平成 24 年 11 月 23 日
京都工芸繊維大学コンピュータ部

Lime 46

はじめに

こんにちは。部長の峯岡です。これを読んでいただいているということは、今年も無事、Lime を発行することができたということでしょう。

さて、Lime も今回で第 46 号です。Lime は、部員の活動の一部を記したものです。日々の研究や開発の成果、興味を持った事に対する調べ物や、果ては個人的趣味全開なもの。是非、最後までお付き合いください。

平成 24 年 11 月 16 日

京都工芸繊維大学コンピュータ部部長 峯岡 健人

目 次

1 Lua で Hello, world まで on Android

情報工学専攻 修士 2 回 荒木 修

1.1 はじめに

本記事では、「Android のアプリを作りたい。でも、Java じゃなくて Lua でコードを書きたい。」な人のために、Android SDK（正確には Android NDK）から Lua のコードを実行して Hello, world! を表示するまでの方法を解説していきます。

1.2 開発環境

本記事は、以下の開発環境を想定しています。

- Eclipse Indigo SR2
- Android SDK 20.0.3 (2.2 APIs を使用)
- Android NDK r8b

さらに、SurfaceView で画面に文字が描画でき、Android NDK で C 言語で書かれたコードが実行可能である状態を前提に、説明します[1]。

1.3 Lua のコードを実行できるようにする

最初に、Android NDK から Lua で書かれたコードを実行できるようにします（Lua のソースコードのコンパイル方法は [1]、assets 内のファイルを native 関数から読み込む方法は [2] を参考にしました）。

まず、Lua のソースコードを http://www.lua.org/からダウンロードします。今回は、Lua-5.2.1.tar.gz をダウンロードしました。ダウンロードしたファイルを展開して、src 内のファイルを Android Application Project の jni/Lua フォルダにコピーします（図 1.1）。

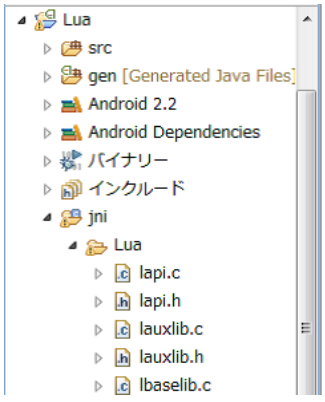

図 1.1: Lua ソースコードの配置

次に、Lua のソースコードがコンパイルされるように Android.mk ファイルに追記します。

[1]本当は、Android NDK の導入ぐらいから書くつもりだったが、筆者の怠慢により、いろいろ省略した結果、いつのまにか「想定されるユーザが存在しているのか？」な状態からとなった（汗）

Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := lua-jni
LOCAL_SRC_FILES := lua-jni.c \
 Lua/lapi.c Lua/lauxlib.c Lua/lbaselib.c Lua/lbitlib.c \
 Lua/lcode.c Lua/lcorolib.c Lua/lctype.c Lua/ldblib.c Lua/ldebug.c \
 Lua/ldo.c Lua/ldump.c Lua/lfunc.c Lua/lgc.c Lua/linit.c \
 Lua/liolib.c Lua/llex.c Lua/lmathlib.c Lua/lmem.c Lua/loadlib.c \
 Lua/lobject.c Lua/lopcodes.c Lua/loslib.c Lua/lparser.c Lua/lstate.c \
 Lua/lstring.c Lua/lstrlib.c Lua/ltable.c Lua/ltablib.c Lua/ltm.c \
 Lua/lua.c Lua/lundump.c Lua/lvm.c Lua/lzio.c

include $(BUILD_SHARED_LIBRARY)
```

ただし、このままコンパイルすると Lua/llex.c でエラーが見つかるので、202 行目をコメントアウトして書き換えます。

Lua/llex.c

```
...
#if !defined(getlocaledecpoint)
//#define getlocaledecpoint() (localeconv()->decimal_point[0])
#define getlocaledecpoint() '.'
#endif
...
```

これで、Lua のソースコードがコンパイルされるようになりました。
次に、Android SDK から Lua が動かせるようにするために、native 関数を定義します。

MainSurfaceView.java

```
...
public class MainSurfaceView extends SurfaceView
implements SurfaceHolder.Callback
{
    static { System.loadLibrary("lua-jni"); } /* ネイティブライブラリをロードする。*/

    native private String cmain(String path); /* Lua コードを実行するネイティブ関数 */
    ...
}
```

lua-jni.c

```
#include <jni.h>
#include "Lua/lua.h"
#include "Lua/lauxlib.h"

JNIEXPORT jstring JNICALL Java_com_example_lua_MainSurfaceView_cmain
(JNIEnv *env, jobject thiz, jstring path)
{
    /* Lua ステートを作成する。 */
    lua_State *L = (lua_State*)luaL_newstate();
    luaL_openlibs(L);

    /* Lua コードを実行する。 */
    jstring jpath = (*env)->GetStringUTFChars(env, path, 0); /* ファイルパスを取り出す。*/
    int res = luaL_dofile(L, jpath);
    (*env)->ReleaseStringUTFChars(env, path, jpath);

    /* エラーの場合、エラーメッセージを返す。*/
    if (res != 0) return (*env)->NewStringUTF(env, lua_tostring(L, -1));
    return (*env)->NewStringUTF(env, "OK");
}
```

これで Java 側からファイルパスを指定して cmain 関数を呼び出すことで、Lua のコードが動かせるようになりました。

ただし、まだ肝心の Lua で書かれたファイルを読み込むことができません。Android では、Java 側で assets 内に置かれたファイルを読み込むことができますが、C 言語側では assets 内のファイルを読み込むことができません。そこで、アプリが起動したときに、assets/Lua 内に置かれたファイルを C 言語側で読み書きできる/data/data/パッケージ名/files 内にコピーするようにします。

MainSurfaceView.java

```
…
public class MainSurfaceView extends SurfaceView
implements SurfaceHolder.Callback
{
    …
    public MainSurfaceView(Context context)
    {
        …
        /* assets/Lua 内のファイルをすべて/data/data/パッケージ名/files 内にコピーする。 */
        AssetManager am = context.getResources().getAssets();
        byte[] buffer = new byte[1024];
        int n = 0;
        try
        {
            String[] list = am.list("Lua");
            for (String file : list)
            {
                InputStream is = am.open("Lua/" + file);
                FileOutputStream fos = context.openFileOutput(file, Context.MODE_PRIVATE);
                while ((n = is.read(buffer)) != -1) fos.write(buffer, 0, n);
                fos.close();
                is.close();
            }
        } catch (IOException e) {}
        …
    }
    …
}
```

これで、アプリ起動時に assets/Lua 内のファイルが/data/data/パッケージ名/files 内にコピーされるようになりました。

では、実際に、Lua のファイルが実行できるか試してみます。しかし、現状では Lua の実行結果を見える形で出力することができないので、ここでは assert 関数を使って無理矢理、結果を返しています。

assets/Lua/main.lua

```
local sum = 0
for i = 1, 10 do sum = sum + i end
assert(false, sum)
```

MainSurfaceView.java

```
…
public class MainSurfaceView extends SurfaceView
implements SurfaceHolder.Callback
{
    …
    @Override
    public void surfaceCreated(SurfaceHolder holder)
    {
        Log.d("Lua", cmain("/data/data/" + context.getPackageName() + "/files/main.lua"));
    }
}
```

実行結果（LogCat）

```
/data/data/com.example.lua/files/main.lua:3: 55
```

1.4 Lua から Java の関数を実行する

前節では、Java から Lua のコードを実行できるようにしました。今度は、実行した Lua のコードから Java の関数を呼び出せるようにします。これによって、Lua でコードを書いた Android アプリを作ることができるようになります。ここでは、画面上に文字列を描画する関数を作成して、Lua 上から呼び出せるようにします（JNI を使って、C 言語から Java の関数を呼び出す方法は [3] を参考にしました）。

まずは、C 言語から Java の関数を呼び出せるようにします。

MainSurfaceView.java

```
…
public class MainSurfaceView extends SurfaceView
implements SurfaceHolder.Callback
{
    …
    /* 文字列を画面に描画する。 */
    public void drawString(String s)
    {
        Canvas canvas = holder.lockCanvas();
        Paint paint = new Paint(Paint.ANTI_ALIAS_FLAG);
        paint.setTextSize(30);
        paint.setColor(Color.WHITE);
        canvas.drawText(s, 50, 50, paint);
        holder.unlockCanvasAndPost(canvas);
    }
}
```

lua-jni.c

