Game Algorithm

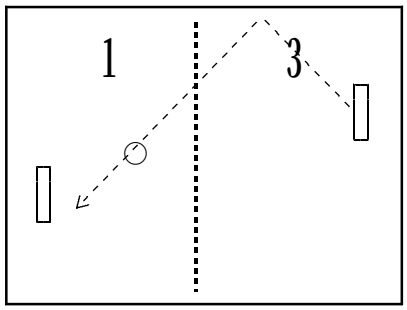
▶ 第１回　スカッシュゲーム ◀

ボールをパッドで打ち返すゲーム

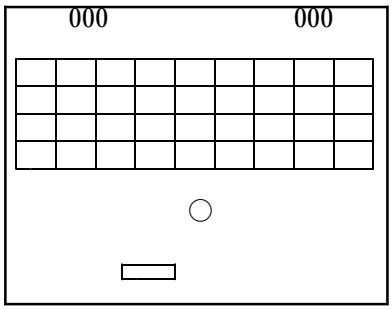
テレビゲームが登場したばかりの時代に流行したのが、「テニスゲーム」タイプのゲームです。このタイプのゲームは、プレイヤーがコントローラでパッドを上下に動かし、ボールを当てて跳ね返すだけという、非常に単純なゲームです。移動するものがパッドとボールだけであるため、当時の非力なハードウェアでも現実的な速度を実現することができました。

このテニスゲームを発展させたのが「ブロックくずし」です。これは、「パッドでボールを打ち返し、それをブロックに当てて破壊し、得点を稼ぐ」というものです。

今回は、これらのゲームに利用されているアルゴリズムを紹介します。



▲テニスゲーム



▲ブロックくずし

スカッシュゲーム

以下のプログラムは、パッドでボールを跳ね返す「スカッシュゲーム」です。プログラムを実行すると、画面にパッドとボールが表示されます。プレイヤーは、カーソルの左右と左シフトキーでバッドを操作し、ボールを跳ね返してください。ボールを１回跳ね返すごとに、得点が増加します。また、ボールは何かに跳ね返されるたびに、速度が上がっていきます。

```
/*
================================================================================================
                                        スカッシュ
    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】
     Squash.cpp
                スカッシュ

  【履歴】
     * Version     1.00       2003/08/05  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>

/*****************************************************************************************************/
/*                                          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;
};

// パッド
struct PAD {
    int      width;     // 幅
    RECTD    rect;      // 領域
};

// ボール
struct BALL {
    POINTD   pos;       // 座標
    double   r;         // 半径
    double   speed;     // スピード
    double   dir;       // 移動角度
    double   add_x;     // ｘ方向増分
    double   add_y;     // ｙ方向増分
};

/*****************************************************************************************************/
/*                                        外部参照変数                                               */
/*****************************************************************************************************/
extern GameFuncPtr   GameFunc;       // ゲーム関数へのポインタ

/*****************************************************************************************************/
/*                                          定　　数                                                 */
/*****************************************************************************************************/
static const double   PI          = 3.14159265358979323846264338327950;
static const int      WALL        = 16;
static const RECT     MOVENABLE   = {0 + WALL, 32 + WALL, 640 - WALL, 480};

/*****************************************************************************************************/
/*                                        グローバル変数                                             */
/*****************************************************************************************************/
static int       g_nScore      = 0;     // スコア
static int       g_nHighScore  = 0;     // ハイスコア

static PAD       g_Pad;                 // パッド
static BALL      g_Ball;                // 球

static int       g_nCnt        = 0;     // 汎用カウンタ

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

static void MovePad(PAD& pad, const BYTE byKeyState[]);
static void MoveBall(BALL& ball);
static void SetBallDir(BALL& ball, const double dDir);
static void CollisionDetection(PAD& pad, BALL& ball);

static void DrawSquash(const HDC hDC);
static void DrawBG(const HDC hDC);
static void DrawWall(const HDC hDC);
```

```
static void DrawPad(const HDC hDC, const PAD& pad);
static void DrawBall(const HDC hDC, const BALL& ball);

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

/*****************************************************************************************************/
/*                                          スカッシュ                                               */
/*****************************************************************************************************/
void Squash()
{
    g_nScore = 0;

    // パッド初期化
    g_Pad.width       = 60;
    g_Pad.rect.left   = 290.0;
    g_Pad.rect.top    = 460.0;
    g_Pad.rect.right  = g_Pad.rect.left + g_Pad.width;
    g_Pad.rect.bottom = g_Pad.rect.top  + 2.0;

    // ボール初期化
    g_Ball.pos.x = 320.0;
    g_Ball.pos.y = 430.0;
    g_Ball.r     =  10.0;
    g_Ball.speed =   1.0;
    SetBallDir(g_Ball, -60.0);

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

/*****************************************************************************************************/
/*                                      スカッシュメイン処理                                         */
/*****************************************************************************************************/
void SquashProc()
{
    // キー入力
    BYTE   byKeyState[256];
    DIGetKeyboardState(byKeyState);

    // 座標計算処理
    MovePad(g_Pad, byKeyState);
    MoveBall(g_Ball);

    // 衝突判定
    CollisionDetection(g_Pad, g_Ball);

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

    DDFlip();
    WaitFrame();

    // ゲームオーバー判定
    if(g_Ball.pos.y >= 480.0)
        GameFunc = GameOver;
}

/*****************************************************************************************************/
/*                                          パッド移動                                               */
/*****************************************************************************************************/
void MovePad(PAD& pad, const BYTE byKeyState[])
{
    // 増分設定
    double   dAdd = 0.0;
    if(0 != (byKeyState[DIK_LEFT  ] & 0x80))
        dAdd -= 2.0;
    if(0 != (byKeyState[DIK_RIGHT ] & 0x80))
        dAdd += 2.0;
    if(0 != (byKeyState[DIK_LSHIFT] & 0x80))
        dAdd *= 2.0;
```

