ソフトゼミ B 2016 第 6 回

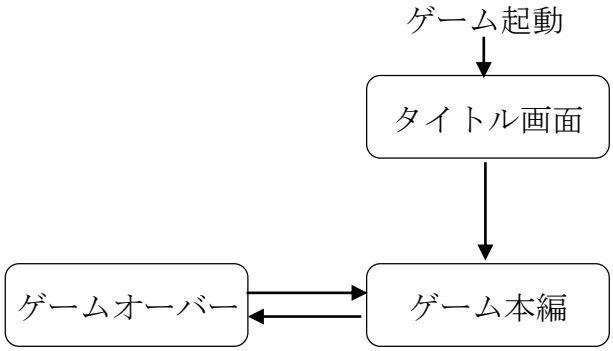
プレイヤーの移動と重力

前回までのシューティングゲームの概要は理解できたでしょうか？今回からアクションゲームの製作に入っていきます．アクションでは新たに「重力」と「ブロック判定（壁判定)」という要素が入ってきます．今回はそのうちの「重力」について解説していきます．わからないところがあったら気軽に質問してください．

まずは新しいプロジェクト「action」を Visual Studio に作ってください．その後「main.cpp」という項目をプロジェクトに追加し，プロパティから設定を行ってください．第 1 回の資料を参照するといいでしょう．
ここまで終わったら，配布されたソースコードを main.cpp にコピペしてください．これで準備は完了です．

➢ ゲーム全体の構造

今回のゲームは次のような構造をしています．



プレイヤーの移動だけを考えたいので，クリア画面はありません．（次回に実装します）

➢ 「重力」の考え方

このゲームのプログラムでは，プレイヤーの座標を（現在の x,y 座標）＋（現在の x,y 方向の速さ）で決めています．したがって重力をかけるには，プレイヤーが空中にいる場合に限って y 方向下向きの速さを徐々に増やしていけばよい

ことになります．逆に着地していればｙ方向の速さは０にします．

➢ ソースコード

それでは，ソースコードを以下に示します．穴埋め部分があるので，コメントを参照しながらうめていってください．わからなければお近くのエレ研部員まで．

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

#define WINDOW_WIDTH 640   //ウィンドウの大きさ(横)
#define WINDOW_HEIGHT 480 //ウィンドウの大きさ(縦)
#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                     //攻撃の大きさ
#define WIDTH_SIZE 35                      //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]

#pragma warning(disable：4996)//fopen,fscanf でエラーが出るのを防ぐ
                    //このエラーの原因は「マイクロソフトの余計なおせっかい」です

struct SImg{
        int migi;
        int r_attack;
        int kabe;
        int yuka;
        int goal;
        int r_enemy;
        int gameover;
        int title;
        int clear;
        int clear2;
        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
	int kakusi;
};

