Game Algorithm

▶ 第５回　仮想画面によるキャラクタの通路移動アルゴリズム ◀

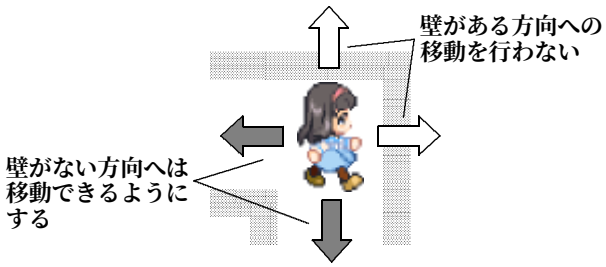
通路の移動

２次元のゲームでは、平面のフィールドの中で、上下左右にキャラクタを操作して移動させます。このタイプのゲームには、

①通路が存在して、移動がかなり限定されているもの
②障害物はとくに設定せず、自由に移動できるもの

という２種類があります。今回は、このうち①の「通路が存在して移動範囲が限定されている」パターンで進めていきます。

①のパターンのゲームの代表格といえば、パックマンや平安京エイリアンです。とくに前者は世界的な人気があり、現在でもリメイクされて発売されています。
これらのゲームでは、登場するキャラクタは、壁によって構成された通路上を移動します。このような動作を行わせるには、キャラクタの移動処理をするときに、壁の存在する方向へ移動しないようにする必要があります。
たとえば、以下の図を見てください。このように、キャラクタの上と右に壁が存在する場合、上や右に移動しないように判断する必要があります。



壁の存在の判断方法にはいろいろな方法がありますが、今回は"仮想画面"を用いた方法を使ってみます。この方式は、壁とキャラクタの大きさが同じ場合に有効で、プログラム的にも簡単になります。

仮想画面とは

仮想画面による通路の判断では、キャラクタの大きさを１単位とした仮想画面を用意する必要があります。
この仮想画面とは､見た目は複雑なグラフィックになっているキャラクタを単純な数値に置き換えて、そのキャラクタの存在箇所に対応する数値を記録したものです。こうすることで、場所を指定するとそこにどのキャラクタが存在するか、即座に判断できるようになります。
サンプルとして右の図を見てください。このように、登場するキャラクタが

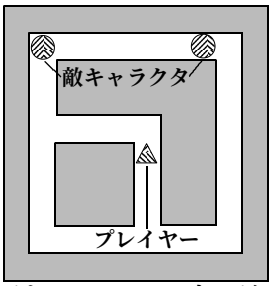
・プレイヤーが操作するキャラクタ
・敵キャラクター
・壁

の３種類で、フィールドが１０×１０の大きさのゲームを考えてみましょう。この場合、仮想画面として１０×１０のサイズの記録箇所を用意します。そして、画面上ではグラフィックによって表示されているキャラクタを数字に置き換え、そのキャラクタが存在する箇所に数値を記録します。
右の図のような仮想画面を見るとわかるように、

・プレイヤーキャラクタが存在する箇所に「１」
・敵キャラクタが存在する箇所に「２」
・壁には「３」
・何もない場所には「０」

が記録されるわけです。

このような仮想画面を使うことによるメリットはいくつかあります。まず、通路の判断が即座にできること。そして、キャラクター同士の衝突判断に使いやすいことなどです。



▲壁があるフィールド上のゲーム

3	3	3	3	3	3	3	3	3	3
3	2	0	0	0	0	0	2	0	3
3	0	3	3	3	3	3	3	0	3
3	0	3	3	3	3	3	3	0	3
3	0	0	0	0	0	3	3	0	3
3	0	3	3	3	1	3	3	0	3
3	0	3	3	3	0	3	3	0	3
3	0	3	3	3	0	3	3	0	3
3	0	0	0	0	0	0	0	0	3
3	3	3	3	3	3	3	3	3	3

▲左の状態を仮想画面に投影する

"２次元配列"で仮想画面を実現

仮想画面をプログラムで実現する場合は、"配列"を用います。プログラム中での仮想画面は、座標から存在キャラクタを引き出す、データベースのような働きをします。配列の添字を座標に対応させ、対応する添字の要素にキャラクターの番号を記録しておけば、仮想画面と同じ動作ができるのです。

