            else
                dAngle = 0.0;

            // 弾生成
            SHOT   new_shot;
            new_shot.kind        = battery[i].kind;
            new_shot.rect.left   = BATTERY_X;
            new_shot.rect.top    = BATTERY_Y;
            new_shot.rect.right  = new_shot.rect.left + 8.0;
            new_shot.rect.bottom = new_shot.rect.top  + 8.0;
            new_shot.add_x       = SHOT_SPEED[new_shot.kind] * cos(dAngle);
            new_shot.add_y       = SHOT_SPEED[new_shot.kind] * sin(dAngle);
            shot.push_back(new_shot);
        }
    }
}

/*****************************************************************************************************/
/*                                              衝突検出                                             */
/*****************************************************************************************************/
bool CollisionDetection(PLAYER& player, shot_list& shot, BATTERY battery[])
{
    RECT   rcIntersect, chara1, chara2;

    chara1.left   = (int)player.rect.left;
    chara1.top    = (int)player.rect.top;
    chara1.right  = (int)player.rect.right;
    chara1.bottom = (int)player.rect.bottom;

    // プレイヤーと弾
    for(shot_list_it it = shot.begin(); it != shot.end(); it++) {
        chara2.left   = (int)(*it).rect.left;
        chara2.top    = (int)(*it).rect.top;
        chara2.right  = (int)(*it).rect.right;
        chara2.bottom = (int)(*it).rect.bottom;
        if(0 != IntersectRect(&rcIntersect, &chara1, &chara2))
            return true;
    }

    // プレイヤーと砲台
    for(int i = 0; i < 4; i++) {
        chara2.left   = (int)battery[i].rect.left;
        chara2.top    = (int)battery[i].rect.top;
        chara2.right  = (int)battery[i].rect.right;
        chara2.bottom = (int)battery[i].rect.bottom;
        if(0 != IntersectRect(&rcIntersect, &chara1, &chara2))
            return true;
    }

    return false;
}

/*****************************************************************************************************/
/*                                          エスケープゲーム描画                                     */
/*****************************************************************************************************/
void DrawEscape(const HDC hDC)
{
    // ライン(ペン)設定
    HPEN   hPen    = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
    HPEN   hDefPen = (HPEN)SelectObject(hDC, hPen);

    DrawBG     (hDC);                // 背景描画
    DrawPlayer (hDC, g_Player);      // プレイヤー描画
    DrawShot   (hDC, g_Shot);        // ショット描画
    DrawBattery(hDC, g_Battery);     // 砲台描画

    // ライン(ペン)解放
```

```
    SelectObject(hDC, hDefPen);
    DeleteObject(hPen);
}

/*****************************************************************************************************/
/*                                              背景描画                                             */
/*****************************************************************************************************/
void DrawBG(const HDC hDC)
{
    // 文字列描画
    SetTextColor(hDC, RGB(255, 255, 255));
    SetBkMode(hDC, TRANSPARENT);

    LPCTSTR   TITLE = "Escape Game";
    TextOut(hDC, 0, 0, TITLE, lstrlen(TITLE));

    TCHAR   szText[256];
    wsprintf(szText, "High Score : %4d      Score : %4d", g_nHighScore, g_nScore);
    TextOut(hDC, 200, 0, szText, lstrlen(szText));
}

/*****************************************************************************************************/
/*                                            プレイヤー描画                                         */
/*****************************************************************************************************/
void DrawPlayer(const HDC hDC, PLAYER& player)
{
    Rectangle(hDC, (int)player.rect.left, (int)player.rect.top,
              (int)player.rect.right, (int)player.rect.bottom);
}

/*****************************************************************************************************/
/*                                              弾描画                                               */
/*****************************************************************************************************/
void DrawShot(const HDC hDC, shot_list& shot)
{
    // 塗りつぶし(ブラシ)生成
    HBRUSH   hBrush[5];
    hBrush[0] = CreateSolidBrush(RGB(  0,   0,   0));
    hBrush[1] = CreateSolidBrush(RGB(  0,   0, 255));
    hBrush[2] = CreateSolidBrush(RGB(  0, 255,   0));
    hBrush[3] = CreateSolidBrush(RGB(255, 255,   0));
    hBrush[4] = CreateSolidBrush(RGB(255,   0,   0));

    HBRUSH      hDefBrush = (HBRUSH)SelectObject(hDC, hBrush[0]);
    SHOT_KIND   kind      = NONE;
    for(shot_list_it it = shot.begin(); it != shot.end(); it++) {
        // 塗りつぶし(ブラシ)設定
        if(kind != (*it).kind) {
            SelectObject(hDC, hBrush[(*it).kind]);
            kind = (*it).kind;
        }
        Rectangle(hDC, (int)(*it).rect.left, (int)(*it).rect.top,
                  (int)(*it).rect.right, (int)(*it).rect.bottom);
    }

    // 塗りつぶし(ブラシ)解放
    SelectObject(hDC, hDefBrush);
    for(int i = 0; i < 5; i++)
        DeleteObject(hBrush[i]);
}

/*****************************************************************************************************/
/*                                              砲台描画                                             */
/*****************************************************************************************************/
void DrawBattery(const HDC hDC, BATTERY battery[])
{
    // 塗りつぶし(ブラシ)設定
    HBRUSH   hBrush    = CreateSolidBrush(RGB(255, 0, 255));
    HBRUSH   hDefBrush = (HBRUSH)SelectObject(hDC, hBrush);

    for(int i = 0; i < 4; i++)
        Rectangle(hDC, (int)battery[i].rect.left, (int)battery[i].rect.top,
                  (int)battery[i].rect.right, (int)battery[i].rect.bottom);
```

```
    // 塗りつぶし(ブラシ)解放
    SelectObject(hDC, hDefBrush);
    DeleteObject(hBrush);
}

/*****************************************************************************************************/
/*                                          ゲームオーバー                                           */
/*****************************************************************************************************/
void GameOver()
{
    g_nCnt = 0;

    // ハイスコア処理
    if(g_nHighScore < g_nScore)
        g_nHighScore = g_nScore;

    GameFunc = GameOverProc;
}

/*****************************************************************************************************/
/*                                        ゲームオーバー処理                                         */
/*****************************************************************************************************/
void GameOverProc()
{
    // 画面構築
    DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));
    const HDC   hDC = DDGetDC(DDS_BACKBUF);
    DrawEscape(hDC);
    TextOut(hDC, 278, 232, "Game Over", 9);
    DDReleaseDC(DDS_BACKBUF);

    DDFlip();
    WaitFrame();

    // ５秒たったらメイン処理へ
    g_nCnt++;
    if(FPS * 5 <= g_nCnt)
        GameFunc = Escape;
}
```