ESライブラリ＆＆
ゲームプログラミング

○　荷物勇者編 - 第１０回　地形データで移動制限　○

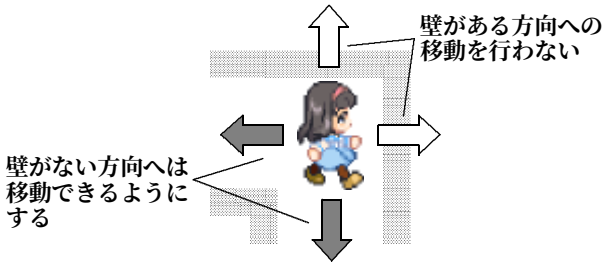
通路の移動

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ー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、パックマンやデビルワールドです。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ーは、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、キャラクター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、キャラクター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、壁とキャラクター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

仮想画面による通路の判断では、キャラクターの大きさを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ーを単純な数値や文字に置き換えて、そのキャラクター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ーが存在するか、即座に判断できるようになります。
サンプルとして次の図を見てください。このように、登場するキャラクターが

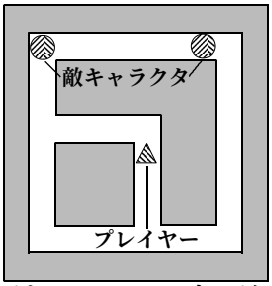
・プレイヤーが操作するキャラクター
・敵キャラクター
・壁

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、そのキャラクター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、画面座標から存在キャラクターを引き出す、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、

```
map_data[y][x]        ※map_data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ます。

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ーの数値を記録させるには、

```
map_data[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(map_data[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

メリット①

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ー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、キャラクター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ー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。

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ー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」という具合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、敵キャラクターの種類まではわかりません。

複数敵キャラクターが存在する場合、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、敵キャラクターの数値を変更し、「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ー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ーが存在する」ということが即座にわかります。

画面座標と仮想画面座標の変換

仮想画面を使用した場合、画面上の座標と仮想画面上の座標を相互に変換しなければならない場面が多々あります。このとき、仮想画面から画面上の座標に変換するには、

仮想画面の座標　×　キャラクターのサイズ　（＋オフセット）

という計算式で行うことができます。オフセットとは、画面座標の原点が(0, 0)でない場合に補正しなければならない値です。
逆に、画面上の座標を仮想画面上の座標(添字)にするには、以下のように逆の計算式となります。

（オフセット－）画面上の座標　÷　キャラクターのサイズ

課　　題

壁がある場合は移動できないようにしましょう。

ヒント１：今までは無条件に移動できましたが、条件を設定し、条件を満たしたときだけ移動できるようにします
ヒント２：移動できる条件は、移動先が「壁でないとき」です
ヒント３：プレイヤーの座標をMapDataの添字に変換します
ヒント４：変換した添字をもとに、移動先が壁かどうか調べます
ヒント５：壁でないときは移動させます
ヒント６：以下のようになります

```
// 左
if(KeyBuf.IsPressed(Keys_Left)) {
    // プレイヤーの座標をmapDataの添字に変換
    int   mx = (int)(※ここは各自考えましょう);
    int   my = (int)(※ここは各自考えましょう);

    // 左隣が壁かどうか調べる
    if(mapData[my][※ここは各自考えましょう] != '#') {
        // 左隣が壁でない場合は移動
        plyPos.x -= 32.0f;
    }
}
```

ヒント７：「プレイヤーの隣」は添字にある値を足したり引いたりします