プログラマー勉強会　第四回

１．背景をつける

宣言

```
//敵の情報いろいろ
…

//背景の情報いろいろ
int back_y;
int back_g = LoadGraph("Data/back.bmp");

//ゲームの情報いろいろ
int score;
```

初期化

```
//--------------ゲームの処理をする-----------------
//変数「mode」が0ならば、タイトル画面の処理をする
if (mode == 0) {
        if (key_z == 1) {          //Z キーが押されたら画面をタイトルからメインへ
                …
                back_y = 0;
                score = 0;
        }
}
```

実行すると、止まったままの背景が表示される

描画する

```
//--------------ゲームの描画をする-----------------
//変数「mode」が0ならば、タイトル画面の描画をする
if (mode == 0) {
        …
}
//変数「mode」が1ならば、メイン画面の描画をする
else if (mode == 1) {
        //背景を描画
        DrawGraph(0, back_y, back_g, TRUE);
        DrawGraph(0, back_y - 2400, back_g, TRUE);
        …
```

注意：　背景は必ず一番初めに描画する

　　表示する順番を間違えると・・・

２．背景を動かす

```
//--------------ゲームの処理をする-----------------
//変数「mode」が0ならば、タイトル画面の処理をする
if (mode == 0) {
        …
}
//変数「mode」が1ならば、メイン画面の処理をする
else if (mode == 1) {
        //登場している敵を動かす
        for (i = 0; i < MAX_ENEMY; i++) {
                …
        }

        //背景を移動させる
        back_y += 5;
        if (back_y >= 2400)
                back_y -= 2400;   //背景のループ

        //敵と弾丸の当たり判定
        …
```

実行すると、ちゃんと動いている背景が描画される

応用

１．変数をひとまとめにする

プレイヤーの変数は、こんなにたくさんある

```
//プレイヤーの情報いろいろ
int player_x, player_y, player_life, player_xsize, player_ysize;
int player_safetime;
```

→　もっとゲームを展開していくとなると、さらにたくさん変数が必要になる

→　変数が多くなって、管理が大変になってしまう

→「構造体」を使う

構造体：C 言語が持つ機能。
　　異なるデータ型(int や char　など)を含む複数の変数をまとめることができる。

プレイヤーの情報を構造体でまとめる

```
struct character {
        int x, y, life, xsize, ysize;
        int safetime;
};
```

こう書くと、「character」という複数の変数がまとめられた構造体がつくられる

つくった構造体を使って、変数を宣言する

```
character player;
```

「int ～」「char ～」など、元々C 言語にあるデータ型を使った変数の宣言と同じように宣言できる

変数 player の中には、int 型の「x,y,life…」が詰まっている

それらを使う方法：「構造体で宣言した変数名」.「中に詰まっている変数名」
例：

```
player.x = 0;
```

実際に構造体を使って改造してみる

まずは、構造体を宣言する

```
//ループ用の変数
int i, j;

struct character{
        int x, y, life, xsize, ysize;
        int safetime;
};

//プレイヤーの情報いろいろ
```

いらなくなった変数の宣言を削除する

```
//プレイヤーの情報いろいろ
int player_x, player_y, player_life, player_xsize, player_ysize;
int player_safetime;                //ダメージを受けないセーフタイム
        …
//弾丸の情報いろいろ
int shot_x[MAX_SHOT], shot_y[MAX_SHOT], shot_life[MAX_SHOT],
                                    shot_xsize[MAX_SHOT], shot_ysize[MAX_SHOT];
        …
//敵の情報いろいろ
int enemy_x[MAX_ENEMY], enemy_y[MAX_ENEMY], enemy_life[MAX_ENEMY],
                                    enemy_xsize[MAX_ENEMY], enemy_ysize[MAX_ENEMY];
        …
```

画像用の変数など、構造体でまとめていない変数は消さないように注意する

構造体を使って、今消した変数の代替を宣言する

```
struct character{
                int x, y, life, xsize, ysize;
                int safetime;
};

character player;
character shot[MAX_SHOT];
character enemy[MAX_ENEMY];
```

おそらく大量にエラーが出始めたはずなので、あちこちの変数を正しく書き換える

```
//---------------ゲームの処理をする-----------------
//変数「mode」が0ならば、タイトル画面の処理をする
if (mode == 0) {
        if (key_z == 1) {          //Zキーが押されたら画面をタイトルからメインへ
                mode = 1;
                //ゲームに必要な情報を初期化していく
                player.x = 200;
                player.y = 420;
                player.life = 5;
                player.xsize = 40;
                player.ysize = 40;
                player.safetime = 0;
                shot_trigger = 0;
                for (i = 0; i < MAX_SHOT; i++) {
                        shot[i].x = 0;
                        shot[i].y = 0;
                        shot[i].life = 0;
                        shot[i].xsize = 0;
                        shot[i].ysize = 0;
                }
                for (i = 0; i < MAX_ENEMY; i++) {
                        enemy[i].x = 0;
                        enemy[i].y = 0;
                        enemy[i].life = 0;
                        enemy[i].xsize = 0;
                        enemy[i].ysize = 0;
                }
                …
```

