ソフトゼミ B　第 2 回

画像処理と移動

◆ はじめに

今回のゼミ第 2 回ではシューティングの基礎である画像描画および自機や敵の動きを勉強していきましょう。また、全体の構成を把握できるようにゲームループ部分も作っていきます。

◆ 画像およびコードの準備

まず、ゲームを作るには準備が必要です。ですので作成に必要な画像とソースコードを配布します。

1. img ファイルとソースコードのテキスト main.txt を配布します。まず img ファイルに「background.png」「bullet.png」「enemy.png」「gameover.png」「item.png」「player.png」「title.png」の 7 つの画像が入っていることを確認してください。
2. 前回の第 1 回でやったように新しいプロジェクトを作成してください。プロジェクト名は shooting など自分にわかりやすいものにするとよいでしょう。
3. 全ての設定が終わったら配布した main.txt の内容をコピーして、main.cpp に貼り付けてください。

これで準備完了です。それではゲーム作成をしてみましょう。

◆ 穴埋め

C 言語を理解しただけではゲームを作成するのは難しいでしょう。そのためにゲーム作りにはゲーム的な考えも必要となってきます。そこで今回は穴埋め形式で理解を深めていきます。main.cpp では第 2 回で必要なところを抜粋しました。穴埋めの答えは前で説明しますが、意味が分からない場合は必ず近くのソフト班員に手を挙げて聞いてください。それでははじめましょう。

```
#include"DxLib.h"
#include<math.h>
#define WIDTH 640
#define HEIGHT 480
#define ENEMY 20
#define ITEM 30
#define BULLET 200
//自機の struct
struct S_player{
	int x,y; //座標
	int vx,vy; //座標方向の速度
	int hp; //hit point
	int score; //得点
	int size_x;//x 軸方向の大きさ
	int size_y;//y 軸方向の大きさ
};
//敵の struct
struct S_enemy{
	int 【穴埋め】,【穴埋め】;
	int 【穴埋め】,【穴埋め】;
	int hp;
	int t; //その敵が作られてからのフレーム数を格納するメンバ
	int 【穴埋め】;
	int 【穴埋め】;
};
//弾の struct
struct S_bullet{
	double 【穴埋め】,【穴埋め】;
	double 【穴埋め】,【穴埋め】;
	int hp; //hp が 0 なら画面上に存在していない、 1 なら存在している
	int atk; //弾の攻撃力
	int 【穴埋め】;
	int 【穴埋め】;
};
//背景の struct
struct S_background{
```

```
        int 【穴埋め】,【穴埋め】;
        int vy;
        int size_x;
        int size_y;
};
//画像の struct
struct S_image{
        int img_player; //自機の画像を格納
        int img_enemy; //敵の画像
        int img_bullet; //弾の画像
        int img_gameover; //ゲームオーバー画面
        int img_title; //タイトル画面
        int img_background; //背景
};
enum e_flag{ //列挙型 enum ここでは
        title, //title=0
        gameplay, //gameplay=1
        gameover, //gameover=2
}; //の定数を定義していると考える

//グローバル変数の宣言
【穴埋め】 player; //自機の struct をインスタンス化
【穴埋め】 enemy[ENEMY]; //敵の struct を
【穴埋め】 playerbullet[BULLET]; //弾の struct を自機の弾用に
【穴埋め】 enemybullet[ENEMY][BULLET]; //弾の struct を敵の弾用に
【穴埋め】 item[ITEM]; //アイテムの struct を
【穴埋め】 background; //背景の struct を
【穴埋め】 imglist; //画像の struct を
e_flag flag; //e_flag 型の変数(中身が title,gameplay,gameover の 3 つにしかならない変数)
char key[256]; //キーボードの入力状況を格納するための変数
int white=GetColor(255,255,255); //画面に表示する文字の色を格納

//初期化用関数
int initialize(){

        //player の初期化
```

```
player.hp=100;
player.score=0;
player.size_x=50;
player.size_y=50;
player.x=(WIDTH - player.size_x) / 2;
player.y=HEIGHT - player.size_y;
player.vx=10;
player.vy=10;

//playerbullet の初期化
for(int i=0;i<BULLET;i++){
        playerbullet[i].hp=0;
        playerbullet[i].vx=0;
        playerbullet[i].vy=-10;
        playerbullet[i].atk=1;
        playerbullet[i].size_x=10;
        playerbullet[i].size_y=10;
}

//enemy と enemybullet の初期化
for(int i=0;i<ENEMY;i++){
        enemy[i].hp=0;
        enemy[i].size_x=50;
        enemy[i].size_y=50;
        for(int j=0;j<BULLET;j++){
                enemybullet[i][j].hp=0;
                enemybullet[i][j].atk=1;
                enemybullet[i][j].size_x=10;
                enemybullet[i][j].size_y=10;
        }
}
//background の初期化
        background.x=【穴埋め】;
        background.y=【穴埋め】;
        background.size_x=1280;
        background.size_y=720;
```

```
                background.vy=10;
                return 1;
        }
        //画像のロードをする関数
void load(){

        imglist.img_player = LoadGraph("img/player.png"); //自機
        imglist.img_enemy = LoadGraph("img/enemy.png"); //敵
        imglist.img_bullet = LoadGraph("img/bullet.png"); //弾
        imglist.img_gameover = LoadGraph("img/gameover.png"); //ｹﾞｰﾑｵｰﾊﾞｰ画面
        imglist.img_title = LoadGraph("img/title.png"); //ﾀｲﾄﾙ画面
        imglist.img_background = LoadGraph("img/background.png"); //背景
}

//自機を動かす関数
void move_player(){

        //キーボードからの入力により自機の座標を動かす
        if(key[KEY_INPUT_【穴埋め】]) player.x += player.vx; //右
        if(key[KEY_INPUT_【穴埋め】]) player.x -= player.vx; //左
        if(key[KEY_INPUT_【穴埋め】]) player.y += player.vy; //下
        if(key[KEY_INPUT_【穴埋め】]) player.y -= player.vy; //上

        //自機が画面から出ないように制限
        if(player.x<0) player.x=0;
        if(player.x + player.size_x > WIDTH) player.x = WIDTH - player.size_x;
        if(player.y<0) player.y=0;
        if(player.y + player.size_y > HEIGHT) player.y = HEIGHT - player.size_y;
}

//敵を動かす関数
void move_enemy(){
        //すべての敵
        for(int i=0;i<【穴埋め】;i++){
                //hp0 でない、つまり存在しているてき
                if(【穴埋め】!=0){
```

```
                        //x,y 方向の速度を現座標に加える
                        enemy[i].x += enemy[i].vx;
                        enemy[i].y += enemy[i].vy;

                        //y 方向で画面からいなくなったら、消す
                        if(enemy[i].y>【穴埋め】){
                                【穴埋め】=0;
                        }
                }
        }
}

//背景を動かす関数
void move_background(){
        //x,y 方向の速度を現座標に加え,背景の大きさでループさせる
        background.y = (background.y + background.vy) % background.size_y;
}

//敵を作る関数
void create_enemy(){
        //0～30 の乱数で 0 だったら
        if(GetRand(30)==0){
                //すべての敵から
                for(int i=1;i<【穴埋め】;i++){
                        //存在していないものを探し、敵を作る
                        if(enemy[i].hp==0){
                                enemy[i].hp=3;