```
...
JNIEXPORT jstring JNICALL Java_com_example_lua_MainSurfaceView_cmain
(JNIEnv *env, jobject thiz, jstring path)
{
    /* Java の関数を C 言語から呼び出せるようにする。 */
    Env = env;
    Thiz = thiz;
    jclass jcls = (*Env)->GetObjectClass(Env, Thiz);
    jdrawString = (*Env)->GetMethodID(Env, jcls, "drawString", "(Ljava/lang/String;)V");
    ...
}
```

これで、(*Env)->CallVoidMethod(Env, Thiz, idrawString, 出力文字列); を実行することで、C 言語から Java の関数を呼び出せるようになりました。

次に、C 言語で呼び出せるようになった関数を Lua で呼び出せるようにします。Lua では、スタックを経由して、あらかじめ登録した C 言語の関数を呼び出すことができる [4] ので、それを利用します。

lua-jni.c

```
...
int drawString(lua_State* L)
{
    jstring s = (*Env)->NewStringUTF(Env, lua_tostring(L, 1)); /* 文字列をスタックから取り出す */
    (*Env)->CallVoidMethod(Env, Thiz, jdrawString, s); /* Java の関数を呼び出す。*/
    (*Env)->DeleteLocalRef(Env, s);
    lua_settop(L, 0);
    return 0;
}

JNIEXPORT jstring JNICALL Java_com_example_lua_MainSurfaceView_cmain
(JNIEnv *env, jobject thiz, jstring path)
{
...
    /* Lua ステートを作成する。 */
    lua_State *L = (lua_State*)luaL_newstate();
    luaL_openlibs(L);
    lua_register(L, "drawString", drawString); /* C 言語の関数を登録する。*/
...
}
```

これで完成です。早速、実行してみましょう（図 1.2）。

assets/Lua/main.lua