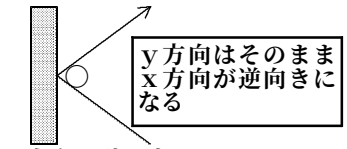
```
    pad.rect.left  += dAdd;
    pad.rect.right += dAdd;
}

/*****************************************************************************************************/
/*                                              球移動                                               */
/*****************************************************************************************************/
void MoveBall(BALL& ball)
{
    ball.pos.x += ball.add_x;
    ball.pos.y += ball.add_y;
}

/*****************************************************************************************************/
/*                                             衝突検出                                              */
/*****************************************************************************************************/
void CollisionDetection(PAD& pad, BALL& ball)
{
    const double   UP_SPEED = 0.02;     // スピード増加量

    // 壁とパッド
    if(MOVENABLE.left > pad.rect.left) {
        pad.rect.left  = MOVENABLE.left;
        pad.rect.right = pad.rect.left  + pad.width;
    } else if(MOVENABLE.right < pad.rect.right) {
        pad.rect.right = MOVENABLE.right;
        pad.rect.left  = pad.rect.right - pad.width;
    }

    // 壁と球
    const double   dDir = ball.dir;
    if(MOVENABLE.left >= ball.pos.x - ball.r || MOVENABLE.right <= ball.pos.x + ball.r) {
        SetBallDir(ball, 180.0 - dDir);
        ball.speed += UP_SPEED;
    } else if(MOVENABLE.top >= ball.pos.y - ball.r) {
        SetBallDir(ball, -dDir);
        ball.speed += UP_SPEED;
    }

    // パッドと球
    RECT   rcBall;
    rcBall.left   = (int)(ball.pos.x - ball.r);
    rcBall.top    = (int)(ball.pos.y - ball.r);
    rcBall.right  = (int)(ball.pos.x + ball.r);
    rcBall.bottom = (int)(ball.pos.y + ball.r);

    RECT   rcPad;
    rcPad.left   = (int)pad.rect.left;
    rcPad.top    = (int)pad.rect.top;
    rcPad.right  = (int)pad.rect.right;
    rcPad.bottom = (int)pad.rect.bottom;

    RECT   rcIntersect;
    if(IntersectRect(&rcIntersect, &rcPad, &rcBall)) {
        SetBallDir(ball, -ball.dir);
        g_nScore   += 1;
        ball.speed += UP_SPEED;
    }
}

/*****************************************************************************************************/
/*                                          球移動方向設定                                           */
/*****************************************************************************************************/
void SetBallDir(BALL& ball, const double dDir)
{
    // 角度を-360.0 ～ 360.0の範囲にする
    const int   w = (int)dDir / 360;
    ball.dir      = dDir - w * 360.0;

    ball.add_x = ball.speed * cos(ball.dir * PI / 180.0);
    ball.add_y = ball.speed * sin(ball.dir * PI / 180.0);
}

/*****************************************************************************************************/
```

```
/*                                          スカッシュ描画                                          */
/*******************************************************************************************************/
void DrawSquash(const HDC hDC)
{
    // ライン(ペン)設定
    HPEN   hPen    = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
    HPEN   hOldPen = (HPEN)SelectObject(hDC, hPen);

    DrawBG(hDC);                // 文字描画
    DrawWall(hDC);              // 壁描画
    DrawBall(hDC, g_Ball);      // 球描画
    DrawPad(hDC, g_Pad);        // パッド描画

    // ライン(ペン)解放
    SelectObject(hDC, hOldPen);
    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);

    TextOut(hDC, 0, 0, "Squash Game", 11);

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

/*******************************************************************************************************/
/*                                          壁描画                                                     */
/*******************************************************************************************************/
void DrawWall(const HDC hDC)
{
    Rectangle(hDC, MOVENABLE.left - WALL, MOVENABLE.top - WALL, MOVENABLE.right + WALL, MOVENABLE.top);
    Rectangle(hDC, MOVENABLE.left - WALL, MOVENABLE.top,        MOVENABLE.left,         480);
    Rectangle(hDC, MOVENABLE.right,       MOVENABLE.top,        MOVENABLE.right + WALL, 480);
}

/*******************************************************************************************************/
/*                                          パッド描画                                                 */
/*******************************************************************************************************/
void DrawPad(const HDC hDC, const PAD& pad)
{
    Rectangle(hDC, (int)pad.rect.left, (int)pad.rect.top, (int)pad.rect.right, (int)pad.rect.bottom);
}

/*******************************************************************************************************/
/*                                          球描画                                                     */
/*******************************************************************************************************/
void DrawBall(const HDC hDC, const BALL& ball)
{
    MoveToEx(hDC, (int)(ball.pos.x + ball.r), (int)ball.pos.y, NULL);
    AngleArc(hDC, (int)ball.pos.x, (int)ball.pos.y, (int)ball.r, 0, 360);
}

/*******************************************************************************************************/
/*                                          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;
}
```

