ソフトゼミ B　第 2 回　画像処理と移動

■ はじめに

ゼミ B 第 2 回はゲーム制作の基本となる文字や画像の描画、および自機や敵の移動について勉強しましょう。またゲームの土台となる**ゲームループ**を作り、ちゃんとしたゲームの流れを作ってみましょう。

■　画像ファイルの読み込み準備

まずはゼミＢのシューティングで使うソースコード「main.txt」と画像データが入ったフォルダ「img」を配布します。画像フォルダの中に「burret.png」「enemy.png」「gameover.png」「haikei.png」「item.png」「player.png」「title.png」「tokuten.png」の 8 つの画像データが入っていることを確認してください。次に Visual C++で、第 1 回でやったとおりに新しいプロジェクトを作成してください。この時にプロジェクト名は「Shooting_Sample」などとわかりやすい名前にしておいてください。プロパティページでの設定まで終わらせたら、今度は配布したほうの「main.txt」を開き、すべて選択→コピーした後にプロジェクトの方の「main.cpp」にペーストしてください。

これで準備完了、ここからはプロジェクト穴埋めをします。

■　プログラムの穴埋め

配布した「main.txt」をそのまま張り付けても穴埋め箇所があるのでまだ動きません。

そこでここからは穴埋めを埋めていきます。以下のプログラムは配布した main.cpp に書かれているデータのうち、第二回で必要なところを抜粋して掲載しました。穴埋め部分は壇上で解説していきますのでしっかりと聞いて理解したうえで埋めていくようにしましょう。また聞き逃した部分がありましたら、近くにいるソフト班員までお尋ねください。

```
#include "Dxlib.h"
#include <math.h>
#define ENEMY_BURRET 256 //敵の弾の最大数
#define PLAYER_BURRET 256 //プレイヤーの弾の最大数
#define ENEMY 10 //敵の最大数
#define HEIGHT  480     //縦幅の大きさ
#define WIDTH 640 //横幅の大きさ
//////経過時間
int time;
//////得点
int tokuten;
//////キーボード状態取得用
char keystate[256];
/*構造体の宣言*/
//自機のストラクト
struct Player{
        double x,y;       //座標
        double vx,vy;  //速度
        int life;    //体力
        double size_x,size_y;  //サイズ
};
//敵のストラクト
struct Enemy{
        double  ,  ;       //座標
        double     ,     ;  //速度
        int          ;       //体力
        int          ,          ; //サイズ
};
//背景のストラクト(都合上アイテムで宣言)
struct Item{
        double  ,  ;      //座標
        double     ,     ;  //速度
        double          ,          ; //サイズ
};
```

```
//画像のストラクト
struct Img{
        int          ;  //自機
        int          ;  //敵 10 匹
        int          ;  //ゲームオーバー画面
        int          ;     //タイトル画面
        int          ;     //背景
};

//Player 型の player
struct Player player;

//Enemy 型の enemy
struct          [          ];

//Img 型の img
struct Img imglist;

//背景型の背景
struct Item haikei;

//初期化用関数
int init(void){
        int i;
        // DX ライブラリの初期化に失敗すると、即終了
        if( DxLib_Init() == -1 ){ return -1; }

//player の初期化
        player.size_x=50;
        player.size_y=50;
        player.x=WIDTH/2;
        player.y=HEIGHT-100;
        player.life=500;
        player.vx=10;
        player.vy=10;
```

```
//敵の初期化
        for(i=0;i<        ;i++){
                enemy[i].life=0;
                enemy[i].size_x=50;
                enemy[i].size_y=50;
                enemy[i].vx=0;
                enemy[i].vy=10;
                enemy[i].x=0;
                enemy[i].y=0;
        }

//背景の初期化
        haikei.size_x=        ;
        haikei.size_y=         ;
        haikei.x=         ;
        haikei.y=         ;
        haikei.vx=    ;
        haikei.vy=     ;


//時間の初期化
        time=0;
//得点の初期化
        tokuten=0;

        return(1);
}

//時を進める
void time_progress(void){
        time++;
        time=time%1000;
}
//敵を生み出す
void create_enemy(void){
        int i;
```

```
	//一定フレームごとに
	if(time%10 == 0){
		//全ての敵に関して
		for(i = 0 ; i < [        ]; i++){
			//enemy[i]の命があったらスキップ
			if([                ]){continue;}
			//enemy[i]の命を生み出す
			enemy[i].life = 5;
		//敵の位置を乱数で決める
			enemy[i].x = rand() % (WIDTH - (int)enemy[i].size_x);

		//敵の動きをランダムに
			if(enemy[i].x > WIDTH/2){
				if(enemy[i].vx < 10){
					enemy[i].vx = 0;
				}[      ]{
					enemy[i].vx -= 2.5;
				}
			}[      ]{
				if(enemy[i].vx > 10){
					enemy[i].vx = 0;
				}[      ]{
					enemy[i].vx += 2.5;
				}
			}
			enemy[i].y = 0;
		//これがないと１ケ所の座標が使用中でないものすべてに代入される
			break;
		}
	}
}

//敵を動かす
void move_enemy(void){
	int i;
	for(i=0;i<[        ];i++){
```

```
                //敵の命がある時
                if(            ){
                        enemy[i].x+=enemy[i].vx;
                        enemy[i].y+=enemy[i].vy;
                }
                //今は下にしか敵が進まないから
                if(enemy[i].y>HEIGHT+enemy[i].size_y)
                {            }
        }                       //画面外にでたらコロス
}
//player を動かす
void move_player(void){

        GetHitKeyStateAll(keystate);   //キーボードの状態を取得
        //キーボード入力で動かす
        if(keystate[KEY_INPUT_LEFT]){
                player.x-=player.vx;