```
drawString("Hello, world!")
```

1.5 おわりに

本記事では、Android SDK から Lua のコードを実行して、さらに Lua のコードから Java の関数を実行する方法を説明しました。これを駆使すれば、Lua だけで書いた Android アプリが作れるわけですが、実際に作ろうと思うと、アク

図 1.2: 実行画面

ティビティのライフサイクルに関する処理をどうするかとか、いろんな障壁にぶち当たったりして、現在、試行錯誤中です。今後は、（うまくいけば）実際に Android アプリを Lua で作って公開を考えています。

参考文献

[1] 技術メモ（仮）: Lua 5.2.0 on Android (1)
http://retrocatsoft.blogspot.jp/2012/02/lua-520-on-android-1.html

[2] 自堕落なぺぇじ : Android SDK assets の内容を全てローカルにコピーする
http://d.hatena.ne.jp/corrupt/20110203/1296695472

[3] JNI Java VM メモ : 他言語から Java 実行
http://www.ne.jp/asahi/hishidama/home/tech/java/jni_vm.html

[4] Roberto Ierusalimschy(著)，新丈 径 (翻訳)，Programming in Lua プログラミング言語 Lua 公式解説書，アスキー・メディアワークス，2009.

2 iPhoneApp開発ファーストステップ Xcode4.5 iOS6対応

造形工学課程 3 回生 山家 一晟

2.1 はじめに

2009 年頃からスマホ元年などと話題になった iPhone を初めとする、スマートフォン。ボディの大部分をディスプレイが占め、ほとんどの操作をタッチスクリーンで行う事が特徴的です。ここでは、初学者が Objective-C について深く知る前の、まずは開発環境に触ってみる方にアプリケーションとその開発方法について紹介します。

（しかし、紙面の都合上、環境導入から話し始めると非常に長くなっていまい収まらないので、Xcode のインストール等の環境整備の話は省きます。前提として、AppStore から Xcode を入手し、起動できる状態であるとします。）

2.2 iPhone シミュレータに hello world!!を表示させる

Xcode を起動すると図 2.1 のようなポップアップメニューが出ます。指示に従い、新規プロジェクトを作成します。

図 2.1: 新規プロジェクト作成画面

簡単な App 作り。今回はシンプルなものを組むので、図 2.2 のように SingleView Application を選択します。

図 2.3 のようにテキストボックスに入力します。ProductName は、プロジェクトの名前なので hello!にしてみました。OrganizationName は、自分の名前を入力して下さい。CompanyIdentifier は会社名ですが、今回は適当な名前を入れています。

ポップアップ画面が消え、図 2.4 のような状態になったと思います。App の iOS バージョンへの対応状況、起動画面、アイコンなどはこの画面で設定します。インターフェイス部分の組み上げは、図 2.5 のように左カラムにあるリストの StoryBoard で行います。

まずはプログラムを実行してみましょう。図 2.4 左上の Run のボタンを押せば iPhone シミュレータが起動し、真っ白な画面が広がる筈です。このように、何も書かなくても既に iPhone 上で動く為に最低限必要なコードはテンプレートと

図 2.2: SingleView Application を作成

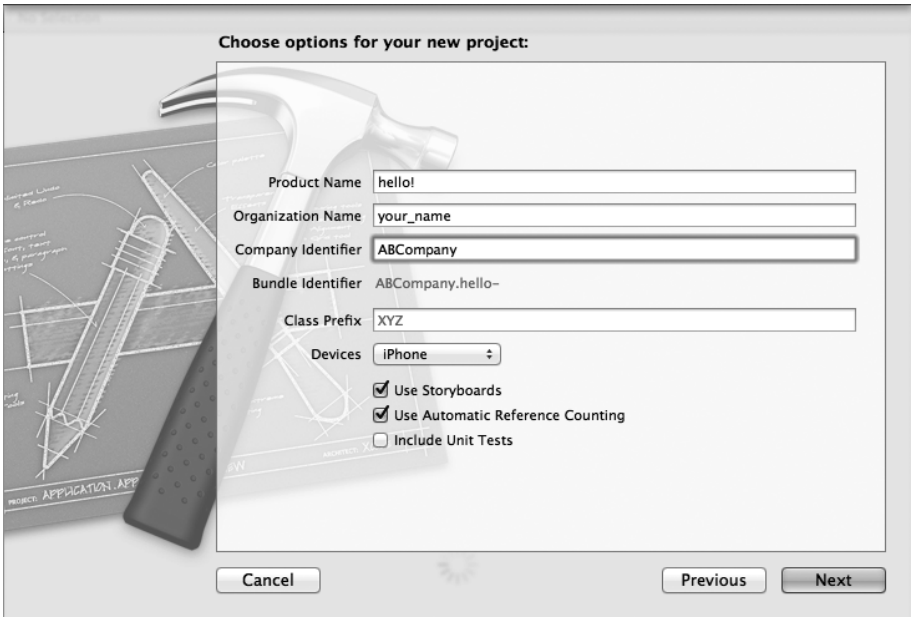

図 2.3: 新規プロジェクトの設定

して最初から用意されているのです。

では、せっかくですので、何か表示させてみましょう。図 2.6 に手順画面を示します。MainStoryboard を選択してください。画面中央の Edit は図 2.7 のようになります。図 2.6 の数字の順序に従い

1. MainStoryBoard を選択
2. 右下カラムを開く。
3. TextLabel を探し、中央の白紙の画面にドラッグ＆ドロップ。
4. TextLabel をダブルクリックし編集モードにしてから “HelloWorld!!” と入力。位置を任意の場所に整える。
5. 右上カラムの一番上のタブで右から 3 つ目のタブを開き、テキストのインスペクタを設定。

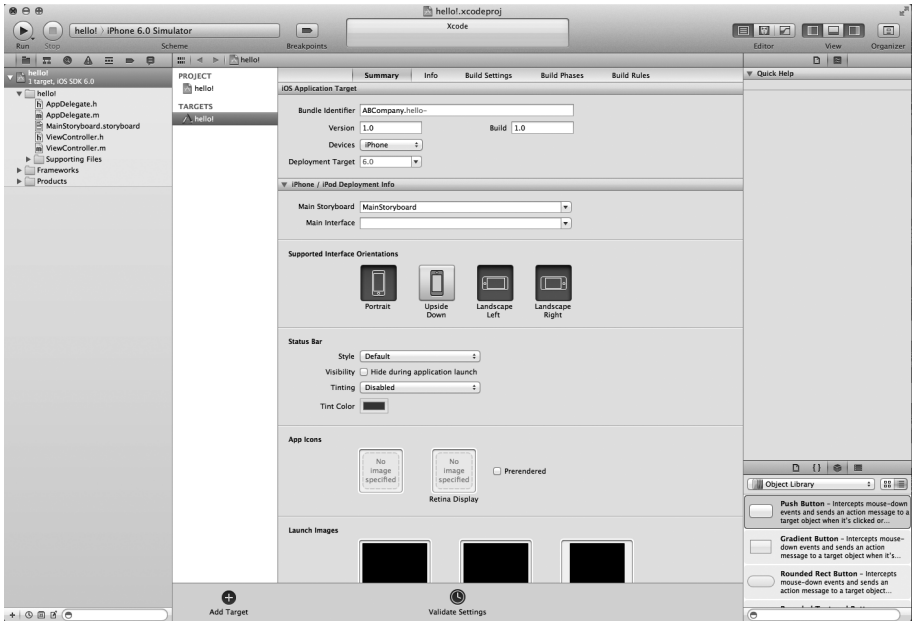

図 2.4: プロパティ画面

図 2.5: 図 2.4 左のカラム

再度図 2.4 左上の Run を押してみてください。どうでしょうか、図 2.7 のように “HelloWorld!!” の文字が表示されたと思います。

画面上にオブジェクトを配置するだけならこのようにコードを全くいじらなくてもアプリケーションが出来てしまうのです。

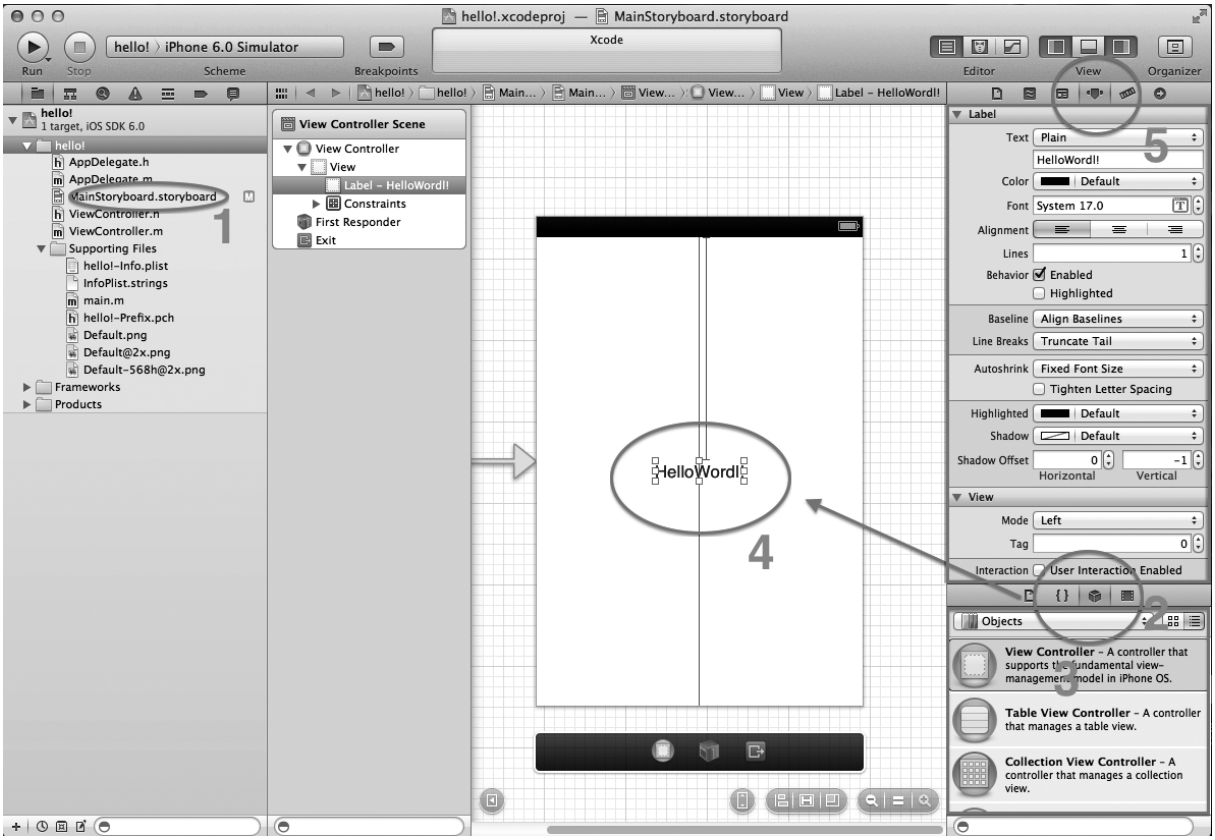

図 2.6: 手順画面

図 2.7: “HelloWorld!!” 実行画面

2.3 ボタンを押すとランダムに名言を切り替えるアプリ

ボタンを使ったアプリケーションを作成しましょう。ボタンを押せば中央の文字が変化するアプリケーションを制作します。まず、図 2.8 のような画像を用意します。形式は.jpg か.png を使用します。今回はとりあえず sky.png として説明します。

