ソフトゼミ B　第 7 回

ブロック判定

ソフトゼミ B 第 7 回はブロック判定についてです。前回のゼミで左右に移動したりジャンプをしたりという操作はできるようになりましたが、障害物も何もないところをただ移動するだけではアクションゲームとしてはつまらないですよね。そこで今回はステージにブロックを設置してプレイヤーとそれとの当たり判定を追加してみましょう。今回のゼミは最初にマップチップやファイル読み込みに関して簡単に解説をし、次にブロックとの当たり判定について穴埋めをしてもらうという形で進めていきます。

◆ マップチップとは

マップチップとは、アクションゲームや RPG で使用するマップを同じ大きさに細かく区切り、そこに 1 つずつステージのパーツを敷き詰めていくという技法です。アクションゲームで使用されるパーツには足場やトゲなど様々なものが考えられますが、今回はとりあえず足場となるブロックとクリアフラッグを適当に配置したマップのテキストデータを配布します。

◆ ファイル読み込み

マップチップを使用するためにはまずマップのテキストデータをプログラム内に読み込む必要があります。そこで登場するのが FILE 型のポインタ変数です。とりあえず今はファイルを読み込んだり、もしくはファイルに書き込んだりする際に使用する変数だと思っていてくれれば結構です。

```
//マップチップ読み込み
void Map(void){
	FILE *file;
	if(( file = fopen("map/map.txt","r")) == NULL ){
		OutputDebugString( "MapData Read Error¥n" ); //デバック画面に表示
		exit( EXIT_FAILURE );
	}
	for( int j=0 ; j < HEIGHT_SIZE ; j++ ){
		for( int i=0 ; i < WIDTH_SIZE ; i++ ){
```

```
                    fscanf(file, "%d,", &map[i][j]);
                }
        }
        fclose(file);
}
```

配布したプログラムには上の関数が含まれています。この関数ではまずファイルポインタ file を宣言し、次に fopen 関数によってファイルを開いています。(この if 文はもしファイルを正常に開けなかった場合にプログラムを終了させるというものです。)

そして、開いたファイルの中身を 2 次元配列 map[WIDTH_SIZE][HEIGHT_SIZE]に格納しています。これで読み込みは完了です。その後は fclose 関数によってファイルを閉じています。このように、ファイルを扱う際は終わった後にきちんと閉じることを忘れないでください。

◆ プログラムの穴埋め

上の様にファイルの内容を変数に格納したら、後はその変数を使ってプログラムを書いていきましょう。

```
        ////////////////////////////////////////////////////省略////////////////////////////////////////////////////

//壁判定 1（左右判定）
void judge1(int blx,int bly){
        if( 【穴埋め】 <= (player.x+HERO_SIZE) && 【穴埋め】 < (blx+BLOCK-4) &&
        【穴埋め】 < (player.y+HERO_SIZE) && 【穴埋め】 < (bly+BLOCK) ){
                //右方向
                player.kabejr = 1;
                if( map[blx/BLOCK][bly/BLOCK] == 4 ){
                        【穴埋め】;
                }
        }
        if( blx+4 <= 【穴埋め】 && player.x < 【穴埋め】&&
        bly < 【穴埋め】 && player.y < 【穴埋め】 ){
                //左方向
                player.kabejl = 1;
                if( map[blx/BLOCK][bly/BLOCK] == 4 ){
```

```
                    【穴埋め】;
                }
        }
}
```

上の関数は move 関数から呼び出します。blx,bly はそれぞれブロックの左上の x 座標、y 座標です。これが分かってしまえば、あとはシューティングゲームの時のように当たり判定の条件式を完成させて後処理を追加するだけです。

後処理の部分の if 文は、マップ上で 4 がクリアフラッグを表すので、触ったのが 4 番のブロックだったらゲームクリアーとするというものです。そのためにはどのような関数を呼び出せばよいか考えてみてください。

```
//壁判定 2（上下判定）
void judge2(int blx,int bly){
        if(【穴埋め】&& 【穴埋め】&& 【穴埋め】 && 【穴埋め】 ){
                if(player.vy >= 0){
                        player.jfly = 0;
                        player.y = 【穴埋め】;
                        player.vy = 0;
                }
                else{
                        player.vy = 1;
                        player.y = 【穴埋め】;
                }
                if( map[blx/BLOCK][bly/BLOCK] == 4 ){
                        【穴埋め】;
                }
        }
}
//////////////////////////////////////////////////省略//////////////////////////////////////////////////
void draw(void){
    ClearDrawScreen();              //画面クリア
        for(int i=0; i<BLOCK; i++){
            DrawGraph(i*BLOCK,0,img.haikei,TRUE);
        }                       //背景かく
        for(int j=0; j<HEIGHT_SIZE; j++){           //y 軸 10 分割単位で
```

```
            for(int i = 0 ; i < WINDOW_WIDTH/BLOCK; i++){
                //x 軸 32 分割いろいろで
                if(i < WIDTH_SIZE && i >= 0){
                    //横の部分より狭い範囲でマップ判断
                    switch(map[i][j]){
                        case 0:
                        case 3: 【穴埋め】;
                        case 1:
                          DrawGraph(i*BLOCK,j*BLOCK,img.yuka,TRUE);break;
                          //床
                        case 2:
                          DrawGraph(i*BLOCK,j*BLOCK,img.kabe,TRUE);break;
                          //壁（通り抜け不可）
                        case 4:
                          DrawGraph(i*BLOCK,j*BLOCK,img.goal,TRUE);break;
                          //ゴール旗（触れたらゴール）
                        case 5:
                        default:
                          DrawGraph(i*BLOCK,j*BLOCK,img.null,TRUE);break;
                          //ぬる
                    }
                }
                else if(j == 14){
                    DrawGraph(i*BLOCK,j*BLOCK,img.yuka,TRUE);
                }
            }
        }
////////////////////これ以降は前回までと変わらないので省略//////////////////////////
```

上下判定の後処理では、通り抜けできないブロックに対しての処理を考えてみてください。下からぶつかった場合は下へと跳ね返され、上からぶつかった場合はブロックの上に乗せる必要があります。ここではプレイヤーの y 軸方向の速度でどちらからぶつかったかを判定しています。

draw 関数内の穴埋めは、マップチップの番号が 0 と 3 の場合はこの switch 文では何もしないようにするための処理を書いてください。