//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 SImg img;
struct SPlayer player;
struct SEnemy enemy;
int map[WIDTH_SIZE][HEIGHT_SIZE];
char keyState[256];
```

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

	if ((file=fopen("map/map_proto.txt","r"))== NULL){
		//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
		OutputDebugString("MapData Read Error¥n");
		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]);//カンマに気を付けてください
		}
	}
	fclose(file);
}

//画像読み込み
void image(void) {

	img.migi = LoadGraph("img/migi.png");
	img.r_attack = LoadGraph("img/r_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.gameover = LoadGraph("img/gameover.png");
	img.title = LoadGraph("img/title.png");
	img.clear = LoadGraph("img/clear.png");
	img.clear2 = LoadGraph("img/clear2.png");
	img.haikei = LoadGraph("img/haikei.png");
	img.null = LoadGraph("img/null.png");

}
```

```
//変数の初期化
void init(void) {

	player.x = 32;			//主人公の初期 x 座標
	player.y = 400;			//主人公の初期 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;
	player.kakusi = 0;

	enemy.size_x = 32;
	enemy.size_y = 32;
	enemy.x = 300;
	enemy.y = 480 - BLOCK - enemy.size_y;
	enemy.vx = 0;
	enemy.vy = 0;
	enemy.count = 0;
	enemy.life = 1;
	enemy.dire = 0;

}


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)
{	//エスケープキーおよび X キー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;
	}
}

//ゲームオーバー画面の表示＋マップのリロード＋変数の初期化
void gameover(void) {

	init();				//変数の初期化

	ClearDrawScreen();
	DrawGraph(0, 0, img.gameover, TRUE);	//ゲームオーバー画面を描画
	ScreenFlip();

while (keyState[KEY_INPUT_ESCAPE] != 1 && keyState[KEY_INPUT_X] != 1)
{		//エスケープキー及び X キー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;
	}
	Map();	//マップのリロード
}



//壁判定　(上下判定/ブロックの上に乗る)
void judge_v(int blx, int bly) {
	//主人公とブロックが重なっていたら
if (blx < (player.x + HERO_SIZE) && player.x < (blx + BLOCK) && bly <= (player.y +
HERO_SIZE) && player.y < (bly + BLOCK)) {

		if (【穴埋め１】) {//落ちてる最中あるいは y 方向には動いていないなら
			player.jfly = 0;
			player.y = (bly - HERO_SIZE);
			player.vy = 0;
		}
		else {//上向きに動いているなら
```

```
                player.vy = 0;
                player.y = (bly + BLOCK);
            }

        }

        int white = GetColor(255, 255, 255);
DrawFormatString(550, 120, white, "J2:%4d", map[blx / BLOCK][blx / BLOCK]);

}

//主人公の移動
void move(void) {

        player.vx = 0;                                  //横移動リセット

        if (player.jfly != 0 && player.vy < MAXSPEEDY)     //重力(空中にいれば重力を
働かせる)
            player.vy += 1;

        player.kabe_l = 0;          //壁判定リセット(今回は気にしなくて OK です)
        player.kabe_r = 0;

        GetHitKeyStateAll(keyState);      //キーボードのすべてのキーの状態を取得
                                          //その状態は変数 keyState に記録される

        if (keyState[KEY_INPUT_LEFT]) {                 //主人公の左向き操作(もし
        左ボタンが押されていたら)
            if (player.kabe_l == 0 && player.x > 0) {
                player.vx -= 4;    //主人公の x 方向の速さを 4 減らす
            }
             【穴埋め 2】;//主人公は左向き
        }

        if (keyState[KEY_INPUT_RIGHT]) {                //主人公の右向き操作(もし
右ボタンが押されていたら)
```

```
		if (player.kabe_r == 0 && player.x + HERO_SIZE <= BLOCK *
	WIDTH_SIZE) {
			player.vx += 4;  //主人公の x 方向の速さを 4 増やす
		}
		player.dire = 0;// 主人公は右向き
	}

	if (player.jfly == 0 && keyState[KEY_INPUT_X]) {		//プレイヤーのジ
ャンプ操作(空中にいないときに X キーが押されたら)
		player.jfly = 1;		//主人公は空中にいる
		player.vy -= 15;		//主人公の y 速度を変化させる(上方向の速さ)
	}


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



	player.jfly = 1;


	player.x += player.vx;		//主人公を x 速度分だけ移動
	player.y += player.vy;		//主人公の y 速度分だけ移動


	//壁判定の呼び出し
	for (int j = 0; j<HEIGHT_SIZE; j++) {
		for (int i = 0; i<WIDTH_SIZE; i++) {
			if (map[i][j] != 0)
				judge_v(i*BLOCK, j*BLOCK);
		}
	}

	//敵が存在していて，プレイヤーとぶつかったら，ゲームオーバー
	if (【穴埋め 3】) {
```

```
		if (enemy.life != 0) {
			gameover();
		}
	}


	if (player.y>【穴埋め 4】) {//画面の外に出たら（下に落ちたら）ゲームオーバー
		gameover();
	}

	if (player.x == 0 && (player.y > 400 && player.y < 402))
		player.kakusi += 1;
}

//敵キャラの移動
void enemymove(void) {

	if (【穴埋め 5】) {  //敵が存在していて
		if (enemy.vx >= 0) {			//敵の速度が右向きなら
			enemy.vx = 2;			//速度は右向き
			enemy.count++;
			【穴埋め 6】;			//敵は右を向く
		}
		else {					//敵の速度が左向きなら
			enemy.vx = -2;			//速度は左向き
			enemy.count--;
			【穴埋め 7】;			//敵は左を向く
		}
	}

	if (enemy.count >= 100)  //ある程度動いたら速度の向きを変える
		enemy.vx = -2;
	else if (enemy.count <= 0)
		enemy.vx = 2;

	enemy.x += enemy.vx;			//敵の x 座標に速度を加算する
```

```
	enemy.y += enemy.vy;			//敵のｙ座標に速度を加算する

	if (【穴埋め 8】) {//もし主人公が攻撃中で
		if ((【穴埋め 9】) ||	//右を向いていた時の処理（攻撃が敵に当たれば）

		(player.dire == 1 && (enemy.x < player.x + HERO_SIZE +
ATTACK_SIZE && player.x - ATTACK_SIZE < enemy.x + enemy.size_x) &&
	(enemy.y < player.y + HERO_SIZE && player.y < enemy.y + enemy.size_y) ) )
				//左を向いていた時の処理（攻撃が敵に当たれば）

				enemy.life = 0;//敵は死ぬ
	}

}



//主人公の攻撃
void attack(void) {

	GetHitKeyStateAll(keyState);
	if (【穴埋め 10】) {		//Z キーが押されていたら
		if (player.attack == 0)
			player.attack = 1;//主人公は攻撃中
	}
	else
		player.attack = 0;
}


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 (int i = (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
			else DrawGraph(i*BLOCK - player.x + CENTER, j * BLOCK, img.kabe, TRUE);
		}
	}

	if (player.dire == 0)			//主人公の描画（右向きなら）
		DrawGraph(CENTER, player.y, img.migi, TRUE);		//右向き
	else	//主人公の描画（左向き）
		DrawTurnGraph(CENTER, player.y, img.migi, TRUE);
	//DrawTurnGraph でその画像を左右反転
					//させて表示させる
```

```
	if (player.attack) {				//攻撃の描画(攻撃中で)
		if (player.dire == 0) //右向きなら
			DrawGraph(CENTER + ATTACK_SIZE, player.y,
img.r_attack, TRUE);
		else			//左向きなら
			DrawTurnGraph(CENTER - ATTACK_SIZE, player.y,
img.r_attack, TRUE);
	}


	if (enemy.life > 0) {				//敵の描画
		if (enemy.dire == 0)			//向きの判別(右向きなら)
			DrawGraph(enemy.x - player.x + CENTER, enemy.y,
img.r_enemy, TRUE);
		else					//左向きなら
			DrawTurnGraph(enemy.x - player.x + CENTER, enemy.y,
	img.r_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
		image();
```

```
	//変数の初期化
	init();

	SetDrawScreen(DX_SCREEN_BACK); //裏画面処理

	//タイトル表示
	title();

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

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

		//動き
		move();
		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;

}
```