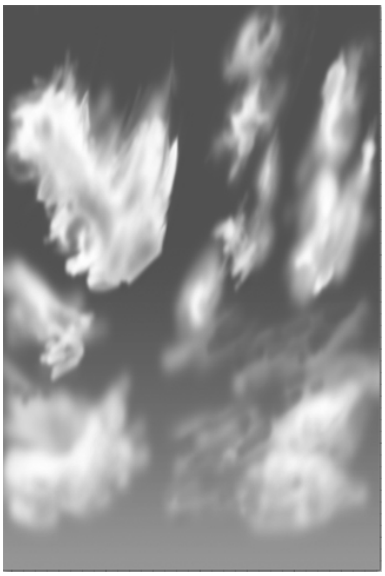

図 2.8: sky.png 960×640

図 2.6 左側にある Supporting Files にドラッグ&ドロップします。すると図 2.9 のようにポップアップされるので Destination にチェックを入れて Finish を押します。

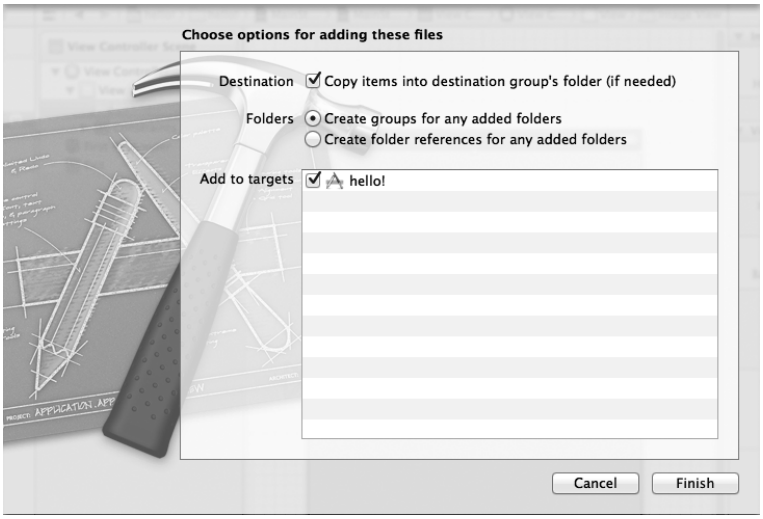

図 2.9: 画像移動時に表示されたポップアップ

次に、MainStoryBoard を開きます。

HelloWorld!!のラベルを削除します。

ここで、画面上のステータスバー（上の電波と時間とバッテリーを表示している部分）が邪魔なので消しておきましょう。左カラムの一番上の “hello!” を選択してください。すると中央に図 2.10 が出てくるので赤丸をした hide during application launch にチェックします。MainStoryBoard に戻ります。図 2.6 右上カラムの図 2.11 の部分を選択。Status Bar のプルダウンメニューから None を選択します。画面の部分からステータスバーが消えましたね。

では今度は、図 2.6 右下のカラムから図 2.12 の枠の中で Image View を探して、それを中央の画面にドラッグ&ドロッ

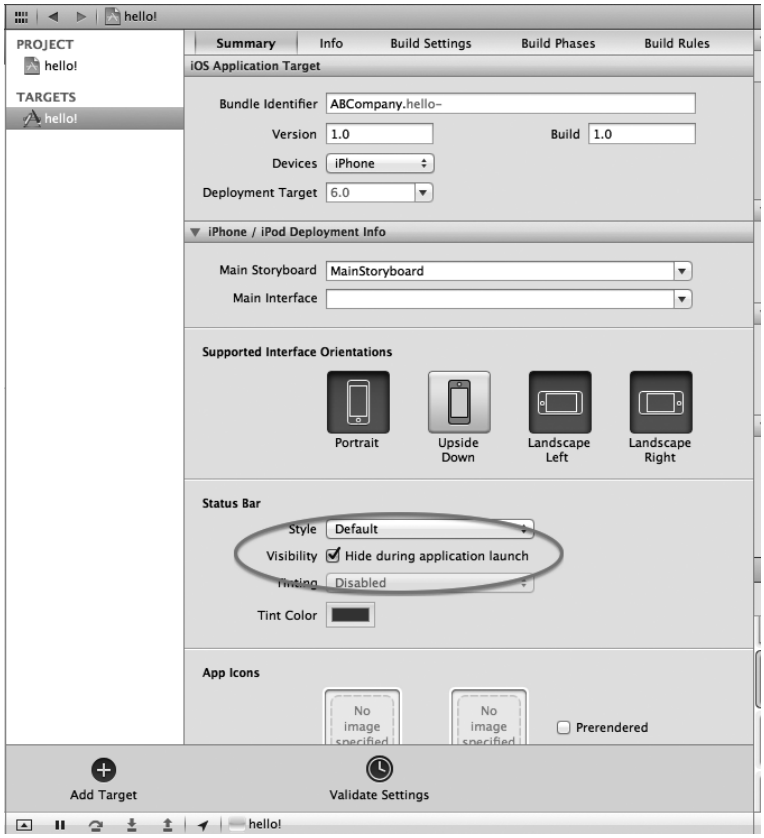

図 2.10: “hello!” 選択時の画面

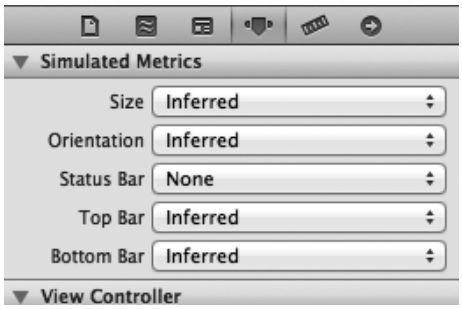

図 2.11: 図 2.6 右上のカラム

プしてください。配置した View を画面全体を覆うように調整します。左上のカラム、右から 3 つめのタブ（インスペクタタブ）を開いて Image View/Image のプルダウンメニューから sky.png を選択。今回は画像サイズが 960×640 で画面と同じアスペクト比の為歪みませんが、一応アスペクト比の設定をしておきます。View/Mode から Scale to Fill を選択します。

ImageView と同じように RoundRect Button を任意の場所にドラッグ＆ドロップしてください。ボタンをダブルクリックして文字を編集します。“今ここに新たな名言が誕生した。” と入力します。同じように TextLabel をドラッグ＆ドロップします。“猫も棒から落ちる” と入力します。図 2.13 のようになっている筈です。

図 2.12: 図 2.6 右下のカラム

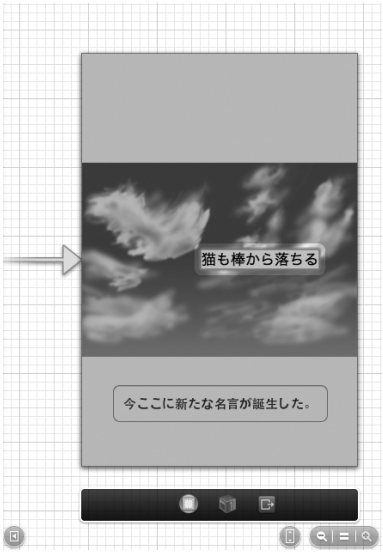

図 2.13: アプリ作成の途中経過画面

TextLabel を見やすいように少し編集しましょう。図 2.6 右上のカラムのインスペクタタブを開いてください。図 2.14 のように設定します。Color、Fonr、Alignment、Shadow、Shadow Offset を調整しました。

では、画面上の部品とプログラムを繋ぎます。図 2.6 右上にある、図 2.15 のボタンから真ん中のボタンを選択します。ViewController.h が表示されている事を確認します。中央の画面上のイメージを Ctrl + 右クリックでクリックしたまま図 2.16 のようにカーソルを移動し、@interface ... の下の部分に移動し離します。 図 2.17 のようなポップアップ画面が出てくるので、図のように Name を Sky に設定します。次に TextLabel を同様に Ctrl + クリックして先の下に配置します。同様に Name を word に設定します。そして同じく RoundRectButton を下に配置しますが、今度は Name だけでなく Connection も設定します。Name を Change、Connection を Action にします。これらを行うと図 2.17 のように記述されるはずです。

図 2.18 のように記述されている筈です。

では、今度は左カラムより ViewController.m を選択します。図 2.19 のように入力します。

図 2.19 の赤枠の部分を編集します。-(IBAction) の部分が RoundRectButton をボタンとして追加した為に ViewController.m に追加された関数です。3 つの文字列を用意して、ランダム関数を用いて 1 つを選び self.wrod.text に設定します。また、デバイスを回転させても画面が回転しないように設定します。

最後にデバイスを回転させても画面が回転しないように固定します。

以上で終了です。Run（実行）してみましょう。図 2.20 が実行画面です。画面上のボタンで名言が変化しているでしょうか？シミュレータでデバイスの回転を行っても画面は変化しない筈です。

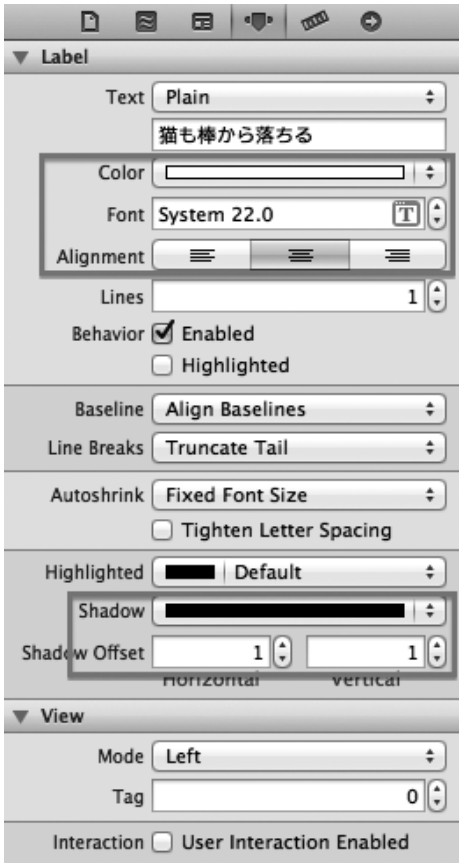

図 2.14: TextLabel の設定

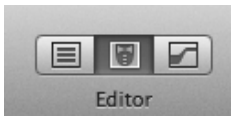

図 2.15: Editor ボタン

2.4 終わりに

いかがでしたでしょうか？UI とプログラムを繋いだ簡単な iOSApplication 制作は。非常に楽しめたなら幸いです。興味が湧いたのなら、ぜひネットや書籍などを調べて好きなアプリケーションを作成してください！

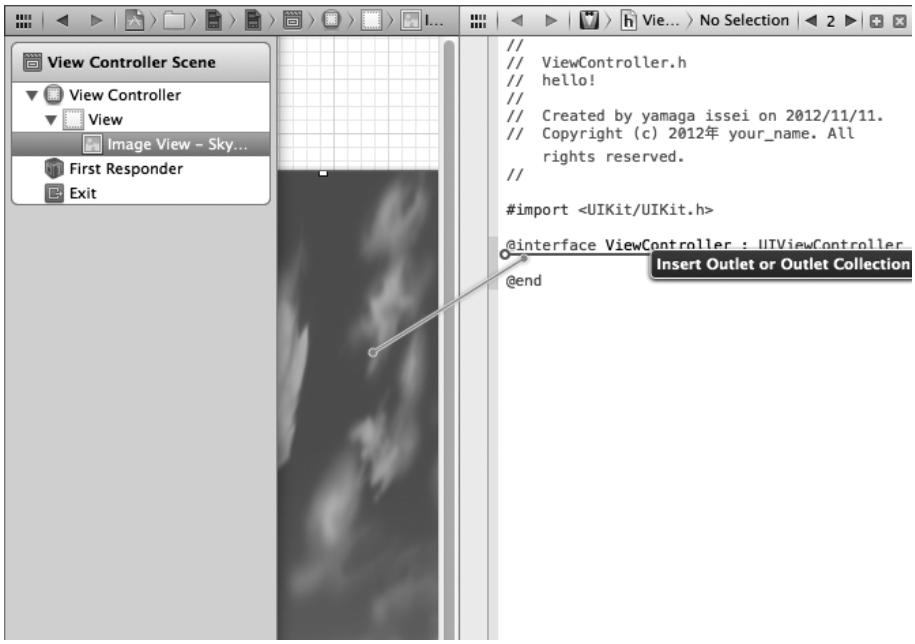

図 2.16: 部品とプログラムを繋ぐ手順画面

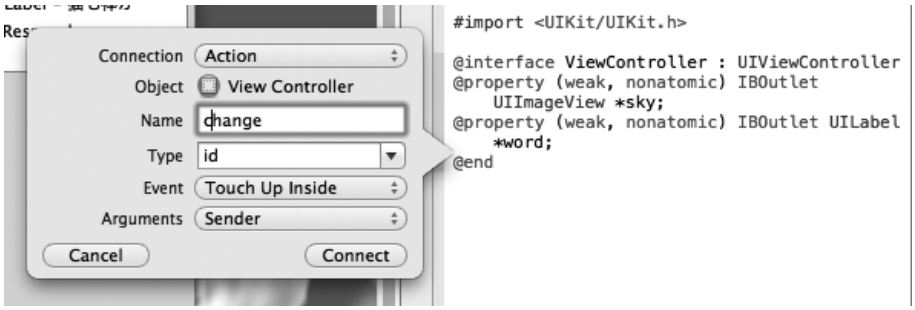

図 2.17: RoundRectButton のポップアップ

