ソフトゼミ B 2015 第 6 回

プレイヤーの移動と重力

シューティングの制作お疲れ様でした。今回からアクションゲームのプログラミングに入っていきます。アクションでは新たに「重力」と「ブロック判定」という要素が入ってきます。今回はそのうち重力について解説していきます。

➢ 準備

何度も言うとおり今回からアクションに入るため、新しくプロジェクトを作りましょう。また、ソースファイルである main.cpp も作ります。ゼミ B 初回の資料にやり方は書いてあるので、もし忘れてしまった方はその手順に沿ってやってみてください。
設定が終わった方は早速今日配った「z.e.m.i._b_act_01」というフォルダ内の「zemi_b_06.txt」というテキストファイルに書いてあるソースコードを main.cpp にコピペしてください。ただし、いうまでもなくそのままでは動かないのでソースコード内の穴埋めを埋めてみてください。
また、穴埋めができた人はプロジェクトのフォルダに「img」、「map」のフォルダを入れて実際に動かしてみましょう。

➢ ソースコード

```
//(注意)今回使用するソースはあくまでアクションゲームの基本を理解するためのもので、
次回以降使うソースよりも内容を簡略化してあります。
//そのため次回以降の流用はできません。

#include "DxLib.h"
#include <stdio.h>

#define WINDOW_WIDTH 640     //ウィンドウの大きさ(縦)
#define WINDOW_HEIGHT 480    //ウィンドウの大きさ(横)
#define GRA 1                        //重力加速度
#define MAXSPEEDY 20         //自由落下の最高速度
#define CENTER 288                   //主人公の中央座標
#define HERO_SIZE 32         //主人公の大きさ
#define ATTACK_SIZE 32               //攻撃の範囲
```

```
#define WIDTH_SIZE 50		//スクロールの幅(横)
#define HEIGHT_SIZE 15			//スクロールの幅(縦)
#define BLOCK 32			//タイルの大きさ[px]

//画像関連の構造体
struct SImg
{
	int migi;
	int hidari;
	//int r_attack;
	//int l_attack;
	int kabe;
	int yuka;
	int goal;
	int r_enemy;
	int l_enemy;
	int gameover;
	int title;
	int clear;
	int haikei;
	int null;
};

//プレイヤー関連の構造体
struct SPlayer
{
	int x,y;		//主人公の X,Y 座標
	int vx,vy;			//主人公の X,Y 軸方向の速度
	int jfly;		//飛んでいるかどうかの判別　0:着地/1:それ以外
	int dire;		//主人公の向いている方向 0:右/1:左
	//int attack;			//攻撃中かどうかの判別		0:通常/1:攻撃中
	//int kabe_r;			//右方向に壁があるかどうかの判定
	//int kabe_l;			//左方向に壁があるかどうかの判定
};

//敵関連の構造体
```

```
struct SEnemy
{
	int x,y;			//敵の X,Y 座標
	int vx,vy;				//敵の X,Y 軸方向の速度
	int life;			//敵の体力
	int size_x,size_y; //敵の大きさ
	int count;				//敵の動く際のカウント
	int dire;			//敵の向いている方向	0:右/1:左
};

//構造体の宣言
struct SImgimg;
struct SPlayer player;
struct SEnemy enemy;
int map[WIDTH_SIZE][HEIGHT_SIZE];
char keyState[256];

//マップチップ読み込み
void Map(void){
	FILE *file;

	if((file = fopen("map/map_test.txt","r")) == NULL)
	{
		OutputDebugString("MapData Read Error¥n");	//もしマップデータが開くことが出来なかったらデバッグ画面に"Map Data Read Error"と表示
		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);
}

//画像読み込み
```

```
void image(void){

	img.migi = LoadGraph("img/migi.png");
	img.hidari = LoadGraph("img/hidari.png");
	//img.r_attack = LoadGraph("img/r_attack.png");
	//img.l_attack = LoadGraph("img/l_attack.png");
	img.kabe = LoadGraph("img/kabe.png");
	img.yuka = LoadGraph("img/yuka.png");
	img.goal = LoadGraph("img/goal.png");
	//img.r_enemy = LoadGraph("img/r_enemy.png");
	//img.l_enemy = LoadGraph("img/l_enemy.png");
	img.gameover = LoadGraph("img/gameover.png");
	img.title = LoadGraph("img/title.png");
	img.clear = LoadGraph("img/clear.png");
	img.haikei = LoadGraph("img/haikei.png");
	img.null = LoadGraph("img/null.png");

}

//変数の初期化
void init(void){

	player.x = 0;			//プレイヤーの初期 x 座標
	player.y = 416;			//プレイヤーの初期 y 座標
	player.vx = 0;
	player.vy = 0;
	player.jfly = 0;
	player.dire = 0;
	//player.attack = 0;
	//player.kabe_r = 0;
	//player.kabe_l = 0;

	//enemy.size_x = 32;
	//enemy.size_y = 32;
	//enemy.x = 70;
	//enemy.y= 480 - BLOCK - enemy.size_y;
```

```
	//enemy.vx = 0;
	//enemy.vy = 0;
	//enemy.count = 0;
	//enemy.life = 1;
	//enemy.dire = 0;

}

//タイトル画面の表示
void title(void){

	ClearDrawScreen();
	DrawGraph(0,0,img.title,TRUE);			//タイトル画面を描画
	ScreenFlip();

	while( keyState[KEY_INPUT_ESCAPE] !=1 &&keyState[KEY_INPUT_X] !=
1 ){			//エスケープかつ X が押されていないとき
		GetHitKeyStateAll(keyState);
		if(ProcessMessage() == -1)
		//エラーが発生したらループを抜ける
			break;
	}
}

////プレイヤーキャラの攻撃
//void attack(void){
//
//	GetHitKeyStateAll(keyState);
//	if(keyState[KEY_INPUT_/*穴埋め 1*/]){   //S キーが押されていたら
//		if(player.attack == 0)
//			player.attack = 1;
//	}
//	else
//		player.attack = 0;
//}
```

```
//プレイヤーキャラの移動
void move(void){

	player./*穴埋め 2*/ = 0;					//横移動リセット

	if(player.jfly != 0 &&player.vy< MAXSPEEDY)	//重力
		player.vy += 1;



	GetHitKeyStateAll(keyState);			//キーボードの状態を取得

	if(keyState[KEY_INPUT_LEFT]){			//プレイヤーの左向き操作
	//	if(player.attack == 0){
	//		if(player.kabe_l == 0 &&player.x> 0){
				player./*穴埋め 3*/ -= 4;	//プレイヤーを x 方向に移動
させる
	//		}
			player.dire = 1;
	//	}
	}

	if(keyState[KEY_INPUT_/*穴埋め 4*/]){		//プレイヤーの右向き操作
