第 2 回　画像の制御と初期化

第 1 回では、画面をセットし画像を表示しました。今回は画像を動かしてみることにします。また、前回少し触れましたが変数の初期化についても説明していきます。

まず、前回作った構造体に下記の変数を追加してください

int sokudo_x;　　int sokudo_y;

int kasoku_x;　　int kasoku_y;　　int lr;

変数名を見ての通り画像を動かす時の速さと加速度を入れておくための変数です。
加速度は慣性を付けたり文字通り加速・減速を扱うのに使います。lr は後に説明します。

画像を動かすというのはとても簡単で画像の座標を変えるだけで動いてくれます。
下記のサンプルを見て下さい

```
void g_jiki( void){
	//自機に関する関数

	jiki.x += jiki.sokudo_x;
	jiki.y += jiki.sokudo_y;
	//今の速度を実際の座標に代入する

	DrawGraph( jiki.x - JIKI_HABA / 2, jiki.y - JIKI_TAKASA / 2,
	jiki.handle, true);

}
```

前回作った自機を表示させる関数内に網かけの部分を追加しています。
jiki.sokudo_x や jiki.sokudo_y に数値を代入するか、上記の jiki.sokudo_x や jiki.sokudo_y の代わりに数字（1～5）を入れてみてください。
画像が動き続けるはずです。

ここで、少し話が逸れるのですが初期化について話します。
ゼミＡでも聞いたと思いますが、宣言しただけの変数にはわけのわからない数値が入っています。そのため初期化が必要になります。宣言する際に int a=0; と書けばいいと思うかもしれませんが、ゲームではゲームオーバーやタイトルに戻ったりゲームの途中に初期化しなければならないことが多々あり、ゲームを作る上でこの初期化を行う関数を作っておくことは非常に大事です。

ここで、グローバル変数や定義を下記のように追加してください。

```
#define JIKI_SOKUDO 8          //自機の歩く速度
int key_up;          int key_down;
int key_left;        int key_right;
int key_a;           int key_b;
```

この key～は後に説明しますが、キーボードで操作を行うためのものです。

次に初期化の話に戻ります。下記のサンプルを見て下さい

```
void m_format( void){
        //いろいろと初期化する関数

        SetDrawScreen( DX_SCREEN_BACK);
        SetWaitVSyncFlag( FALSE);
        //垂直同期は待ちませんよ？
        time = 0;

        key_left = 0;
        key_right = 0;
}
```

```
void g_format( void){
        //ゲームの初期化を行う関数

        int i;
        int j;
        //i、j は for 文用。

        jiki.x = 300;
        jiki.y = 200;
        //初期位置  X = 300, Y = 200 の位置

        jiki.haba = JIKI_HABA;
        jiki.takasa = JIKI_TAKASA;
        //自機の画像の幅と高さの代入
        //GetGraphSize という便利な関数を使うとこんなことしなくていい

        jiki.lr = 1;
        //画像の左右を初期化。右が 1。左が-1

        jiki.handle = LoadGraph( "data/jiki.bmp");

        jiki.sokudo_x = 0;
        jiki.sokudo_y = 0;
        //速度を初期化
```

```
	for( i = 0; i <= TIP_MAX_X; i++){
		for( j = 0; j <= TIP_MAX_Y; j++){
			tip[i][j] = 0;
		}
	}
	//チップを初期化。前回の g_loop()にあったやつをコピペすれば ok かと

	//tip[i][j] == 1 でチップが存在する。 0 ならなし。

	tip_handle = LoadGraph( "data/tip.bmp");
	//チップ画像のロード
}
```

このように、初めに使う画像や画面のセットなども一緒にしておくと便利です。for 文を使うことにより配列を使った変数全体の初期化が簡単になります。lr ってのが出てきましたが今は無視してかまいません。

m_format はメイン文に g_format は g_loop 内に追加して初期化しましょう。

あと新しい関数を作ったらプロトタイプ宣言を忘れないこと……

それでもって、SetWaitVSyncFlag って何よ？と思いますよね

SetWaitVSyncFlag(TRUEorFALSE);