配列の添字と座標の対応については、配列を"２次元配列"にすることによって、ｘ軸とｙ軸を直接対応付けることができます。

具体的には、２次元配列の１つ目の添字をｙ座標、２つ目の添字をｘ座標に対応させてやれば、

```
array[y][x]        ※arrayは10×10の２次元配列
```

で座標(x, y)の仮想画面にアクセスできるわけです。

たとえば、上の図の例で、座標(7, 1)の仮想画面に敵キャラクタの数値を記録させるには、

```
array[1][7] = 2;
```

とします。また、座標(5, 3)の仮想画面に、壁が存在するかどうかを調べるには、

```
if(array[3][5] == 3)
```

とすればいいことになります。

仮想画面のメリット①

このような仮想画面を利用すると、壁の判定、衝突検出などに関係する対応キャラクタの選択などの処理が、大幅に簡略化および高速化できます。

壁の判定については、キャラクタの進行方向の仮想画面を調べることによって、壁があるかどうかがすぐにわかります。壁であれば移動方向を変える処理を行えば、壁によって仕切られた通路をキャラクタが移動していくようになります。

移動方向を変えるときも、新しい移動先に壁があるかどうかを仮想画面によって調べ、移動できる方向を新しく設定してやればよいのです。

仮想画面のメリット②

仮想画面に記録する数値を、より複雑なものにすることによって、配列で管理されている複数のキャラクタの添字を直接求めることもできます。

前述の例では、プレイヤーキャラクタが「１」、敵キャラクタ「２」、壁は「３」というぐあいに数値を付けました。敵キャラクタが複数いる場合、仮想画面からの情報が「２」であったとすると、敵キャラクタが存在することはわかるものの、どの敵キャラクタかはわかりません。

複数敵キャラクタが存在する場合、それらは配列で管理されていることが多いはずです。このような場合、敵キャラクタの数値を変更し、「10」や「0xF0」などの計算しやすい数値に、敵キャラクタのデータを格納している配列の添字を加えたものを仮想画面に記録します。すると、仮想画面からのデータが敵キャラクタを示すものであった場合、たとえば13や0xF3とすると、「配列の４つ目に格納されている敵キャラクタが存在する」ということが即座にわかります。

§ 課題 §

以下のプログラムは、仮想画面を使用した通路移動アルゴリズムです。プレイヤーキャラクタが通路にそって移動できることを確認しましょう。

```
/*
========================================================================================================
                                        エイリアンゲーム
    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】
     AlienGame.cpp
                エイリアンゲーム

  【履歴】
     * Version    1.00       2003/08/31  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 <list>

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

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

/*******************************************************************************************************/
/*                                          定　　数                                                   */
/*******************************************************************************************************/
const DWORD   WIDTH  = 640;                         // 画面幅
const DWORD   HEIGHT = 480;                         // 画面高さ

const DWORD   BASE_X =   0;                         // マップ描画開始位置(ｘ座標)
const DWORD   BASE_Y =  32;                         // マップ描画開始位置(ｙ座標)
```

```
const DWORD   BLOCK  =  16;                          // ブロックのサイズ

const DWORD   MAX_X  = (WIDTH  - BASE_X) / BLOCK;    // マップのｘ最大数
const DWORD   MAX_Y  = (HEIGHT - BASE_Y) / BLOCK;    // マップのｙ最大数

/*****************************************************************************************************/
/*                                          グローバル変数                                           */
/*****************************************************************************************************/
static int         g_MAP[MAX_Y][MAX_X];              // 仮想画面
static PLAYER      g_Player;                         // プレイヤー

/*****************************************************************************************************/
/*                                          インライン関数                                           */
/*****************************************************************************************************/
// 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;
}

// スクリーン座標をマップ座標に変換
inline DWORD ToMapX(const DWORD x) { return (x - BASE_X) / BLOCK; }
inline DWORD ToMapY(const DWORD y) { return (y - BASE_Y) / BLOCK; }

// マップ座標をスクリーン座標に変換
inline DWORD ToScreenX(const DWORD x) { return x * BLOCK + BASE_X; }
inline DWORD ToScreenY(const DWORD y) { return y * BLOCK + BASE_Y; }

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

static void InitMap(int map[MAX_Y][MAX_X]);
static void CreateBG(const DDSRFC dds, const int map[MAX_Y][MAX_X]);

static void PlayerProc(PLAYER& player, const BYTE byKeyState[], int map[MAX_Y][MAX_X]);
static bool IsWall(RECTD& rect, const int map[MAX_Y][MAX_X]);

static void DrawAlien();
static void DrawPlayer(const HDC hDC, PLAYER& player);

/*****************************************************************************************************/
/*                                          エイリアンゲーム                                         */
/*****************************************************************************************************/
void Alien()
{
    // マップ初期化
    InitMap(g_MAP);

    // マップからプレイヤーの位置を探す
    int   x, y;
    for(y = 0; y < MAX_Y; y++) {
        for(x = 0; x < MAX_X; x++) {
            if(1 == g_MAP[y][x]) {
                // プレイヤー初期化
                g_Player.rect.left   = ToScreenX(x);
                g_Player.rect.top    = ToScreenY(y);
                g_Player.rect.right  = g_Player.rect.left + BLOCK;
                g_Player.rect.bottom = g_Player.rect.top  + BLOCK;

                // ループから抜ける
                x = MAX_X;
                y = MAX_Y;
            }   // if(MAP)
        }   // for(x)
    }   // for(y)

    // 背景構築
    CreateBG(DDS_OFFSCRN1, g_MAP);

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