	//	if(player.attack == 0){
	//		if(player.kabe_r == 0 &&player.x + HERO_SIZE <= BLOCK *
WIDTH_SIZE){
				player.x += 4;
	//		}
	//	}
		player.dire = 0;
	}

	if(player.jfly == 0 &&keyState[KEY_INPUT_X]){		//プレイヤーのジ
ャンプ操作
		player.jfly = 1;				//空中にいる
		player./*穴埋め 5*/ -= 15;
		player.y += player./*穴埋め 5*/;
```

```
		}

		if(keyState[KEY_INPUT_R])				//デバッグ用ジャンプ
			player.vy -= 5;

		if(/*穴埋め 6*/){		//プレイヤーの y 座標が画面サイズ-ブロック一マスより大きいとき
			player.jfly = /*穴埋め 7*/;		//着地状態にする
			player.vy = 0;			//落下速度を 0 にする
			player.y = WINDOW_HEIGHT-BLOCK-HERO_SIZE;
		}
		else{
			player.jfly = 1;		//空中にいる
		}

		player.x += player.vx;				//移動(プレイヤーの座標に速度を加算)
		player.y += player.vy;

}



//描画の関数
void draw(void){

		//画面クリア
		ClearDrawScreen();

		//背景の描画
		for(int j = 0;j <BLOCK;j++)
			DrawGraph(j * BLOCK,0,img.haikei,TRUE);

		//マップチップからの描画
		for(int j = 0;j <HEIGHT_SIZE;j++){
			for(inti = (player.x / BLOCK) - 9;i < (player.x / BLOCK) + 12;i++){
```

```
                        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: break;
                                case 1:         DrawGraph(i*BLOCK - player.x +
CENTER,j * BLOCK,img.yuka,TRUE); break;     //床
                                case 2:         DrawGraph(i*BLOCK - player.x +
CENTER,j * BLOCK,img.kabe,TRUE); break;     //壁
                                case 3:         DrawGraph(i*BLOCK - player.x +
CENTER,j * BLOCK,img.goal,TRUE); break;     //ゴール旗(触れたらゴール)
                                default: DrawGraph(i*BLOCK - player.x +
CENTER,j * BLOCK,img.null,TRUE); break;     //null
                                }
                        }
                        else if(j == 14)
                                DrawGraph(i*BLOCK - player.x + CENTER,j *
BLOCK,img.yuka,TRUE);                                                   //最下段
は床を描画
                }
        }

        if(player.dire == 0)                    //主人公の描画
                DrawGraph(CENTER,player.y,img.migi,TRUE);           //右向き
        else
                DrawGraph(CENTER,player.y,img.hidari,TRUE);         //左向き



        //if(player.attack){                            //攻撃の描画
        //      if(player.dire == 0)
        //              DrawGraph(CENTER                                  +
ATTACK_SIZE,player.y,img.r_attack,TRUE);
        //      else
        //              DrawGraph(CENTER                                  +
ATTACK_SIZE,player.y,img.l_attack,TRUE);
        //}
```

```
	//if(enemy.life> 0){				//敵の描画
	//		if(enemy.dire == 0)			//向きの判別
	//			DrawGraph(enemy.x - player.x + CENTER,enemy.y,img.r_enemy,TRUE);
	//		else
	//			DrawGraph(enemy.x - player.x + CENTER,enemy.y,img.l_enemy,TRUE);
	//}

	//デバッグ用の変数描画
	int white = GetColor(255,255,255);
	DrawFormatString(450,20,white,"X :%4d",player.x);
	DrawFormatString(550,20,white,"Y :%4d",player.y);
	DrawFormatString(450,40,white,"VX:%4d",player.vx);
	DrawFormatString(550,40,white,"VY:%4d",player.vy);
	/*DrawFormatString(450,60,white,"右壁:%4d",player.kabe_r);
	DrawFormatString(550,60,white,"左壁:%4d",player.kabe_l);*/
	if(player.dire){
		DrawString(450,80,"左向き",white) ;
	}
	else{
		DrawString(450,80,"右向き",white) ;
	}
	if(player.jfly){
		DrawString(550,80,"空中",white) ;
	}
	else{
		DrawString(550,80,"着地",white) ;
	}
	/*if(player.attack){
		DrawString(450,100,"攻撃",white) ;
	}
	else{
		DrawString(450,100,"通常",white) ;
	}*/
```

```
	ScreenFlip();
}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
ipCmdLine, int nShowCmd)
{
	//ウィンドウのタイトルを変更
	SetMainWindowText("action");

	//ウィンドウモードに変更
	ChangeWindowMode(TRUE);

	//解像度とカラービット数を設定
	SetGraphMode(WINDOW_WIDTH, WINDOW_HEIGHT, 32);

	SetWindowSizeChangeEnableFlag(TRUE) ;

	//初期化に失敗したらやめる
	if(DxLib_Init()== -1){return -1;}

	////マップチップ読み込み
	Map();

	//画像のロード
	/*穴埋め 8*/;

	//変数の初期化
	init();

	SetDrawScreen(DX_SCREEN_BACK);

	//タイトル表示
	title();

	/************************** ゲームループ ****************************/
```

```
	while(ProcessMessage() == 0&&CheckHitKey(KEY_INPUT_ESCAPE)==0){

		//動き
		/*穴埋め 9*/;
		//enemymove();

		//攻撃
		//attack();

		//描画
		draw();

	}

	/************************ ゲームループおわり ************************/
	//DX ライブラリ使用の終了処理
	DxLib_End();

	//SOFT の終了
	return 0;

}
```