                                enemy[i].t=0; //敵が作られた瞬間を t=0 とする
                                enemy[i].x = WIDTH/4+GetRand(WIDTH/2);
                                        //敵の出現位置を
                                enemy[i].y = -enemy[i].size_y;
                                        //乱数で決める
                                enemy[i].y = -enemy[i].size_y;
                                if(enemy[i].x < WIDTH/2){ //出現位置により速度を変
える
                                        enemy[i].vx = GetRand(5);
```

```
                                        }else{
                                                enemy[i].vx = -GetRand(5);
                                        }
                                        enemy[i].vy=5+GetRand(5);
        //1 フレームで 1 体しか作りたくないので、1 体作ったら break で抜ける
                                        break;
                                }
                        }
                }
}

//数値的な更新をする関数
void update(){
        //キーボード入力状況を取得
                GetHitKeyStateAll(key);

        //flag の中に何があるかで処理を変える
        switch(flag){
                case title: //title のとき
                        //Enter が入力されたら
                        if(key[KEY_INPUT_RETURN]){
                                flag=gameplay; //flag の中身を gameplay に変更
                                initialize(); //初期化関数を呼び出す
                        }
                        break;

                case gameplay: //gameplay のとき
                        //移動の関数
                        【穴埋め】;        //自機を動かす関数
                        【穴埋め】;       //敵を動かす関数
                        【穴埋め】;        //背景を動かす関数

                        //いろいろと作る関数
                        【穴埋め】;        //敵を作る関数

                        //自機の hp が 0 になったら
```

```
                        if(【穴埋め】==0){
                                flag=gameover; //flag の中身を gameover に変える
                        }
                        break;

                        case gameover: //gameover のとき
                        //X が入力されたら
                        if(key[KEY_INPUT_【穴埋め】]) flag=title;
                        //flag の中身を title に変える
                        break;
        }
}
//描画を行う関数
void draw(){
        //画面に描画されているものを消す
        ClearDrawScreen();

        //flag の中に何があるかで処理を変える
        switch(flag){
                case title: //title のとき
                        DrawGraph(0,0,imglist.img_title,FALSE); //title 画像の表示
                        break;

        case gameplay: //gameplay のとき
                //背景の描画、一番奥なので一番最初に描画
                DrawGraph(background.x,background.y,
                imglist.img_background,FALSE);
                DrawGraph(background.x,background.y -
                background.size_y,imglist.img_background,FALSE);
                //hp と score の描画
                DrawFormatString(540,50,white,"HP %d",player.hp);
                DrawFormatString(540,70,white,"Score %d",player.score);
                //自機の描画
                DrawGraph(player.x,player.y,imglist.img_player,FALSE);
                //自機の弾の描画
                for(int i=0;i<【穴埋め】;i++){
```

```
                        //存在しているものについて
                        if(playerbullet[i].hp){
                                //描画する
                                DrawGraph( playerbullet[i].x,
                                playerbullet[i].y,imglist.img_bullet,FALSE);
                        }
                }
                //敵と敵の弾の描画
                for(int i=0;i<【穴埋め】;i++){
                        //存在しているものについて
                        if(enemy[i].hp!=0){
                                //描画する
                                DrawGraph(enemy[i].x,enemy[i].y,
                                imglist.img_enemy,FALSE);
                        }
                        for(int j=0;j<BULLET;j++){
                                //存在しているものについて
                                if(enemybullet[i][j].hp){
                                        //描画する
                                        DrawGraph((int)enemybullet[i][j].x,
                                  (int)enemybullet[i][j].y,imglist.img_bullet,FALSE);
                                }
                        }
                }
        case gameover: //gameover のとき
                DrawGraph(0,0,imglist.img_gameover,FALSE); //gameover 画面の描画
                break;
        }
        //裏画面描画したものを表画面に反映
        ScreenFlip();
}
//main 関数
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nShowCmd ){
        //Dx ライブラリの初期化,失敗したら終了
        if(DxLib_Init()!=0) return 0;
```

```
        //表示をウィンドウモードに設定
        ChangeWindowMode(TRUE);
        //ウィンドウの大きさ、カラービット数を設定
        SetGraphMode( WIDTH, HEIGHT, 32 );
        //裏画面描画を設定
        SetDrawScreen( DX_SCREEN_BACK );
        //画像のロード
        【穴埋め】;        //画像のロードを行う関数
        //flag の初期化をここで行う
        flag=title;
//////////////////////ゲームループ開始//////////////////////
        while(ProcessMessage()==0){
        //Escape が入力されたら while を抜ける
        if(key[KEY_INPUT_【穴埋め】]) break;
        //数値的な処理をしてから
        update();
        //描画を行う
        draw();
}
//////////////////////ゲームループ終了//////////////////////
DxLib_End();
        return 0;
}
                                        ~おしまい~
```

