ソフトゼミＢ　第７回　ブロック判定

前回の制作で床を左右移動して、飛び跳ねるゲームは出来ましたがアクションゲームと言うからには穴も壁も無く、平坦な道を進むだけのゲームは退屈でしょう。そこで今回はそんな単調なマップにブロックや穴を追加していこうと思います。そのためにマップチップの導入とブロック判定の処理を行います。今回はソースをかなり改造する上、複雑な所もありますので、頑張って付いてきてください。

マップチップとは

昔 RPG ツクールやスーパー正男などのゲームを作った人ならばわかりやすいと思いますが、広大なフィールドをより少ないパーツで表現するためにステージ全体を細かく区切り様々な種類のパーツを、タイルを並べるようにマップ全体に敷き詰める技法です。

今回はこちらが用意したマップチップを使いますが、自分でマップ作りたい！という方には専用のマップメーカーを用意したのでそちらを利用してください。

また、マップメーカーの解説については時間の都合上ゼミＢでは説明しませんので、仕組みが気になる方はソフト班長まで質問してください。

プログラムの追加

今回のマップチップは二次元配列map[50][15]に格納します。
横方向に50マスもあるのは次回のスクロール処理を見越してのことなので、今回の第７回では初めの20マスしか使用しません。

● ファイルの読み込み

初めに、テキストファイルに保存されているデータを読み込みます。

```
//第7回追加区画
//マップチップ読み込み
void load(void){
  FILE *file;
  if((file = fopen("map/map.txt","r"))==NULL){
    exit(EXIT_FAILURE);
  }
```

```
  for(int j=0;j<15;j++)
  {
    for(int i=0;i<SIDESIZE;i++)
    {
      fscanf(file,"%d,", 【穴埋め】);
    }
  }
  fclose(file);
  //読み込みここまで
}
//第7回追加区画ここまで
```

ここで初めてファイルポインタが登場します。ファイルポインタとはファイルの出入力に必要な情報を含んだ特別な型であるFILE型のポインタのことです。

初めにfopenでファイルをrモードで（モードについては後述）開き、中身がなければ強制終了させます。次に二重for文とfscanfを使い、読み取ったファイルを一つずつmap[i][j]に格納していきます。そして最後にfcloseでファイルポインタを閉じます。

- ## マップチップの描画

ここからは読み込んだマップチップを新たに描画する機能を追加します。

まずは前回作った床を破壊するので、以下の部分を削除してください

```
//背景の描画
//空の描画
for(i=0;i<32;i++){
  DrawGraph(i*32,0,img.sola,TRUE);
  DrawGraph(i*32,448,img.yuka,TRUE);←ここを削除
}
```

そしてその下に、以下を追加してください。

```
//第7回追加区画
for(int j=0;j<15;j++)
{
  for(int i=0;i<32;i++)
  {
    if(i<SIDESIZE&&i>=0)
    {
```

```
      switch(【穴埋め】){
        case 0:
        case 3: 【穴埋め】;
        case 1:
        case 5: DrawGraph(i*32,j*32,img.kabe,TRUE);break;
        case 2: DrawGraph(i*32,j*32,img.yuka,TRUE);break;
        case 4: DrawGraph(i*32,j*32,img.goal,TRUE);break;
        case 6: DrawGraph(i*32,j*32,img.toge,TRUE); break;
        default: DrawGraph(i*32,j*32,img.null,TRUE);break;
      }
    }
  }
}
//第7回追加区画ここまで
```

switch文は、ファイル読み込みで読み込んだマップデータについて各々判定しています。マップデータはi,jの順に読み込むので注意してください。データが0と3の時はなにも描画しません。switch文を抜け出すにはどうすればいいのかを考えて2つ目の穴埋めをしてください。