```
/*****************************************************************************************************/
/*                                          マップ初期化                                             */
/*****************************************************************************************************/
void InitMap(int map[MAX_Y][MAX_X])
{
    // マップデータ(0..通路, 1..プレイヤー, 2..モンスター, 3..壁)
    LPCTSTR   map_data[] =
    {
        "33333333333333333333333333333333333333330",
        "32000000000000000000000000000000000000230",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30000000000000000000000000000000000000030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30000000000000000001000000000000000000030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30000000000000000000000000000000000000030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "30333330333330333330333330333330333333030",
        "32000000000000000000000000000000000000230",
        "33333333333333333333333333333333333333330",
        "00000000000000000000000000000000000000000"
    };

    // マップへ転送
    for(int y = 0; y < MAX_Y; y++)
        for (int x = 0; x < MAX_X; x++)
            map[y][x] = *(map_data[y] + x) - '0';
}

/*****************************************************************************************************/
/*                                          背景構築                                                 */
/*****************************************************************************************************/
void CreateBG(const DDSRFC dds, const int map[MAX_Y][MAX_X])
{
    DDCreateSurface(dds, WIDTH, HEIGHT, 0);
    DDColorFill(dds, NULL, RGB(0, 0, 0));

    HDC   hDC = DDGetDC(dds);

    // 文字描画
    SetTextColor(hDC, RGB(255, 255, 255));
    SetBkMode(hDC, TRANSPARENT);

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

    const COLORREF   WALL_COLOR = RGB(128, 128, 128);   // 壁の色
    // ペン生成
    HPEN   hPen        = CreatePen(PS_SOLID, 1, WALL_COLOR);
    HPEN   hDefPen     = (HPEN)SelectObject(hDC, hPen);
    // ブラシ生成
    HBRUSH   hBrush    = CreateSolidBrush(WALL_COLOR);
    HBRUSH   hDefBrush = (HBRUSH)SelectObject(hDC, hBrush);

    // 通路生成
    for(int y = 0; y < MAX_Y; y++) {
        for(int x = 0; x < MAX_X; x++) {
            if(3 == map[y][x]) {
```

```
                const int   WALL_X = ToScreenX(x);
                const int   WALL_Y = ToScreenY(y);
                Rectangle(hDC, WALL_X, WALL_Y, WALL_X + BLOCK, WALL_Y + BLOCK);
            }   // if(MAP)
        }   // for(x)
    }   // for(y)