                }
        if(keystate[KEY_INPUT_RIGHT]){
                player.x+=player.vx;
                }
        if(keystate[KEY_INPUT_UP]){
                player.y-=player.vy;
                }
        if(keystate[KEY_INPUT_DOWN]){
                player.y+=player.vy;
                }
        //画面外に行かないように移動制限
        if(player.x>      -player.size_x){
                player.x=      -player.size_x;
        }
        if(player.y>       -player.size_y){
        player.y=       -player.size_y;
        }
        if(player.x<0){
        player.x=0;
```

```
        }
        if(player.y<0){
        player.y=0;
        }
}
//画像のロード
void load(void){
        int i;
        imglist.player= LoadGraph("img/        .png");//自機
        for(i=0;i<        ;i++){
                imglist.enemy[i]= LoadGraph("img/        .png");//敵
        }
        imglist.gameover=LoadGraph("img/        .png");//gameover 画面
        imglist.title=LoadGraph("img/        .png");//タイトル画面
        imglist.haikei=LoadGraph("img/        .png");//背景
}
//背景を移動
void move_haikei(void){
                haikei.y+=haikei.vy;
                if(haikei.y>=        )//背景のループ
                        haikei.y=0;
}

//描画
void draw(){
        int i;
//背景表示　背景だから一番最初に描画、ループさせるため２重に宣言
        DrawGraph((int)haikei.x,(int)haikei.y,imglist.        ,TRUE);
        DrawGraph((int)haikei.x,(int)haikei.y-HEIGHT,imglist.        ,TRUE);

//時間表示
        DrawString(WIDTH-150,50,"TIME:",GetColor(255,255,255));
        DrawFormatString(WIDTH-75,50,GetColor(255,255,255),"%6d",time);
//体力表示
        DrawString(WIDTH-150,70,"LIFE:",GetColor(255,255,255));
        DrawFormatString(WIDTH-75,70,GetColor(255,255,255),"%6d",player.life);
```

```
//得点表示
        DrawString(WIDTH-150,90,"SCORE:",GetColor(255,255,255));
        DrawFormatString(WIDTH-75,90,GetColor(255,255,255),"%6d",tokuten);
//自機を描画
        DrawGraph((int)player.x,(int)player.y,imglist.        ,TRUE);
//敵を描画
        for(i=0;i<ENEMY;i++){
                //敵の命がある時描画
                if(enemy[i].life>0){
               DrawGraph((int)enemy[i].x,(int)enemy[i].y,imglist.        ,TRUE);
                }
        }
        //player の体力が 0 になったらゲームオーバー
        if(             ){
                ClearDrawScreen();//画面に描かれたものを消去する
                DrawGraph(0,0,imglist.          ,TRUE);
        }
        ScreenFlip();//表画面に反映
}

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR
lpCmdLine, int nShowCmd ){
        ChangeWindowMode( TRUE ) ;//  ウィンドウモードにする
        SetGraphMode( WIDTH, HEIGHT, 32 );//  解像度とカラービット数を設定
        if(SetDrawScreen(DX_SCREEN_BACK)!=0) return -1; //  裏画面に描画
        init();  //初期化関数呼び出し
        load();  //画像をロードする関数

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

        while(ProcessMessage()==0){
                GetHitKeyStateAll(keystate);   //キーの取得

                ClearDrawScreen();  //画面に描かれたものを消去する

        //esc がオサレているなら抜ける
```

```
            if(keystate[KEY_INPUT_ESCAPE]){break;}
            DrawGraph(0,0,imglist.title,TRUE);
        //表画面に反映
            ScreenFlip();

            if(keystate[KEY_INPUT_RETURN]){

                [      ];          //初期化用関数呼び出し
                while(ProcessMessage()==0){

                    GetHitKeyStateAll(keystate); //キーの取得

                    ClearDrawScreen();//画面に描かれたものを消去
                //esc が押されているなら抜ける
                    if(keystate[KEY_INPUT_ESCAPE]){break;}

                    [          ]; //時を動かす関数
                    [          ]; //player を動かす関数
                    [          ];          //敵を生み出す関数
                    [          ]; //敵を動かす関数
                    [          ]; //背景動かす関数
                    [      ];              //描画関数

                    //player の体力が 0 になったらゲームオーバー
                    if(player.life<=0){
                        //title に戻る
                        if(keystate[KEY_INPUT_X]){break;}
                    }
                }
            }
        }
         /*******************ゲームループおわり**********************/
        [          ]; //DX ライブラリを終了する
        return 0;
}
```