```
#import <UIKit/UIKit.h>

@interface ViewController :
    UIViewController
@property (weak, nonatomic) IBOutlet
    UIImageView *sky;
@property (weak, nonatomic) IBOutlet
    UILabel *word;
- (IBAction)change:(id)sender;
@end
```

図 2.18: RoundRectButton 編集時のソースコード

```
#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the
        view, typically from a nib.
}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be
        recreated.
}

- (IBAction)change:(id)sender {
    NSArray *theWord = @[@"猫も棒から落ちる",@"こい
        つ馬鹿だ！",@"んまっ　つぁ　ちょぎっ！"];
    int r = arc4random() % 3;
    self.word.text = [theWord objectAtIndex:r];
}

-(BOOL)shouldAutorotate {
    return NO;
}

@end
```

図 2.19: 名言入力例

図 2.20: 実行画面

参考文献

[1] 森巧尚 (著), やさしくはじめる iPhone アプリ開発の学校 マイナビ

3 赤外線LEDを用いたラジコンの製作

電子システム工学課程 3 回生 松本 駿
情報工学課程 3 回生 山田 晃久

3.1 概要

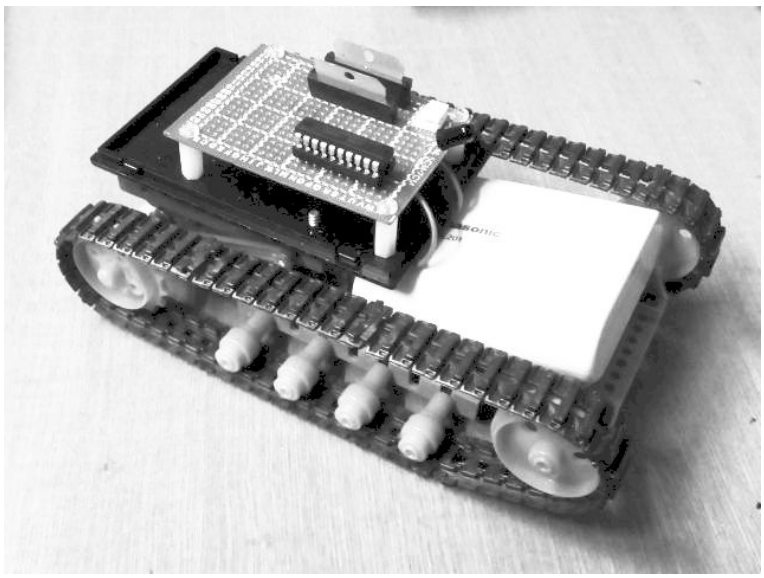

図 3.1: 製作途中の本体写真

赤外線 LED とフォトトランジスタを使って無線で通信し、その信号処理をマイコンで行うことでブルドーザーを動かそう、という計画です。要するにお手製のラジコン。ただし、作業量を考えてお手製とはいいつつも一般の有線制御式のブルドーザーキットを購入し、機体本体とコントローラはここから流用することにしました。製作途中の本体写真が図 3.1 です。

基本的な概念図は図 3.2 のとおり。構想段階のものなので一応他機体との連携も考慮していましたが時間的に実装が難しいため残念ながら見送りです。

送受信どちらの回路にもマイコン (AVR ATTiny2313) を使用しており、その駆動用に DC5V が必要だったのでスマートフォンなどで用いる USB 接続タイプのモバイルバッテリを電源として採用しています。値段がやや高めですが最初から 5V 出力で容量も多めなので使い勝手がよく、重宝しています。

3.2 ブルドーザー

今回のラジコン工作で一番注力したのは赤外線を用いた信号処理で、ブルドーザー自体はそれほど手間をかけていません。タミヤの工作シリーズよりギアボックスとブルドーザーのついたキットを選んで使っています。元のキットは本体を 2m ほどの導線を介してコントローラに繋ぎ、制御するものなのですが、この有線部分を赤外線 LED とフォトトランジスタに置き換えて無線化する工作を行いました。ブルドーザー本体に配置された基板から飛び出す形でフォトトランジスタが配置され、ここに向けてコントローラから赤外線を照射します。

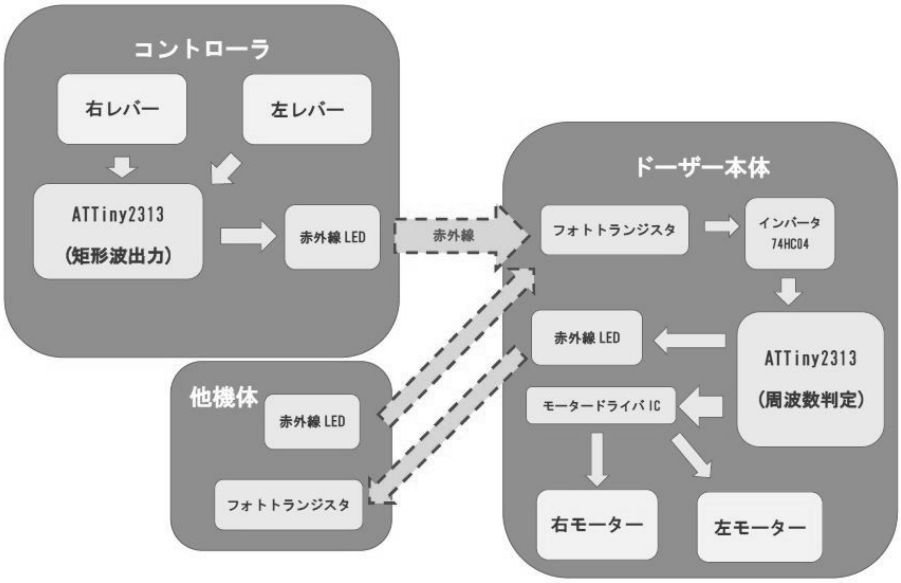

図 3.2: 動作概念図

3.3 送受信部

このラジコンの肝となる無線送受信には、波長 940nm の赤外線を使いました。コントローラ側ではレバーの操作に応じて LED に矩形波を入力し、本体側では受光部からの信号から矩形波の周波数を読み取ってモータドライバ IC を駆動させています。モーターは左右に 1 つずつ搭載し前進後退の他に通常の旋回や、その場で旋回するいわゆる超信地旋回のような回転運動も可能です。ただし、履帯がすぐに取れてしまうのであまり派手な機動を行うことはできません。

コントローラのレバーと周波数と各モーターの動作、全体の動作を表 3.1 にまとめました。

表 3.1: 各動作と周波数対応表

右レバー	左レバー	周波数 [Hz]	右モーター	左モーター	全体動作
初期位置	初期位置	無し	停止	停止	停止
前	前	1000	正転	正転	前進
後	後	2000	逆転	逆転	後退
前	初期位置	3000	正転	停止	右信地旋回
初期位置	前	4000	停止	正転	左信地旋回
後	初期位置	5000	逆転	停止	右逆信地旋回
初期位置	後	6000	停止	逆転	左逆信地旋回
前	後	7000	正転	逆転	右超信地旋回
後	前	8000	逆転	正転	左超信地旋回

3.3.1 送信回路

送信回路ではコントローラの左右のレバーから入力信号を受け取り、上記の表で定められた矩形波を LED に出力します。図 3.3 がその回路図です。

図 3.3 の回路図上ではレバーをスイッチで表し、接点が 2 つのものを使っていますが実際のレバーには、前、後の他に表 3.1 にあるように回路的には何も反応しない初期位置が存在します (ON-OFF-ON スイッチ)。左右共に初期位置で本体は停止です。また、回路図にはありませんがモバイルバッテリ自体の電源スイッチが主電源のスイッチです。

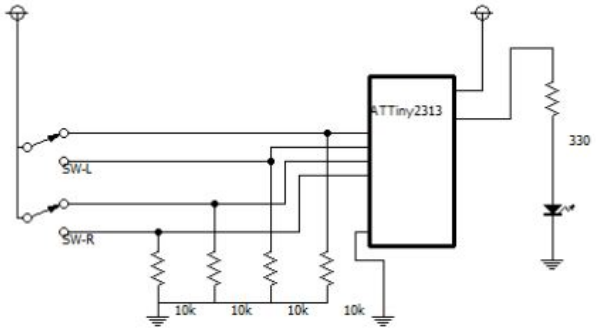

図 3.3: 送信部回路図

3.3.2 受信回路

受信回路では、まずフォトトランジスタが赤外線を受け、矩形波信号を発生させます。次にその矩形波信号をインバータを通してマイコンに入力し、その周波数を判別します。周波数を判別した後、表 1 にしたがってモータドライバ IC TA7291P に 2bit の信号を送ってモータの正転逆転を制御します。周波数判別については次章にて。回路図を図 3.4 に示します。

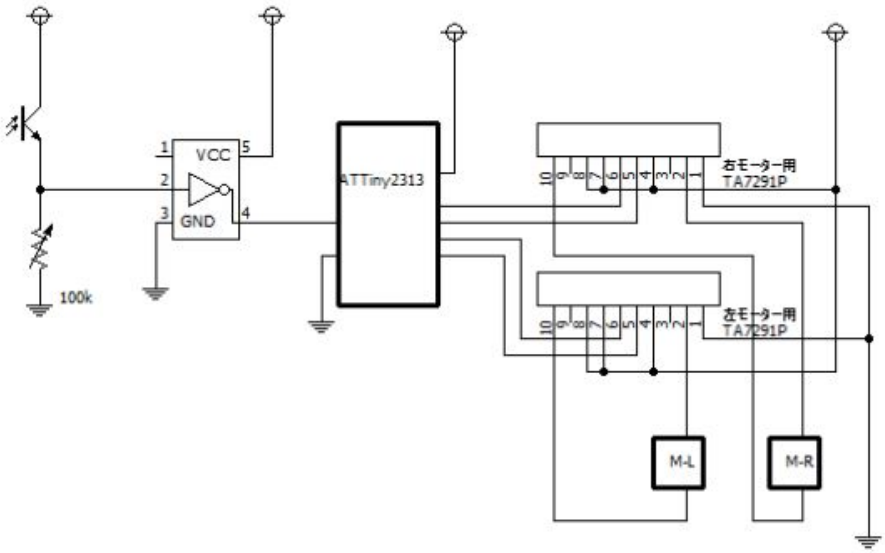

図 3.4: 受信部回路図

回路図中の M-L,M-R がそれぞれ左右のモータに相当します。TA7291P には IC 自体の動作用電源とは別にモータへ電源供給を行うためのピンがあり、速度を抑えるために 3.3V に落とした電圧を印加することも可能です。使う電圧はまだ確定させていませんが、動作試験時は 5V をそのままモータ電源としていました。また、TA7291P から制御できるモータの動作には正転、逆転、動作なしの 3 状態の他にブレーキも備えていますが今回ブレーキを明示的に行うにはコントローラのボタン数が足りず、ブレーキ搭載の必要性も感じなかったため使用していません。

3.4 マイコンプログラム

3.4.1 送信側プログラム

送信側のプログラムは、コントローラから入力信号を受け取り、図 3.3 の表で定められた周波数の矩形波を LED に出力します。メイン関数は次のようになります。