● ブロック判定

いよいよ読み込んだマップチップを元にブロック判定を行います。

判定そのものについては関数内で行いますので、まず以下の2つの関数を変数定義の下に挿入してください。

```
//壁判定１（横判定）
void judge1(int blx,int bly){
  if( player.x < (blx+BLOCK) && player.x > (blx-BLOCK-4) &&
            player.y > (bly-BLOCK) && player.y < (bly+BLOCK) )
  {
    player.kabejr=【穴埋め】;
  }
  if( player.x < (blx+BLOCK+4) && player.x > (blx-BLOCK) &&
           【穴埋め】 && 【穴埋め】 )
  {
    player.kabejl=【穴埋め】;
  }
}
```

```
//壁判定２（上下判定）
void judge2(int blx,int bly){
  if(【穴埋め】 && player.x > (blx-BLOCK) &&
             player.y > (bly-BLOCK) && player.y < (bly+BLOCK) )
  {
    if(player.vy>=0){
      player.jfly=【穴埋め】;
      player.y=(bly-BLOCK);
      player.vy=0;
    }
    else{
      player.vy=1;
      player.y=(bly+BLOCK);
    }
  }
}
```

player.kabejrはプレイヤーの右側に壁があれば1にして、それ以上右にいけないようにします。同様にplayer.kabejlは左に壁があれば1にして、それ以上左に行けないようにします。

壁判定2の1つ目のif文はプレイヤーが壁の内部に入り込んでいるかを判定します。その中のif文はプレイヤーが上から侵入したのか、下から侵入したのかをプレイヤーのy軸方向の速度を使って判定しています。下から落下してきた場合はブロックの上に乗せなければいけないので、それを考えてみてください。

次に、//ブロック判定１、と書いてあるところの後ろに以下を追加してください。

```
//ブロック判定１
player.kabejr=0;
player.kabejl=0;
for(j=0;j<15;j++)
{
  for(i=0;i<SIDESIZE;i++)
  {
    if(map[i][j]!=0){
      judge1(i*32,j*32);
    }
  }
}
```

次に、着地判定の

```
//着地判定
if(player.y>=416){
  player.jfly=0;//地面に着地したら着地判定をオンに
  player.vy=0;//V軸方向の速度を0に
  player.y=416;//落下時に座標にズレがあるので修正する
}
else{
  player.jfly=1;
}
```

の部分を削除し、以下を追加してください

```
//落下判定
if(player.y>=480){
  player.jfly=【穴埋め】;//地面に着地したら着地判定をオンに
   【穴埋め】;
}
else{
  player.jfly=【穴埋め】;
}
//ブロック判定２
for(j=0;j<15;j++)
{
  for(i=0;i<SIDESIZE;i++)
  {
    if(map[i][j]!=0){
      judge2(i*32,j*32);
    }
  }
}
```

2つ目の穴埋めは、穴に落ちた場合はゲームオーバー画面を読み出します。ゲームオーバー画面はdead();で読み出すことが出来ます。

ここまでがブロック判定となります。

最後に、WinMain関数の中のload();のコメントアウトを解除してください。

ファイル出入力に関して補足

ファイルの入出力に関して代表的な4つの関数を紹介します。
fscanfの他にgetc関数とかputc関数とかを使ったほうが楽な場合もありますけど
ここでは割愛します。

・fopen関数

```
fp=fopen(“test.txt”,” r”);
```

fopen関数は基本的に2つの引数があり、初めの引数”test.txt”はファイル名で、後ろの”r”がモードである。モードはrとwがあり、“r”は読み込み専用で書き込むことは出来ない。逆に”w”は書き込み専用であり、書き込むことが出来るが開く際にファイルの内容を消去して、先頭から書き込みをしてしまう。他にも"a"の追加書き込みモードなどがあるが、ここでは割愛する。

・fprint関数

```
fprintf(fp,” test:%d¥n”,num);
```

printfのファイル版で、基本的な動作はprintfと同じですが、ファイルポインタで指定したファイルに出力することが出来ます。この機能を使うには”w”モードでfopenする必要があります。

・fscanf関数

```
fscanf(fp, "%d", &num);
```

scanfのファイル版で、基本的な動作はscanfと同じです。
この機能を使うには”r”モードでfopenする必要があります。

・fclose関数

```
fclose(fp);
```

fopenで開いたファイルポインタで示されるファイルを閉じます。
使った後にはきちんと閉じましょう。