ソフトゼミ B 2015 第 2 回

画像処理と移動

1.はじめに

ソフトゼミ B 第二回では画像の描画と自機及び敵機の動きについて学びます。また、全体の構成を把握するためのループ部分も作成します。シューテイングゲームの基礎となるところなので頑張っていきましょう。

2. 画像及びソースコードの準備

ゲーム作りのために必要な画像及びソースコードを配布します。

1. (define.h)(struct.h)(global.h)(initialize.cpp)(move.cpp)(create.cpp)(update.cpp)(draw.cpp)(main.cpp)と img ファイル内に 14 個の画像が存在するか確認してください。
2. プロジェクトを新しく作成してください。なるべく分かりやすいプロジェクト名がいいでしょう。

以上で準備完了です。

3. 穴埋め

ソフトゼミ A で C 言語を学んできましたが、それだけではゲーム制作に必要な考え方などが足りないため今回は穴埋め方式で理解を深めていきます。穴埋めの回答は黒板に書いていきますが、意味が分からない箇所などがあれば遠慮なくソフト班員の先輩に質問してください。

```
#include"DxLib.h"
#include<math.h>
#define WIDTH 800
#define HEIGHT 600
#define ENEMY 50
#define ITEM 30
#define BULLET 200

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 size_x;
	int size_y;
};

//弾の struct
struct S_bullet
{
	double x,y;
	double vx,vy;
	int【穴埋め】;		//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】;
};


//アイテムの struct
struct S_item
{
	int【穴埋め】,【穴埋め】;
	int【穴埋め】,【穴埋め】;
	int【穴埋め】;		//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【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_item;        //アイテム
        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


【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
【穴埋め】 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 black = GetColor(1,0,0);

void load(){ //画像読み込み

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_item = LoadGraph("img/item.png");              //アイテム
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

}

int initialize(){

        //player の初期化
        player.hp=200;
        player.score=0;
        player.nokori=200;
        player.size_x=50;
        player.size_y=50;
        player.x=(WIDTH - player.size_x) / 2;
        player.y=HEIGHT - player.size_y;
        player.vx=7;
        player.vy=7;
```

```
//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=10;
        playerbullet[i].size_x=10;
        playerbullet[i].size_y=10;
}

//enemy と enemybullet の初期化
for(int i=0;i<【穴埋め】;i++){
        enemy[i].hp=0;
        enemy[i].size_x=50;
        enemy[i].size_y=50;

        for(int j=0;j<【穴埋め】;j++){
                enemybullet[i][j].hp=0;
                enemybullet[i][j].atk=5;
                enemybullet[i][j].size_x=10;
                enemybullet[i][j].size_y=10;
        }
}

//item の初期化
for(int i=0;i<【穴埋め】;i++){
        item[i].hp=0;
        item[i].size_x=40;
        item[i].size_y=40;
}
//background の初期化
background.x=【穴埋め】;
background.y=【穴埋め】;
background.size_x=800;
background.size_y=600;
```

```
	background.vy=10;

	return 1;
}


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<15) player.x=15;
	if(player.x + player.size_x> WIDTH-15) player.x = WIDTH - player.size_x-15;
	if(player.y<15) player.y=15;
	if(player.y + player.size_y> HEIGHT-15) player.y = HEIGHT - player.size_y-15;
}




//敵を動かす関数
void move_enemy()
{
	//すべての敵(enemy[0]～enemy[ENEMY-1(=49)])
	for(int i=0;i<【穴埋め】;i++)
	{
		//HP が 0 ではない、つまり存在しているてき
```

```
                if(enemy[i].【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】)

{
                                enemy[i].【穴埋め】=0;
                        }
                }
        }
}
//自機の弾を動かす関数
/*void move_playerbullet()
{
        //すべての弾(playerbullet[0]～playerbullet[BULLET-1(=199)])
        for(int i=0;i<【穴埋め】;i++)
        {
                //hp!=0、すなわち存在していれば
                if(playerbullet[i].hp==1)
                {
                        //x,y 方向の速度を現座標に加える
                        playerbullet[i].【穴埋め】 += playerbullet[i].【穴埋め】;
                        playerbullet[i].【穴埋め】 += playerbullet[i].【穴埋め】;

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