```
int main(){
    DDRB = 0x00;
    DDRD = 0b00000001;

    TCCR0A = 0b00000010; //CTC 動作

    //20kHz
    TCCR0B = 0x02;        // 1/8 に分周
    OCR0A  = 50;          //TOP 値を 50 に設定

    TIMSK = 0b00000001;  //比較 A 一致割り込み有効

    sei();                // 全体の割り込み許可

    while(1){}
}
```

メイン関数では初期設定を行い、割込みを許可した後に while 文で割り込みを待ち続けます。ベースクロックは 8MHz で、それを 1/8 に分周しています。なお、0b から始まる 2 進リテラルは GCC 拡張です。初期設定で用いているレジスタと設定内容の関係を表 3.2 にまとめました。

表 3.2: レジスタと設定内容の対応表

レジスタ	設定内容
DDRB	ポート B の入出力の方向設定
DDRD	ポート D の入出力の方向設定
TCCR0A	タイマ 0 のモード設定
TCCR0B	タイマ 0 の TOP 設定とクロック設定
TIMSK	割り込み設定

なお、OCR0A では TOP 値を設定しています。

次に割り込みで呼び出される関数のプログラムは次のようになります。

```
/*割り込み*/
ISR(TIMER0_COMPA_vect){
    static char cnt = 0;
    char sw = 0;

    sw = PINB;

    //1kHz
    if((sw & 0b11110000) == 0b10100000){
        OCR0A  = 50;
        cnt++;
        if(cnt >= 10){
            PORTD ^= 0b00000001;
            cnt = 0;
        }
    }
```

```
    //2kHz
    else if((sw & 0b11110000) == 0b01010000){
        OCR0A  = 50;
        cnt++;
        if(cnt >= 5){
            PORTD ^= 0b00000001;
            cnt = 0;
        }
    }

    //3kHz
    else if((sw & 0b11110000) == 0b10000000){
        OCR0A  = 28;
        cnt++;
        if(cnt >= 6){
            PORTD ^= 0b00000001;
            cnt = 0;
        }
    }

    //4kHz
    else if((sw & 0b11110000) == 0b00100000){
        OCR0A  = 62;
        cnt++;
        if(cnt >= 2){
            PORTD ^= 0b00000001;
            cnt = 0;
        }
    }

    //5kHz
    else if((sw & 0b11110000) == 0b01000000){
        OCR0A  = 50;
        cnt++;
        if(cnt >= 2){
            PORTD ^= 0b00000001;
            cnt = 0;
        }
    }

    //6kHz
    else if((sw & 0b11110000) == 0b00010000){
        OCR0A  = 28;
        cnt++;
        if(cnt >= 3){
            PORTD ^= 0b00000001;
            cnt = 0;
        }
    }
```

```
    //7kHz
    else if((sw & 0b11110000) == 0b10010000){
        OCR0A  = 71;
        PORTD ^= 0b00000001;
    }

    //8kHz
    else if((sw & 0b11110000) == 0b01100000){
        OCR0A  = 62;
        PORTD ^= 0b00000001;
    }
    else PORTD &= 0b11111110;
}
```

まずコントローラからの入力信号を受け取り、それによって比較一致の TOP 値を設定します。それぞれの処理の実行に必要な割り込みの発生回数をあらかじめ決めており、出力したい周波数の倍の周波数でそれぞれの処理が実行されます。そこで LED の ON,OFF を切り替えているので、定められた周波数の矩形波を LED から出力します。8MHz を 1/8 しているので、タイマ割り込み自体は 1MHz で発生しています。今回は使っていませんが、10kHz の矩形波を出力する場合、OCR0A の値を 50 にすると割り込み関数は 50 回に 1 回呼ばれるので、20kHz で割り込み関数が呼ばれ、それが 2 回呼ばれると出力がフリップします。2 回フリップすると 1 周期なので、10kHz の周期を出力できます。また、if 文の条件部分の `(sw & 0b11110000)` では、ビット演算子の`&`を用いて入力ポートの上位 4 ビットのみを取り出しています。

3.4.2 受信側プログラム

受信側のプログラムはマイコンに入力された矩形波の周波数を判別し、それに対応した信号を出力します。メイン関数は次のようになります。

```
int main(){
    DDRB = 0b11110000;
    DDRD = 0b00000000;

    TCCR0A = 0b00000010; //CTC 動作

    //40kHz
    TCCR0B = 0x02;        // 1/8 に分周
    OCR0A  = 25;          //TOP 値を 25 に設定

    TIMSK = 0b00000001;  //比較 A 一致割り込み有効

    sei();                // 全体の割り込み許可

    while(1){}
}
```

メイン関数は送信側プログラムとほとんど同じですが、入力された信号を 40kHz でサンプリングするために割り込み発生の周波数を 40kHz にしています。

次に割り込みで呼び出される部分のプログラムは次のようになります。