➢ 重力について

シューティングゲームでは存在しなかった(ゲームによっては存在するけど)要素として重力が出てきました。今回、プレイヤーが空中にいるか着地しているかの判定は player.jfly という変数が制御しています。jfly が 0 のとき、プレイヤーは地面にいて、jfly が 1 のときは空中にいます。その時、move 関数内の

```
if(player.jfly != 0 &&player.vy< MAXSPEEDY)	//重力
		player.vy += 1;
```

という部分で自由落下運動の処理が行われます。プレイヤーが空中にいるとき、プレイヤーは重力加速度として常に下方向に 1 ずつ移動します。g=9.8 がいいという方は double 型でどうぞ。ただし、そのまま移動し続けると条件を加えない限りずっと加

速し続け、画面の下の端からも出て行ってしまいます。そこで、vy が MAXSPEEDY 以上の時は加速しないようにし、プレイヤーが地面に着いたら jfly を 0 にする(=地上にいる状態にする)処理をします。

```
        if(/*穴埋め 6*/){              //プレイヤーの y 座標が画面サイズ-ブロック一マスより大きいとき
                player.jfly = /*穴埋め 7*/;              //着地状態にする
                player.vy = 0;                  //落下速度を 0 にする
                player.y = WINDOW_HEIGHT-BLOCK-HERO_SIZE;
        }
        else{
                player.jfly = 1;                //空中にいる
        }
```

少し難しいので穴埋め 6 のヒント:プレイヤーの y 座標として扱われる点は矩形の左上に位置しているので注意！一方で着地に影響する点はプレイヤーの下の端の点。ということは…？