```
                if(playerbullet[i].y < -playerbullet[i].size_y)
                {
                        playerbullet[i].【穴埋め】=0;
                }
                if(playerbullet[i].x < -playerbullet[i].size_x)
                {
                        playerbullet[i].【穴埋め】=0;
                }
        }
}*/

//敵の弾を動かす関数
/*void move_enemybullet()
{
        //すべての敵の
        for(int i=0;i<【穴埋め】;i++)
        {
                //すべての弾
                for(int j=0;j<【穴埋め】;j++)
                {
                        //存在していれば
                        if(enemybullet[i][j].hp)
                        {
                                //x,y 方向の速度を現座標に加える
                                enemybullet[i][j].【穴埋め】 += enemybullet[i][j].
【穴埋め】;
                                enemybullet[i][j].【穴埋め】 += enemybullet[i][j].
【穴埋め】;

                                //画面からいなくなったら、消す
                                if( enemybullet[i][j].x > WIDTH &&
                                        enemybullet[i][j].x                    <
-enemybullet[i][j].size_x&&
                                        enemybullet[i][j].y > HEIGHT &&
```

```
                                        enemybullet[i][j].y                    <
-enemybullet[i][j].size_y )
                                {
                                                enemybullet[i][j].【穴埋め】=0;
                                }

                        }

                }
        }
}*/

//アイテムを動かす関数
void move_item()
{
        //すべてのアイテム
        for(int i=0;i<【穴埋め】;i++)
        {
                //存在していれば
                if(item[i].hp)
                {
                        //x,y 方向の速度を現座標に加える
                        item[i].x += item[i].vx;
                        item[i].y += item[i].vy;
                        //vy を 1 増やして
                        item[i].vy++;
                        //vy が 5 より大きくならないように
                        if(item[i].vy>5)
                        {
                                item[i].vy=5;
                        }
                        //y 方向で画面からいなくなったら、消す
                        if(item[i].y >【穴埋め】)
                        {
                                item[i].hp=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;
}



void create_enemy()//敵を作る関数
{
        //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++)
                {
                        //HP が 0(=今存在していない奴)を探し、敵を作る(HP を与
える)
                        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;
                                }
                        }
                }
}

//自機の弾を作る関数
/*void create_playerbullet()
{
        static int t=0;  //X が連続で入力されていたフレーム数を格納,static だから初
回だけ初期化されて、それからずっと値は保持される

        //X が入力されていたら
        if(key[KEY_INPUT_X])
        {
                //連続入力フレーム数を 1 増加
                t++;
                //3 フレームごとに
                if(t%3==1)
                {
                        //すべての弾の中から
                        for(int i=0;i<【穴埋め】;i++)
                        {
                                //存在していないものをさがし、弾を作る
                                if(playerbullet[i].hp==0)
                                {
                                        //存在を与える
                                        playerbullet[i].hp=1;
```

```
                                        //自機の正面に弾をつくる
                                        playerbullet[i].x = player.x + (player.size_x - playerbullet[i].size_x)/2;
                                        playerbullet[i].y = player.y - playerbullet[i].size_y;
                                        //1 フレームで 1 発だけ作るので、1 発作ったら break で抜ける
                                        break;
                                    }
                                }
                            }
                        }
}*/


//敵の弾を作る関数
//void create_enemybullet()
//{
//      double ang;  //自機と敵の角度を格納する変数
//      //すべての敵で
//      for(int i=0;i<【穴埋め】;i++)
//      {
//          //存在しているやつ
//          if(enemy[i].hp!=0)
//          {
//              //その敵が作られてから何フレームかを格納している変数を 1 増加させる
//              enemy[i].t++;
//              //その敵が作られてから 15 フレーム周期ごとに
//              if(enemy[i].t%15==0)
//              {
//                  //その敵の弾で
//                  for(int j=0;j<【穴埋め】;j++)
//                  {
//                      //存在していないものを探し,弾をつくる
```