これは、CRT の垂直同期信号を待つか否かを決めるもので、

TRUE : 待つ（デフォルト）FALSE : 待たない

となっています。ぶっちゃけ無くても一向にかまいません。ScreenFlip 時に待たない分処理が早くなるかもしれませんが、もし異常（画面が波打ったり）が出るようなら TRUE にするか書かない方が無難です(モニタの仕組みの話になるのでスルーしても構いません)。

それでは、初期化も終わったので実際に画像を動かしてみましょう

CheckHitKey(int KeyCode) ;

特定キーの入力状態を得るために使います。鍵っ子とは関係ありません。

int KeyCode には KEY_INPUT_～の形のキーコードが入ります。

代表的なのは

KEY_INPUT_UP （上キー）	KEY_INPUT_DOWN （下キー）
KEY_INPUT_RIGHT（右キー）	KEY_INPUT_LEFT （左キー）
KEY_INPUT_（使いたいアルファベット）	KEY_INPUT_RETURN（エンターキー）

です。

詳しいことや他のキーについては DX ライブラリのリファレンスを参照してみてください。かなりの量があります。

では、これを基に下記の新しい関数を作ります。

```
void m_key( void){
        //キーからの入力を変数に返す関数

        if( CheckHitKey( KEY_INPUT_RIGHT) == 1) key_right++;
        else key_right = 0;
        if( CheckHitKey( KEY_INPUT_LEFT) == 1) key_left++;
        else key_left = 0;
        if( CheckHitKey( KEY_INPUT_Z) == 1) key_a++;
        else key_a = 0;
        if(CheckHitKey(KEY_INPUT_X)==1||
          CheckHitKey( KEY_INPUT_RETURN) == 1) key_b++;
        else key_b = 0;

        //現在、押されている状態が何フレーム目かを記録する方式。
        //ボタンを離すと 0 になり、ボタンを押すと押しているフレーム分加算される
}
```

さて、これはいったい何をしているのか考えてみましょう。コメントアウトに書いてあるとおり押してある時間を記録しているんでしょ、そんなことはわかる。でも何故こんなことしているの？と思うかもしれません。でも、こうすると便利な事があります、わかりますか？

では、一つの答えです。例えばジャンプを考えてみましょう軽く押したときと、強く押したときで変化をつけられますよね。その他にも加速に影響を与えたりと様々な事に応用できます。

さて、

実際に書き込んでみたらゲームのループ内に

```
m_key();
printfDx(右%d,key_right) ;
clsDx() ;
```

と書いてみて値が変動するか確かめてみてください。

printfDx(%d とか%f,表示させる値を持つ変数) ;

これは DX ライブラリで画面に表示させるために使います。あくまで簡易的なものでちょっと動作を確認したいときに便利です。使い方は見ての通り printf と変わりません。

clsDx(void) ;

これは上記の簡易画面出力をクリアするためのもの。消さないと永遠に残りますよ……

キーの確認が終わったら邪魔なので printfDx と clsDx は消すなりコメントアウトで省くなりして下さい。

やっとキー入力の検出が終わりましたので、実際に画像を自分の思いどおりに動かしてみましょう。

下記の関数を見て下さい。

```
void g_jiki_sousa( void){
	jiki.kasoku_x = 0;
	jiki.kasoku_y = 0;
	//加速度を初期化。
	//これを毎フレームごとに初期化しないと加速度が速度と同じになる
	//微分微分♪

	if( key_right > 0){
		jiki.kasoku_x += JIKI_SOKUDO;
		jiki.lr = 1;
	}
	else if( key_left > 0){
		jiki.kasoku_x += -JIKI_SOKUDO;
		jiki.lr = -1;
	}
	//右キー、左キーからの入力があれば、その方向へ加速させる

	jiki.sokudo_x *= 95 / 100;
	//摩擦係数 0.05。これを削除すると慣性でずっと動く

	jiki.sokudo_x += jiki.kasoku_x;
	jiki.sokudo_y += jiki.kasoku_y;
	//加速度を速度に代入する
}
```