➢ マップチップの読み込み

シューティングゲームには無かった，マップチップの読み込みについて説明します．このゲームでは map フォルダのなかの map_proto.txt というテキストファイルからマップチップの読み込みをしています．

このようにしてコンピュータ上のファイルからデータを読み込みたい場合があります．このときに使うのが fopen や fscanf などの関数と FILE 型のポインタです．

Map()でしていることを簡単に言うと，fopen 関数で読み込みたいファイルを開いて，fscanf でそのファイル内容を読み込み，終わったら fclose でファイルを閉じる，ということをしています．詳しいことは自分で調べるなりエレ研部員に聞くなりしましょう．

ちなみに，ここでのマップチップとは，マップの特定の場所にどの画像を表示するかというデータを入れておくものです．

➢ 今回使った DX ライブラリのライブラリ関数

以下に一覧で示します．詳しい仕様と使用法は DX ライブラリのホームページを参照してください．

LoadGraph	SetGraphMode
ClearDrawScreen	SetWindowSizeChangeEnableFlag
DrawGraph	ChangeWindowMode
DrawTurnGraph	SetMainWindowText
SetDrawScreen	
ScreenFlip	GetColor
DrawFormatString	
DrawStirng	DxLib_Init
	DxLib_End
GetHitKeyStateAll	ProcessMessage
CheckHitKey	

➢ 次回に向けて

今回の map フォルダの中には map_proto.txt ファイルだけではなく map.txt ファイルというファイルもあります．Map()のなかで読み込んでいるファイルを map.txt に変更して実行してみてください．ブロックが宙に

浮いている，先ほどとは異なるマップが表示されるはずです．

そしてこのブロックに横からぶつかると変なことが起こるはずです．なぜそうなるのか考えてみましょう．ヒントは judge_v．

次回「ブロック判定」ではこの問題を解決するための方法を教えます．

今回までのプログラムで「ここってどういう意味があるの？」と思った部分があったら遠慮なく聞いて下さい．プログラムを自分で書き替えてみるのも勉強になると思います φ(-ω-)))≡=—．