```
/*割り込み*/
ISR(TIMER0_COMPA_vect){
```

```
    static int cnt = 0, i = 0, t;
    static char in, p_in;
    static char b_out[9] = {
         0b00000000,
         0b10100000,
         0b01010000,
         0b10000000,
         0b00100000,
         0b01000000,
         0b00010000,
         0b10010000,
         0b01010000};

    in = PIND;
    if((in & 0b0000001) != (p_in & 0b0000001)) cnt++;
    p_in = in;

    if(i >= 500){
        t = (cnt+12)/25;
        if(t>8) t = 8;
        PORTB = b_out[t];
        i = 0;
        cnt = 0;
    }
    i++;
}
```

まず入力信号を受け取り、前回に割り込みが発生したときと入力が違っていた場合に cnt の値を+1 します。そして割り込みが 500 回発生するごとに、つまり 12.5ms ごとに入力信号の周波数を判断し、それに対応した信号を出力します。また、`t = (cnt+12)/25;` の部分では t に商のみが代入され、cnt の数に読み取り誤差の幅を持たせ、周波数判定に利用しています。

3.5 終わりに

赤外線を用いて信号伝送をやってみたい、でも作るものがないという状態から始めて、どうせならラジコンにしようと思いつきソフトウェアが得意な部員とハードウェアが得意な部員とで共同で始めた計画でした。

実はこの原稿執筆時点ではせいぜい 1m 程度の距離しか送受信に成功しておらず、実用的な距離での信号伝送は実現していません。送受信の LED と受光部の到着待ち段階です。この冊子が配布されることになる学祭当日には完成品をお見せします……多分。 松本

マイコンを用いて何かを作ってみたい、でも何を作っていいのかわからないという状態から始めて、どうせならラジコンにしようと思いつきソフトウェアがこれといって得意でない部員とハードウェアが得意な部員とで共同で始めた計画でした。

実はこの原稿執筆時点ではまだ必要な部品が届いておらず、実機でのテストはしていません。一応マイコンのみでの送受信のテストは成功しているので、この冊子が配布されることになる学祭当日に完成品をお見せできなくても私の責任ではない……はず。 山田

参考文献

[1] AVR-千秋ゼミ

http://www-ice.yamagata-cit.ac.jp/ken/senshu/sitedev/

[2] 東芝セミコンダクター&ストレージ社 TA7291P データシート

http://www.semicon.toshiba.co.jp/

4 JavaScript を用いたちょっとしたゲーム開発 (クライアントサイド編)

情報工学課程 2 回生 永徳 泰明

4.1 はじめに

この記事では私が JavaScript(CoffeeScript) を使って麻雀対戦ゲームを開発したことにまつわることについて書こうと思います。

ただし、麻雀ゲームはゲームロジックの実装部分がどうしても長くなり (牌の待ち上がり判定、点数計算等部分だけで現在 804 行)、それはそれで面白いのですが、少なくとも麻雀のルールがわからない人にはキルミーベイベーのアニメを見るより退屈な代物にしかならないと思われるので避けて話します。

私は開発していく中で、プログラムがうまく動かない、あるいは思った通りの動作をしなかったとき、ブラウザ上で JavaScript プログラムがどのようにして (How) 動くのだろうというところに疑問を持ちました。ここでは初学者にもなるべくわかるようにプログラムがどのようにして動くのだろうという点について解説していきます。

4.2 どうして JavaScript で開発しようと思ったか

まず、解説ではないのですが、どうして JavaScript で開発しようと思ったかについて書きます。

私は去年、Ruby[1] を使ってシューティングゲームを作っていたのですが、ある問題を抱えていました。プログラムを実行するのに恐ろしく手間がかかるのです。

1. Ruby 本体をインストールする (もちろん Windows、Linux でインストール方法は違う) のに 5〜15 分
2. Ruby 用のライブラリをコンパイルするための gcc の opengl ライブラリを入れる
3. Ruby 用のライブラリを入れる (ruby-opengl)
4. ようやく本体を実行する

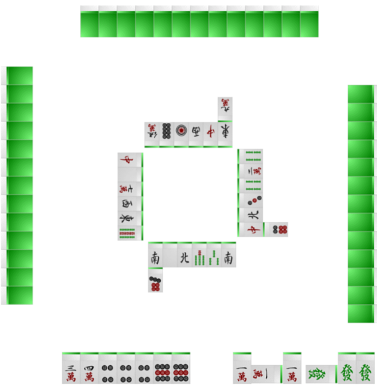

図 4.1: ゲーム画面

[1] オブジェクト指向スクリプト言語

と、Ruby のゲームプログラムを走らせるにはしんどい手順を踏む必要がありました。(開発する PC だけでなく動かす PC でさえも)

これがバイナリアプリケーションならば

- 本体をインストールする
- 必要ならライブラリを入れる

の 2 手順で済むのですが、Ruby がインタプリタ方式で動くプログラミング言語である以上描画機能を拡張するには Ruby 本体に描画 API の bindings を入れるところから始めなくてはならず手間がかかりました。

ここで、

- 開発が簡単
- どこでも動作する

という条件を満たす環境・言語を探していたところ、ウェブブラウザ上で簡単に動かすことのできる JavaScript が立ち上がってきました。

JavaScript はウェブブラウザがそのまま実行環境、開発環境となります。さらに、WebGL や Canvas などの HTML5 の要素と組み合わせることにより普通のアプリケーションと比べて見劣りしないほどの描画機能を持ちます。買いたての PC でもスマホでも Web ページを開けば実行することができます。カップラーメンもびっくりのお手軽さです。さっそく私は Ruby を捨てて、JS プログラミングを始めることにしました。

4.3 どのように JavaScript は実行されるか

どのようにして JavaScript(以下 JS) がブラウザ上で実行されるかについて軽く解説します。

最近の普通のブラウザの場合、読み込んだ html ファイル中の script 要素に記述した JS コード、JS ファイルがそれぞれ要素ごとにコンパイルされ、実行されます。実行の直前に自動的にコンパイルされるので、普通コンパイルのことを意識することはありません。

4.4 どのように JavaScript は画面を描画するか

どのようにして JS が画面に要素 (画像、線、四角…etc) を出力するかについて解説します。

JS で画面に何か出力させようとするとき、まず始めに行うのは DOM[2]や Canvas API[3]等の API(Application Programming Interface) を実行することです。

先の API の関数を呼ぶことで、ブラウザに命令を与えることができます。たとえば画像を表示するときには API の関数を通じてブラウザに <img src=”./画像のファイル名”> という要素を html の中に挿入させ、画像を表示させます。

SDL や DX ライブラリで作成した GUI プログラムとは違い、画面を更新するのはブラウザのお仕事です。

4.4.1 enchant.js

今回、ブラウザの API を操作するのに enchant.js[1] というライブラリを使用しました。enchant.js を使うことで、ライブラリの関数を通じて簡単に HTML を操作することができます。今流行りの jQuery も HTML 要素を操作するための[4](そして最も有名な) ライブラリです。

index.html と test.js、enchant.js、chara0.gif を同じディレクトリに入れて index.html をブラウザで開くと画像が表示されます。

ここで、game.rootScene.addChild(sprite) の行で sprite に対応した HTML 要素が作成され画面に表示されます。要素が作成されるというのは、enchant.js の中で DOM の API が呼び出されて

[2]JS 中から HTML を利用するためのインターフェース
[3]JS から直接画面を描画するための新しいインターフェース
[4]ただし、エフェクトやユーティリティなど他の機能も持っている

```
window.onload = function () {
  enchant();
  var game = new Game(320, 320);
  game.preload('chara0.png');
  game.onload = function() {
    var sprite = new Sprite(32, 32);
    sprite.image = game.assets['chara0.png'];
    game.rootScene.addChild(sprite);
  };
  game.start();
};
```

図 4.2: 画像を表示するプログラム (test.js)

```
<html>
<head>
    <script type="text/javascript" src="enchant.js"></script>
    <script type="text/javascript" src="test.js"></script>
</head>
<body>
</body>
</html>
```

図 4.3: test.js(と enchant.js) を読み込む index.html