やっと lr の説明、見ればわかりますよね・・・。要するに自機の左右の進行方向を保存しています。何故こんなのがいるかはまだ待ってね。

とりあえず、左右に動くようになっています。摩擦とか物理が嫌いな人には辛いけど頑張ってください。この世界の法則を決めましょう（笑）

摩擦などの値を変えてみましょう。だいたいどんな感じかが掴めてくるはずです。スクロールしないのでわかりづらいですが値を変えても速度の減衰が早いのには理由があります。理由を考えてみましょう。答えはあえて書きません。

課題

左右だけでなく上下も動かせるようにしましょう。
他のキーにも割り当ててみよう(ex:a を押すと速く動く,b を押すと上に動く)

初期化を忘れないようにすること

次で最後です。

さっきからずっと先送りにしていた lr これで進行方向がわかるわけです。

顔がどっちに動いてていても同じ方向を向いているのは気味が悪いですよね。

つまり、lr の値を見て画像の表示をすれば、気味が悪くない！！ということ

さてここで、

DrawTurnGraph(x, y, 読み込む画像のハンドル,TRUEorFALSE) ;

こんなのを使います。これは、メモリに読みこんだグラフィックを反転描画することができます。

x,y はもちろん画像の左上頂点の座標、TRUE で透過色が有効 FALSE で無効と DrawGraph と何ら変わりありません。

ここで、左右反転させた画像を作ってメモリに読み込ませて表示させればいいじゃんと思うかもしれませんがよく考えて下さい。メモリは無限にはありませんできる限り節約しましょう。ロードにも時間がかかりますしゲームを重くすることにもつながります。

