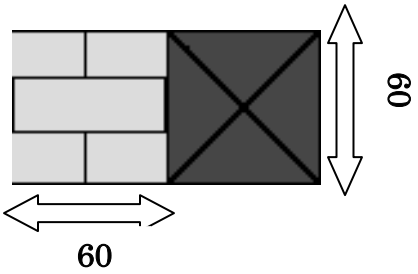
二次元配列のマップデータへの応用

アクションゲームにおける、ステージの構成は重要です。しかし、そのために使う画像をいちいち貼りつけたり、画像と実際の判定を一致させたりといった作業は面倒なことになります。これを二次元配列を利用することで、ブロックごとに管理できるようにすれば比較的簡単にステージが作成できます。

ここでは１画面の簡単なステージを作成します。まずは使用する画像を作成しましょう。



これを Block.bmp という名前で保存します。

ではプログラム作成を始めましょう。
まず Image を２つ、Timer を１つ置いて、ヘッダーで

```
        private:        // ユーザー宣言
    TRect BRect[2],FRect,Full;//左からブロックの範囲、
                              //画像の書き込み先、画面全体
```

と宣言しましょう。後で使います。

```
#include "Unit1.h"
#define BlockSize 60
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    //Formの基本設定です=========================================
    ClientWidth = 900;
    //これで60*60のブロックが横に15個入ります。
    ClientHeight = 600;
    //これで縦には10個入ります。
    Form1->AutoScroll = false;

    //作業台の設定です(CopyRectを利用するため用意します)==========
    Image1->Left = 0;
    Image1->Top = 0;
    Image1->Width = ClientWidth;
    Image1->Height = ClientHeight;
    Image1->Visible = false;

    //使用する画像の設定です=====================================
    Image2->Picture->LoadFromFile("Block.bmp");
    //作成した画像を読み込ませます。
    Image2->Visible = false;

//後で使うRectの設定=============================================
    Full = Rect(0,0,ClientWidth,ClientHeight);
    //Form全域
    for(int i = 0;i < 2;i++){
        BRect[i] = Rect(BlockSize*i,0,BlockSize*(i+1),BlockSize);
    //Block画像
    }
```

ここまでが基本的な設定です。ここから二次元配列を使っていきましょう。

MapSetUp という名前の void 関数を作成し、コンストラクタに

MapSetUp();

と書きます。関数内で以下のように二次元配列を宣言、初期化します。

```
void __fastcall TForm1::MapSetUp()
{
    //マップデータ配列の初期化
    int MapData[10][15] = {
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,0,1,0,0,1,0,0,0,0,1,1,0,0,1},
    {1,0,1,0,1,0,0,0,0,0,1,0,0,0,1},
    {1,0,1,1,0,0,1,1,0,0,1,1,0,0,1},
    {1,0,1,1,0,0,1,1,0,0,0,0,1,0,1},
    {1,0,1,0,1,0,0,0,0,1,0,0,1,0,1},
    {1,0,1,0,0,1,0,0,0,0,1,1,0,0,1},
    {1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    };
```

このように配列を宣言し、初期化するのはヘッダーでは不可能です。また宣言された変数は”{}”でくくられた範囲でのみ有効です。(この場合は MapSetUp 内でのみ有効)

この下にこの配列に合わせて画像を Image1 に描画するよう書き込みます。

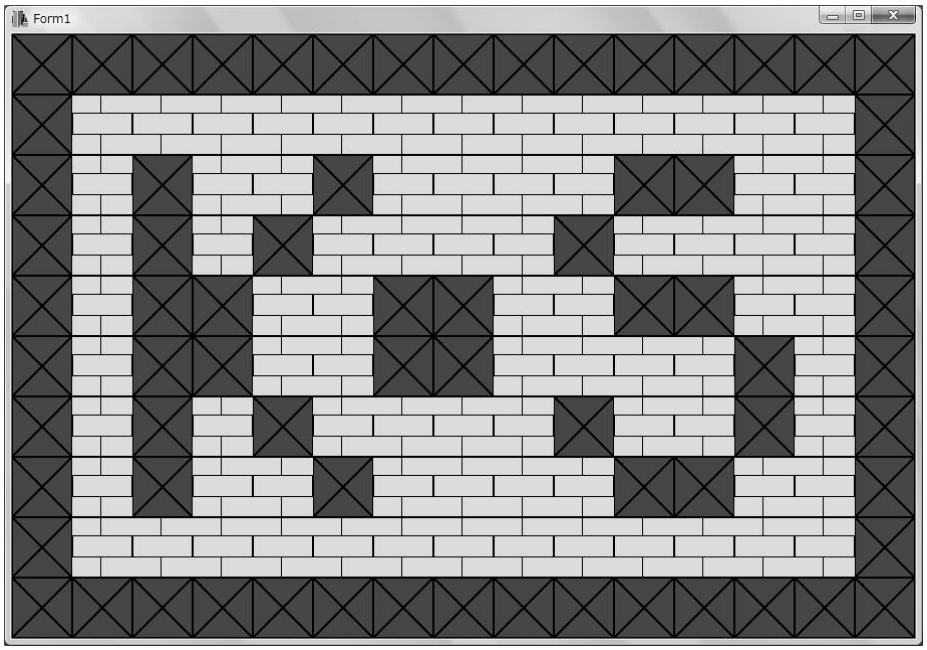
```
    for(int i = 0;i < 15;i++){
        for(int j = 0;j < 10;j++){
        //画像の描画先の決定
            FRect = Bounds
                (BlockSize*i,BlockSize*j,BlockSize,BlockSize);
        //MapDataの数値に合わせて2種類の画像を使い分ける
            Image1->Canvas->CopyRect
                (FRect,Image2->Canvas,BRect[MapData[j][i]]);
        }
    }
},
```

最後に Timer の中に Image1 の画像を Form に描画させるように書きこみます。

```
//---------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    Form1->Canvas->CopyRect(Full,Image1->Canvas,Full);
}
//---------------------------------------------------------
```

これで MapData が 1 の時はレンガ状のブロック、2 の時は×のブロックが表示されるようになっているはずです。

では実行してみましょう……



こんな感じになるはずです。この作り方では、一度作っておけば、後になってブロックの位置を変更したいときはMapDataを書き換えるだけで済みます。また、どういう形なのか、把握しやすいので作る時にも気が楽です。

これを応用していくことで、簡単なアクションゲームや、パズルゲームを作ることもできるので覚えておくと便利でしょう。