(おしまい！)

■ ゲームループとは

ゲームループは「プレイヤーが動作する」→「動作を反映」→「画面に情報を反映する」→・・・の繰り返しを行うものです。

DrawGraph 関数や DrawString 関数、ScreenFlip 関数などを使い描画します。また、この描画の回数の単位をフレームといい、1 秒間に 60 回描画されることを 60FPS(Frame Per Second)と言ったりします。60FPS のゲームといった場合はゲームループを 1 秒間に 60 周しているということです。ゼミ B のゲーム制作では基本的に FPS は制御しません。したがってハードウェア依存になりますが、厳密な描写管理が必要なゲームではないので問題はありません。

もしそれじゃ困るという人や FPS をしっかり学びたいという人は、やや高度ですが「龍神録プログラミングの館（43 章)」：http://dixq.net/rp/43.html などを読んでみるとよいかもしれません。

■ #define の必要性

プログラムの頭に「#define WIDTH 640」のような命令が 3 つあります。

「define」は定義するという意味の英単語で、上の例では「WIDTH(横幅)」を 640 という長さで定義しています。こうしておくと、プログラム中すべての「WIDTH」が 640 に自動的に置き換わりコンパイルが行われます。

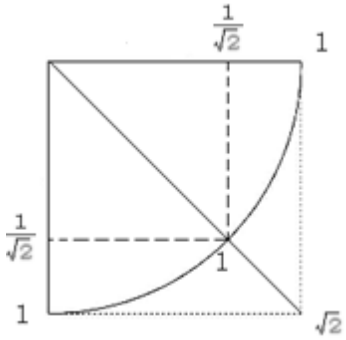
プログラム内で何度も使用する定数(ウインドウの幅や弾の数、敵の上限など)を置換しておけば、最初の#define を書き換えるだけで何度でも容易に変更ができ、バグを減らすことにも繋がっていきます。

また#define のことをマクロと呼ぶこともあることも覚えておきましょう。

次は自機、敵の移動について解説していきます。

■ 自機の移動

自機の移動についてですが、自機を8方向に動かせるようにしていきます。 ここで斜め移動について少し考えてみてください。 例えばｘ軸方向とy軸方向の移動速度が１のとき斜め移動を行うと、 √2の移動速度で進んでしまいます。斜め移動の移動速度も１になるようにしたい場合、斜め移動のときの場合分けをしてｘとｙの移動速度を√2で割ることによって、斜め移動の速度を１にできます。



■ 敵の移動

敵の動き方は乱数での初期座標の指定と場合分けで複雑に見せることができます。基本は下方向に一定の速さで移動させます。また、敵が画面外に出たらその敵に関する処理を行わせないようにしていきます。 すべての敵の処理を継続させると、重くなってしまうからです。

■ math.h（もしくはcmath)のインクルード

乱数を作る標準ライブラリ関数「rand」を呼び出すために、ヘッダファイル「math.h」(「cmath」でもOK)をインクルードしています。
プログラムの頭にある#include<math.h>がそれに当たります。また√2 を使うときにも標準ライブラリ関数「sqrt」を math.h で呼び出す必要があります。三角関数などの簡単な数式でも呼び出せるので、ゲーム制作の際は積極的に使っていきましょう。