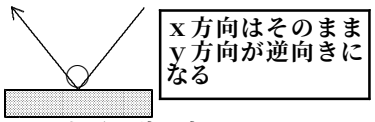
```
/*****************************************************************************************************/
/*                                          ゲームオーバー処理                                       */
/*****************************************************************************************************/
void GameOverProc()
{
    // 画面構築
    DDColorFill(DDS_BACKBUF, NULL, RGB(0, 0, 0));
    const HDC   hDC = DDGetDC(DDS_BACKBUF);
    DrawSquash(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 = Squash;
}
```

跳ね返りアルゴリズム

「テニスゲーム」や「ブロックくずし」でポイントになるのが、”跳ね返りアルゴリズム”です。基本的な跳ね返りの処理は、ボールの移動方向に対してｘ座標もしくはｙ座標についてのみ移動方向を逆にすれば実現できます。

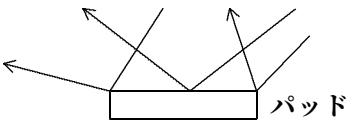


▲左右の壁に当たったとき



▲上下の壁に当たったとき

しかし、このままではボールの移動方向が特定の角度だけになってしまい、面白味がありません。ほとんどのゲームでは、この跳ね返り処理には工夫がなされています。壁での跳ね返りは、前述の方法そのままですが、パッドの跳ね返りにはバリエーションを持たせています。たとえば、パッドとボールの当たる位置によって、反射する角度が変化するようになっています。パッドとボールの中心のズレによって、反射角が変化するようにすることで、戦略性が高くなります。



§ 課題 §

１．ボールがパッドや壁に当たったとき、効果音をならすようにしましょう。

２．現在のプログラムでは、パッドにボールが当たったとき、単純にｙ方向が逆向きになるように跳ね返しています。プログラムを改良し、パッドとボールの中心のズレに応じて、ボールの反射角を変化するようにしましょう。

３．プログラムを改良し、パッドを１つ増やして２人で協力してプレイするダブルス(またはもう一方のパッドは左右逆に動くようにする)にしましょう。

４．プログラムを改良し、２人でプレイする「テニスゲーム」にしましょう。