ソフトゼミB　第6回　プレイヤーの移動と重力

シューティングゲームの制作お疲れ様でした！今回からはアクションゲームのサンプルを制作していきます。アクションゲームとシューティングゲームの違いは、まず重力があるというのが上げられます。次に通行不能なブロックがあるというのも上げられます。

これから３回に渡って行われるアクションゲームでは「プレイヤーの移動と重力」「ブロック判定」「スクロール」の３つを学習していくつもりです。

準備

今回はアクションゲームの初回ということで、第１回のように新しくプロジェクトを作るところから始めましょう。ここまでは復習なのでゼミＢ第 1 回の資料を参考に、プロジェクトの新規作成から main.cpp の作成、DxLib の設定を行なってください。また付属の img フォルダと map フォルダを制作中のプロジェクトフォルダの中に入れてください。

プログラムの解説

設定を行ったら、付属のソースコードを main.cpp にコピペしてください。
しかしながらこの状態ではまだ動きませんので、以下の解説を参考にプログラム中の空欄を穴埋めしていってください。

- move 関数の穴埋め

```
void move(void){
	//重力加速
	if(player.jfly!=0&&player.vy<【穴埋め】){
		player.vy+=GRA;
	}
	//第 7 回追加区画
```

ここで重力加速の処理を行います。player.jfly は 0 の時に着地しているので、0 以外の時に重力加速させる処理をします。また右の player.vy<【穴埋め】は、プレイヤーの Y 軸方向の速度が最高速度以上の時は、これ以上重力加速をしないようにします。

続いて中身の player.vy+=GRA;では、ゲームループごとにプレイヤーの Y 軸方向の速度を加速します。つまり擬似的な重力加速を表現することができます。

```
//キーボードの入力状態の所得
GetHitKeyStateAll(keyState);
//左右キーで移動する
if(【穴埋め】)
{
        if(player.kabejl==0){
                player.x-=4;
        }
        player.dire=1;
}
else if(【穴埋め】)
{
        if(player.kabejr==0){
                player.x+=4;
        }
        player.dire=0;
}
```

次にキー入力と、左右方向の移動判定を行います。キー入力と左右移動はシューティングゲームで学習済みなので詳しい説明は割愛します。

```
//Xキーでジャンプ
if(【穴埋め】&& 【穴埋め】==0)
{
        player.jfly=1;
        player.vy=-15;
}
```

ジャンプ判定を行います。X キーが押されている＆player.jfly は 0 の時、つまり着地している時にジャンプするようにします。処理としては player.jfly を 1 にして空中にいる判定にした後で、プレイヤーの Y 軸速度を-15 にすることで、上方向へジャンプします。

```
//主人公の移動
player.x+=【穴埋め】;
player.y+=【穴埋め】;
//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;
}
```

着地判定を行います。これは上でやったジャンプ判定と逆のことをします。
判定条件は暫定的に player.y>=416 としてありますが、これはウィンドウの高さである 480 から床ブロックと、プレイヤーの大きさである 32px をそれぞれ引いたため、この数字にしてあります。

```
//端に来ると止まる
if(player.x<【穴埋め】){
        player.vx=0;
        player.x=0;
}
else if(player.x>602){
        player.vx=0;
        player.x=602;
}
～～～～～～～～～～～～～～～～～～～～～後略～～～～～～～～～～～～～～～～～～～～～
```

これはプレイヤーが画面外に出ることを阻止する処理です。内容としてはとても簡単で、画面外に出た場合は、画面端ギリギリにまで戻す。それだけの処理です。

- draw 関数の穴埋め

今回はアクションゲーム初回ですので、最低限の描画しかしません。
ですので単純な今回で描画の仕組みを理解するようにしてください。

```
void draw(void){
	ClearDrawScreen();//画面の内容を消去
	//背景の描画
	//空の描画
	for(int i=0;i<20;i++){
		DrawGraph(i*32,0,img.sola,TRUE);
		DrawGraph(i*32,448,img.yuka,TRUE);//第7回で削除
	}
	//第8回追加区画
～～～～～～～～～～～～～～～～～～～～～～～～中略～～～～～～～～～～～～～～～～～～～～～～～～
	//第8回追加区画ここまで
	//主人公の描画
	if(player.dire==0){
		DrawGraph(player.x,player.y,【穴埋め】,TRUE);
	}
	else{
		DrawGraph(player.x,player.y,【穴埋め】,TRUE);
	}
```

ここで主人公の描画を行います。player.dire は主人公の向きを表しています。
構造体の宣言部に書いてある各変数の概要に、player.dire が 0 の時はどちら向きであるかが書いてあるので、それを参考に穴埋めをしてみてください。

- title 関数の穴埋め

```
void title(void){
	ClearDrawScreen();//画面の内容を消去
	DrawGraph(0,0,  【穴埋め】,TRUE);
	ScreenFlip();
	while( 【穴埋め】 != 1 && keyState[ KEY_INPUT_ESCAPE ] != 1 ){
		GetHitKeyStateAll(keyState);
		if( ProcessMessage() == -1 )
		{break ; // エラーが発生したらループを抜ける}
	}
}
```

第一の穴埋めは、タイトル画面で表示される画像はなにか？というのを考えてみてください、二つ目の穴埋めは、X キーか Esc キーが押されていない場合のループですので、穴埋めの右を参考にしてもらえると簡単にわかると思います。