こんな感じで、ほかのエラーが出ている部分を全て書き換える

書き換える法則としては、

player_x　　→　　player.x

shot_life[i]　　→　　shot[i].life

enemy_xsize[i]　→　　enemy[i].xsize

のようになる

全て書き換えるのはちょっとめんどくさい・・・

→　visual Studio には、文字列の置換機能がある

検索する文字列を「player_x」とし、
置換後の文字列を「player.x」、
検索対象を「現在のプロジェクト」にして右下の「すべて置換」ボタンを押す

→　Source.cpp 内の全ての「player_x」という文字列が、
「player.x」に置き換わった

実行して、今まで通り問題なく動作したら改造完了

２．関数を使ってみる

Source.cpp の一番上

```
#include "DxLib.h"

#define WINDOW_XSIZE 640
#define WINDOW_YSIZE 480

#define MAX_SHOT 30
#define MAX_ENEMY 50

void TestMessage();          //関数のプロトタイプ宣言

int WINAPI WinMain(HINSTANCE hI, HINSTANCE hp, LPSTR lpC, int nC) {
        …
```

Source.cpp の一番下

```
        …
        DxLib_End();
        return 0;
}

void TestMessage() {
        DrawString(0,0,"Shooting Game!!",GetColor(255,255,255));
}
```

で、メイン画面の描画で・・・

```
//--------------ゲームの描画をする-----------------
//変数「mode」が0ならば、タイトル画面の描画をする
if (mode == 0) {
        …
}
//変数「mode」が1ならば、メイン画面の描画をする
else if (mode == 1) {
        …
        DrawString(460, 100, "life", GetColor(255, 255, 255));
        for (i = 0; i < player.life; i++) {
                DrawGraph(480 + i * 25, 120, plife_g, TRUE);
        }
        TestMessage();
}
```

実行すると、左上に「Shooting Game!!」と表示されるはず

今度は、タイトル画面とゲームオーバー画面でもTestMessage()と書いてみる

```
//--------------ゲームの描画をする-----------------
//変数「mode」が0ならば、タイトル画面の描画をする
if (mode == 0) {
        //文字を描画する
        DrawString(20, 20, "タイトル画面", GetColor(255, 255, 255));
        TestMessage();
}
//変数「mode」が1ならば、メイン画面の描画をする
else if (mode == 1) {
        …
        TestMessage();
}
//変数「mode」が2ならば、ゲームオーバー画面の描画をする
else if (mode == 2) {
        DrawString(20, 20, "Game Over", GetColor(255, 255, 255));
        DrawFormatString(20, 100, GetColor(255, 255, 255), "Total Score : %d", score);
        TestMessage();
}
```

異なる画面のはずなのに、全く同じ文字列が出力される

関数：　処理を別の場所に分けて描くことができる機能

　　複数回呼び出すと、同じ処理がその場で行われる

３．ファイルの読み書きをする

C 言語では、テキストファイルなどにデータを書き込むことができる

今までの最大スコアを保存する機能をつける

```
//ゲームの情報いろいろ
int score;
int max_score = 0;
```

ゲームオーバー画面からタイトルに戻った時、最大スコアが更新されるようにする

```
//--------------ゲームの処理をする-----------------
//変数「mode」が0ならば、タイトル画面の処理をする
if (mode == 0) {
}
//変数「mode」が1ならば、メイン画面の処理をする
else if (mode == 1) {
}
//変数「mode」が2ならば、ゲームオーバ―画面の処理をする
else if (mode == 2) {
	//Zキーが押されたらタイトル画面に戻る
	if (key_z == 1) {
		mode = 0;
		//もし今回のスコアが最大スコアより高ければ、最大スコアを更新
		if (score > max_score)
			max_score = score;
	}
}
```

最大スコアをタイトル画面に表示する

```
//--------------ゲームの描画をする-----------------
//変数「mode」が0ならば、タイトル画面の描画をする
if (mode == 0) {
	//文字を描画する
	DrawString(20, 20, "タイトル画面", GetColor(255, 255, 255));
	DrawFormatString(20, 60, GetColor(255, 255, 255), "最大スコア：%d", max_score);
}
```

最大スコアが表示されるようになったが、

アプリケーションを終了すると 0 に戻ってしまう

ファイルに保存する

```
//--------------ゲームの処理をする-----------------
//変数「mode」が0ならば、タイトル画面の処理をする
if (mode == 0) {
}
//変数「mode」が1ならば、メイン画面の処理をする
else if (mode == 1) {
}
//変数「mode」が2ならば、ゲームオーバー画面の処理をする
else if (mode == 2) {
	//Zキーが押されたらタイトル画面に戻る
	if (key_z == 1) {
		mode = 0;
		//もし今回のスコアが最大スコアより高ければ、最大スコアを更新
		if (score > max_score)
			max_score = score;
		FILE *wf;
		fopen_s(&wf, "Data/max_score.txt", "w");
		if (wf != NULL) {
			fwrite(&max_score, sizeof(int), 1, wf);
			fclose(wf);
		}
	}
}
```

max_score.txt に、最大スコアが書き込まれるようになった

ゲーム開始時に読み込む

```
//ゲームの情報いろいろ
int score;
int max_score = 0;

FILE* rf;
//ファイルを読み込む
fopen_s(&rf, "Data/max_score.txt", "r");
//ファイルの読み込みに成功したら、そこから値を読み込む
if (rf != NULL) {
	fread(&max_score, sizeof(int), 1, rf);
	fclose(rf);		//ファイルを閉じる
}

//メインループ
while (ProcessMessage() == 0 && CheckHitKey(KEY_INPUT_ESCAPE) == 0) {
	…
```

max_score.txt から、最大スコアが読み込まれるようになった

これで、アプリケーションを終了しても最大スコアが 0 に戻らなくなったはず