◆ ゲームループについて

ゲームループは「動作→描画→画面の反映」を繰り返して行うものです。このゲームでの動作は実は、1 秒間に 60 回行われており、これを 60FPS と言います。この 60FPS より低い FPS のゲーム処理は軽いが動きが重くなります。逆に 60FPS より高い FPS の場合だとゲーム処理は重いが動きは滑らかになります。今回のゲーム制作では細かい動きを要求していないので 60FPS でゲームを制作しています。

詳しいことを知りたい人は「新・ゲームプログラミングの館　3.14 章」http://dixq.net/g/03_14.html を読んでみてください。

◆ #define について

このシューティングゲーム制作で、「#define ○○　100」 などの宣言を最初にしましたが、これに何の意味があるのか疑問に感じる方もいると思います。また、for(i=0;i<10;i++)と書くよりも for(i=0;i<ENEMY;i++)と書くほうが面倒だなと思う人もいると思います。しかし、数字だけでは何をしているのか判断がつかないことが多々あります。つまり逆説的に考えると、define した文字を置けばこの場面で何をしているのか分かりやすくすることができます。プログラムを見直す時にも分かりやすいですしつまらないミスも減ります。また、define は主に「マクロ」とも呼ばれるので特に情報科学科の人は覚えておきましょう。

◆ コメントアウトの重要性について

シューティングゲームにもありましたが、「//自機の画像」などのものがあったと思います。これはコメントアウトと言いゲーム制作では非常に重要なものとなります。ゲームを作るのには膨大なコードが必要となるので頭を整理するのが大変になってきます。そこでコメントアウトをしておけば、後で見たときにすぐに分かるようになっているので作業の効率化を図ることが出来ます。本番のゲーム制作では、このコメントアウトを十分に使って、他人が見ても分かりやすいようなコードを書きましょう。

◆ math.h について

math.h とはヘッダーファイルというものの 1 種です。主に三角関数や平方根などといった難しいプログラミングを要するものを、math.h に追随する関数を使えば一気にやってくれるという万能戦士です。ゲームでもぜひ頻繁に使ってみてください。