＊これに関連して・・・

メモリにロードする作業をループ内に誤って書き込むと・・・

もちろん無限にロードし続けて大変なことになります。

えっ？　俺のパソコンのメモリは 4GB だって？

400KB の画像を 1 フレームずつ一回読み込むと 2 分半ほどでメモリから溢れます。

課題

では、この DrawTurnGraph を用いて画像の表示を行ってみましょう。
画像の表示は g_jiki で行っていましたね。

最後に・・・今回までに出来たソース全文（次回に必要な部分のみ）

```
#include "DxLib.h"

～中略～

#define JIKI_SOKUDO 8
//自機の歩く速度


//以下グローバル変数
int time;

int key_left;
int key_right;

struct mono{

		～中略～

		int lr;
		int sokudo_x;
		int sokudo_y;
		int kasoku_x;
		int kasoku_y;

		int handle;
};

//以下プロトタイプ宣言
void m_format( void);
int m_escape( void);
void m_key( void);

void g_format( void);
void g_loop( void);
void g_tip_draw( void);
void g_jiki( void);
void g_jiki_sousa( void);
//ここまで


int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow){
		ChangeWindowMode( true);
		if(DxLib_Init() == -1){
			return -1;
		}

		m_format();

		g_loop();
```

```
	DxLib_End();
	return 0;
}

void m_format( void){
	//いろいろと初期化する関数

	SetDrawScreen( DX_SCREEN_BACK);
	SetWaitVSyncFlag( FALSE);

	time = 0;

	key_up = 0;
	key_down = 0;
	key_left = 0;
	key_right = 0;
	key_a = 0;
	key_b = 0;
}

void m_key( void){
	//キーからの入力を変数に返す

	//後でジャンプを作るので上下はいらなのです
	if( CheckHitKey( KEY_INPUT_RIGHT) == 1) key_right++;
	else key_right = 0;
	if( CheckHitKey( KEY_INPUT_LEFT) == 1) key_left++;
	else key_left = 0;
	if( CheckHitKey( KEY_INPUT_Z) == 1) key_a++;
	else key_a = 0;
	if(CheckHitKey(KEY_INPUT_X)==1||
	CheckHitKey( KEY_INPUT_RETURN) == 1) key_b++;
	else key_b = 0;
	//あとあと使いたいので

	//現在、押されている状態が何フレーム目かを記録する方式。
	//ボタンを離すと 0 になり、ボタンを押すと押しているフレーム分加算される
}

int m_escape( void){
	～中略～
}


//ここから先は後の回で分割するよ

～中略～
```

```
void g_loop( void){
	//ゲームのメインループ

	g_format();
	//ゲームの初期化

	while( m_escape() >= 0){
		//m_escape の関数は-1 の戻り値でエラーになるので、それ以外は基本ループ

		m_key();		//キーの情報を取得

		g_tip_draw();		//チップの描画をする

		g_jiki();		//自機に関することをまとめた関数

		time++;		//フレーム数をカウントする変数

		ScreenFlip();		//描画

		WaitTimer( 17);

		ClearDrawScreen();	//画面をクリアにする
	}
}

void g_format( void){
	//ゲームの初期化

	int i;
	int j;
	//i、j は for 文。

	jiki.x = 300;
	jiki.y = 200;
	//初期位置  X = 300, Y = 200 の位置

	jiki.haba = JIKI_HABA;
	jiki.takasa = JIKI_TAKASA;
	//自機の画像の幅と高さの代入
	//GetGraphSize という便利な関数を使うとこんなことしなくていい

	jiki.lr = 1;
	//画像の左右を初期化。右が 1。左が-1

	jiki.handle = LoadGraph( "data/jiki.bmp");

	jiki.sokudo_x = 0;
	jiki.sokudo_y = 0;
	//速度を初期化
```

```
	for( i = 0; i <= TIP_MAX_X; i++){
		for( j = 0; j <= TIP_MAX_Y; j++){
			tip[i][j] = 0;
		}
	}
	//チップを初期化
	//tip[i][j] == 1 でチップが存在する。 0 ならなし。


	tip_handle = LoadGraph( "data/tip.bmp");
	//チップ画像のロード
}

void g_tip_draw( void){
	int i;
	int j;
	for( i = 0; i <= TIP_MAX_X; i++){
		for( j = 0; j <= TIP_MAX_Y; j++){
			if( tip[i][j] == 1){

				DrawGraph( i * TIP_HABA - TIP_HABA / 2, j *
TIP_TAKASA - TIP_TAKASA / 2, tip_handle, false);
			}
		}
	}
}


void g_jiki( void){
	//自機に関する関数

	g_jiki_sousa();
	//キーからの入力を自機に入力する関数

	jiki.x += jiki.sokudo_x;
	jiki.y += jiki.sokudo_y;
	//今の速度を実際の座標に代入する

	if( jiki.lr == 1){
		DrawGraph( jiki.x - JIKI_HABA / 2, jiki.y - JIKI_TAKASA / 2,
	jiki.handle, true);
	}
	else{
		DrawTurnGraph( jiki.x - JIKI_HABA / 2, jiki.y - JIKI_TAKASA / 2,
	jiki.handle, true);
	}
}

void g_jiki_sousa( void){
```

```
		jiki.kasoku_x = 0;
		jiki.kasoku_y = 0;
		//加速度を初期化。
		//これを毎フレームごとに初期化しないと加速度が速度と同じになる
		//微分微分♪

		if( key_right > 0){
			jiki.kasoku_x += JIKI_SOKUDO;
			jiki.lr = 1;
		}
		else if( key_left > 0){
			jiki.kasoku_x += -JIKI_SOKUDO;
			jiki.lr = -1;
		}
		//右キー、左キーからの入力があれば、その方向へ加速させる

		jiki.sokudo_x *= 95 / 100;
		//摩擦係数 0.05。これを削除すると慣性でずっと動く
		//double 型じゃないからすぐに減衰するのが難点
		//だったら double 型で初めから作ればいいって？
		//int でやって不具合出て、その後 double 型にするってのを覚えるんだよ、きっと

		jiki.sokudo_x += jiki.kasoku_x;
		jiki.sokudo_y += jiki.kasoku_y;
		//加速度を速度に代入する
}
```

以上お疲れ様でした。