    // ブラシ解放
    SelectObject(hDC, hDefBrush);
    DeleteObject(hBrush);
    // ペン解放
    SelectObject(hDC, hDefPen);
    DeleteObject(hPen);

    DDReleaseDC(dds);
}

/*****************************************************************************************************/
/*                                      エスケープゲームメイン処理                                   */
/*****************************************************************************************************/
void AlienMain()
{
    // キー入力
    BYTE   byKeyState[256];
    DIGetKeyboardState(byKeyState);

    PlayerProc(g_Player, byKeyState, g_MAP);    // プレイヤー処理

    DrawAlien();
    DDFlip();
    WaitFrame();
}

/*****************************************************************************************************/
/*                                          プレイヤー処理                                           */
/*****************************************************************************************************/
void PlayerProc(PLAYER& player, const BYTE byKeyState[], int map[MAX_Y][MAX_X])
{
    const double   ADD = 2.0;   // 移動量

    // キー状態設定
    const int   LR = ((byKeyState[DIK_RIGHT] & 0x80) >> 7) - ((byKeyState[DIK_LEFT] & 0x80) >> 7);
    const int   UD = ((byKeyState[DIK_DOWN ] & 0x80) >> 7) - ((byKeyState[DIK_UP  ] & 0x80) >> 7);

    // 左右移動
    if(0 != LR) {
        const DWORD   OLD_X = ToMapX(player.rect.left);
        OffsetRect(player.rect, ADD * LR, 0.0);
        if(IsWall(player.rect, map)) {
            const double   WIDTH  = player.rect.right  - player.rect.left;
            player.rect.left  = ToScreenX(OLD_X);
            player.rect.right = player.rect.left + WIDTH;
        }
    }

    // 上下移動
    if(0 != UD) {
        const DWORD   OLD_Y = ToMapY(player.rect.top);
        OffsetRect(player.rect, 0.0, ADD * UD);
        if(IsWall(player.rect, map)) {
            const double   HEIGHT = player.rect.bottom - player.rect.top;
            player.rect.top    = ToScreenY(OLD_Y);
            player.rect.bottom = player.rect.top + HEIGHT;
        }
    }
}

/*****************************************************************************************************/
/*                                      領域内に壁があるか調べる                                     */
/*****************************************************************************************************/
bool IsWall(RECTD& rect, const int map[MAX_Y][MAX_X])
{
    int   MAP_LEFT   = ToMapX(rect.left  );
    int   MAP_TOP    = ToMapY(rect.top   );
```

```
    int   MAP_RIGHT  = ToMapX(rect.right );
    int   MAP_BOTTOM = ToMapY(rect.bottom);

    if(ToScreenX(MAP_LEFT) != rect.left)
        MAP_RIGHT++;
    if(ToScreenY(MAP_TOP ) != rect.top )
        MAP_BOTTOM++;

    for(int y = MAP_TOP; y < MAP_BOTTOM; y++) {
        for(int x = MAP_LEFT; x < MAP_RIGHT; x++) {
            if(3 == map[y][x])
                return true;
        }
    }

    return false;
}

/*****************************************************************************************************/
/*                                          エイリアンゲーム描画                                     */
/*****************************************************************************************************/
void DrawAlien()
{
    // 背景描画
    DDBltFast(DDS_BACKBUF, 0, 0, DDS_OFFSCRN1, NULL, DDBLTFAST_WAIT);

    HDC   hDC = DDGetDC(DDS_BACKBUF);

    // スコア描画
    SetTextColor(hDC, RGB(255, 255, 255));
    SetBkMode(hDC, TRANSPARENT);

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

    // ライン(ペン)設定
    HPEN   hPen    = CreatePen(PS_SOLID, 1, RGB(255, 255, 255));
    HPEN   hDefPen = (HPEN)SelectObject(hDC, hPen);

    DrawPlayer(hDC, g_Player);      // プレイヤー描画

    // ライン(ペン)解放
    SelectObject(hDC, hDefPen);
    DeleteObject(hPen);

    DDReleaseDC(DDS_BACKBUF);
}

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