```
//                                          if(enemybullet[i][j].hp==0)
//                                          {
//                                                  //存在を与え、(hp!=0 にする)
//                                                  enemybullet[i][j].hp=1;
//                                                  //敵の正面に弾を作る
//                                                  enemybullet[i][j].x = enemy[i].x
+ enemy[i].size_x / 2;
//                                                  enemybullet[i][j].y = enemy[i].y
+ enemy[i].size_y;
//
//                                                  ////自機狙いにしてみた
//                                                  ////ang に自機と敵との角度を入
れる
//                                                  //ang=atan2( (double) ( (player.y
+ player.size_y/2) -(enemy[i].y + enemy[i].size_y) ) , (double)( (player.x +
player.size_x/2) - (enemy[i].x + enemy[i].size_x / 2) ) );
//                                                  //enemybullet[i][j].vx            =
cos(ang)*8;
//                                                  //enemybullet[i][j].vy            =
sin(ang)*8;
//
//                                                  //1 フレームで 1 発しか作りたく
ないので、1 発作ったら break で抜ける
//                                                  break;
//                                          }
//                                  }
//                          }
//                  }
//          }
//}

//アイテムを作る関数
void create_item(int x,int y,int size_x,int size_y)
{
        //すべてのアイテムで
```

```
	for(int i=0;i<【穴埋め】;i++)
	{
		//存在していないもの(hp==0)を探し、アイテムを作る(hp!=0 にする)
		if(item[i].hp==0)
		{
item[i].hp=1;
			item[i].x = x + (size_x - item[i].size_x); //倒された敵の真ん
中を初期値として設定
			item[i].y = y + (size_y - item[i].size_y);
			item[i].vx=0;
			item[i].vy = -10;

			//1 回よびだされたら 1 個しか作りたくないので、1 個作った
ら break で抜ける
			break;
		}
	}
}


void update() //数値的な更新をする関数
{
	//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
                        move_player();
                        【穴埋め】;
                        //【穴埋め】;
                        //move_enemybullet();
                        move_item();
                        move_background();
                        //いろいろと作る関数
                        【穴埋め】;
                        //create_playerbullet();
                        //create_enemybullet();
                        //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;
        }
}

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);

			//自機の描画
			DrawGraph(player.x,player.y,imglist.img_player,TRUE);
			DrawFormatString(0,580,black,"HP:%d",player.hp);
			DrawFormatString(650,50,black," ス コア:%d",player.score);

			//自機の弾の描画
			/*for(inti=0;i<【穴埋め】;i++)
			{
				//存在しているもの(hp!=0)について
				if(playerbullet[i].hp)
				{
					//描画する
DrawGraph( (int)playerbullet[i].x, (int)playerbullet[i].y, imglist.img_bullet , TRUE);
				}
			}*/
			//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,(int)enemy[i].y,imglist.img_enemy,TRUE);
                                        }
                                        /*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_bull
et,FALSE);
                                                }
                                        }*/
                                }

                                //アイテムの描画
                                for(int i=0;i<ITEM;i++)
                                {
                                        //存在しているもの(hp==1)について
                                        if(item[i].hp)
                                        {
                                                //描画する

        DrawGraph(item[i].x,item[i].y,imglist.img_item,FALSE);
                                        }
                                }
                                break;

                        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 ){
        //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 );
        //Dx ライブラリの初期化,失敗したら終了
        if(DxLib_Init()!=0) return 0;

        //裏画面描画を設定
        SetDrawScreen( DX_SCREEN_BACK );
        //画像のロード
        load();
        //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_ESCAPE]) 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;
}
```

4.　#define について

今回のシューテイングゲーム作成において「definexxxxx 100」などと冒頭で宣言しました。これは、数字だけは何の動作が行われているのか判断がしにくい場面において分かりやすくするために使われます。define は主にマクロと呼ばれるもので覚えておいて損はないと思います。

5.　コメントアウトの大切さ

コメントアウトとは「//描画を行う」などといった部分のことです。これは簡単な説明などだと思ってください。ゲーム制作のような膨大なコードが必要な場合などにおいて、分かりやすくする上で非常に大切なものです。自分の脳内での整理や他人が読んで分かりやすいコードを書くためにもコメントアウトを使うように心掛けましょう。