```
<div style="position: absolute; width: 32px; height: 32px; overflow: hidden;
background-image: url(file:///C:/Users/enhant_mj/chara0.png);
left: 0px; top: 0px; background-position: 0px 0px; "></div>
```

という (chara0.png を表示する) 要素が index.html の中で作成される[5]ことです。

プログラムのほうに注目すると、HTML はソースの中に出てきていないにもかかわらず、sprite と game という変数にオブジェクトを代入して関数を呼び出すだけで HTML の操作ができているのが確認できます。

このように、enchant.js を通じて API を呼び出すことで実際は HTML を操作しているにもかかわらず、そのことをあまり意識することなくプログラムが書けるという利点がありました。また画像を追加する、画像を動かす、画像へのクリックを受け取る等の処理も API を間接的に呼び出すことで簡潔に書け使いやすく感じました。

4.5 どのようにプログラムモジュールが動いているか

ここまで、どのように JS が実行されているかという話、今回のプログラムではライブラリを通じて API を呼び出し、画面を描画しているという話をしました。

ここからはプログラム内部の、さきほどのライブラリを利用して自分が作ったゲームの話になります。自分の書いたプログラムが一行も Lime に載らないのも少し寂しいので、少しだけ解説します。

大まかに、自分の書いたプログラムは牌の役、待ち判定部分 (hai_check.js)、表示部分 (mj_view.js, layout.js)、ゲーム進行部分 (mj_model.js)(+ネットワーク部分) に分けて製作しました。

それぞれの機能の少し詳しい解説をすると、牌の役、待ち判定部分は役、待ちを判定する部分。ゲーム進行部分は、麻雀から表示部分を抜いた、手牌、河などのデータを持ち、プレイヤーが選択した行動に基づきデータを更新する部分。表示部分は、進行部分のイベントに従い画面を更新したり、アガリ画面を表示したり、ユーザーからの入力を受け取る部分です。

どのように動いているか、というと、それぞれのモジュール (に属するインスタンス) が、お互いの関数を呼び出すことによって動作しています。どういうふうにお互いを呼び出すのか、ということは 4.5.1 節に記述します。

設計において、ゲーム進行部分を表示部分に依存させない (表示部分への参照を直接は呼び出さない) ことを特に心がけました。

これによって全く同じプログラム (mj_model.js, hai_check.js) をサーバー側、クライアント側で使いまわすということが可能になりました。[6]

また、プログラムがモジュールごとに分かれていることにより、テスト、デバッグが容易になるといった利点もありました。

[5]デベロッパーツールを使って HTML ソースを表示させることによって確認できる。

[6]今回はクライアント側の話、ということでサーバー側の話はしませんでしたが、実はサーバー側のプログラムも作成中

```
class ModelBase
  constructor:->
    @listeners=[]
  add_listener:(listener)->
    @listeners.push(listener)
  notify:(action)->
    for i in @listeners
      i.update(@,action)
  remove_listener:(listener)->
    @listeners.splice(@listeners.indexOf(listener),1)

class ViewBase
  update:(subject,action)->
```

図 4.4: Observer パターンの実装

```
class Stage extends ModelBase
  ...
    @notify type:"tsumo",actor:@now_player,pai:@yama.shift()
  ...
```

図 4.5: ゲーム進行クラスの定義

4.5.1 Observer パターン

モジュールはバラバラのままでは動作しません。動かすには再度結合させる必要があります。

データ部分から表示部分を切り離してたとして、じゃあどうやってデータが更新したタイミングで表示するのか (例:ツモったときに牌を表示する)、という問題があります。更新をネットワーク通信部分、画面表示部分の両方に通知させるようにしなければならない可能性があり、また、できるだけ表示部分とゲームのメインロジックは切り離したい。という考えから、一対多の依存関係を定義するときによく用いられる Observer パターンを使ってこの部分を実装しました。

図はほぼ実際のコードの引用です。(言語は CoffeeScript[7])

図 4.4 が、ゲームのメインロジックと表示部分の基底クラスになります。図 4.5、4.6 がそれを継承したクラスで、ゲーム進行クラスがゲームで通知したい部分で notify を呼び出すと (立直、ツモ…etc)、add_listener で登録したクラス (表示部分) すべてに対して update が呼ばれます。

こうすることにより、将来ネットワークで (立直、ツモ…etc) などのイベントを送信しなければならなくなった場合にもゲームのロジックに手をいれることなく通知対象を増やすだけで対処できます。

オブジェクト間の一般的な設計構造を抽象化したいくつかのパターンをデザインパターンと呼び [2]、Observer パターンもその一種です。このようにしてパターンを使うことにより、ある程度の疎結合を保ったままプログラムを構築することができます。

4.6 おわりに

制作過程の報告というよりは解説がメインになり、わかりづらくなってしまったのではないか、これでよかったのだろうかと反省しています。ゲームについては、これを書いた時点ではどのモジュールも未完成で、どう見てもテンパイできてないのにたまに立直できたりとかアガリ画面ができてないとかドラだけで上がれたりとか宇宙麻雀を彷彿とさせる出来になっているのですが、(一見) きちんと動くゲームを作れたのはそれだけで嬉しいものがありました。

```
class StegeView extends ViewBase
  ...
  update:(subject,action)->
    switch action.type
      when "tsumo"
        ...(表示処理)
      when "dahai"
        ...
  ...
```

図 4.6: 表示クラスの定義

[7] JavaScript にコンパイルすることのできる言語、JavaScript より記述が容易

最後に、これをきっかけに JS プログラミングを始める人が一人でもいるなら、願ってもない幸せです。

参考文献

[1] Reference - enchant.js - HTML5 + JavaScript Game Engine http://enchantjs.com/ja/reference.html

[2] オブジェクト指向における再利用のためのデザインパターンエリック ガンマ Erich Gamma 著、ラルフ ジョンソン Ralph Johnson 著、リチャード ヘルム Richard Helm 著、ジョン ブリシディース John Vlissides 著、本位田真一訳、吉田和樹訳

5 暗号学

情報工学課程 2 回生 判田 瑞貴

5.1 暗号学とは何か

まず、暗号とは通信を行うときに第三者に通信文が見られても、内容が分からないようにする表記法のことである。
暗号学とは、暗号化や暗号解析に研究を行う学問である。
暗号化とは、暗号を作成する行為を指し、暗号解析とは、暗号の性質や強度を分析、研究する行為を指す。

5.2 現在使われている暗号技術は安全か?

現在、様々なところで暗号化技術が使われているが、その暗号化は「絶対的に安全」ではないが「実質的に安全」な方法が取られている。実質的に安全とは理論的には解読可能であっても、現実的な時間内に解読出来ない、秘密が守られる必要のある期間内は解読不能なものを指す。現時点では絶対に安全と保証できる暗号化方式は実用性に乏しいものであるため、計算量的に安全が保証された実質的に安全な暗号化方式が採用されている。

5.3 対称暗号

対称暗号の概要

対称暗号とは暗号化時と復号化時に同じ鍵を用いる方式である。一般的に非対称暗号（後述）より暗号化処理と復号化処理にかかる時間は高速だが、鍵の配布方法が難しい。例えばメールを暗号化処理を施して送ることを考える。暗号化時と復号化時に同じ鍵を用いるので、事前にメールの送信者は受信者に鍵を送らなければならない。鍵がばれてしまえばメールの内容が読むことが可能になるので、ネットワークを通して送る場合、当然鍵も暗号化して送らなければならない。つまりメールに施した暗号化処理のレベルは鍵に施した暗号化処理と同じレベルになってしまうのである。

ブロック暗号とストリーム暗号

対称暗号はブロック暗号とストリーム暗号に大別できる。
ブロック暗号は通常、64 ビットまたは 128 ビットの固定サイズブロックごとに暗号化処理を行う。ブロック暗号の代表として、DES、AES、Blowfish がある。
ストリーム暗号は通常、平文（暗号化処理を施すために入力される元のデータ）の 1 ビットまたは 1 バイトごとに擬似乱数のビット列を生成する。典型的には暗号化処理ではこのビット列と平文の排他的論理和を演算する。連続するデータストリームの暗号化する際に重宝される。ストリーム暗号の代表として、RC4、LSFR がある。

5.4 非対称暗号

非対称暗号の概要

非対称暗号では公開鍵と秘密鍵という 2 種類の鍵を使用する。公開鍵で暗号化された平文は秘密鍵によってのみ復号化される。そのため公開鍵は公開し、秘密鍵は秘密にすれば、対称暗号のように秘密にしなければならない鍵を通信する必要はなくなる。ただし、一般的に対称暗号に比べ、暗号化・復号化に時間がかかる。非対称暗号を実現するには一方向性を

持った関数が必要である。関数とは X の値が決まれば、Y の値も決まるというような対応関係を指す。一方向性を持った関数とは X から Y を求めることは容易だが、Y から X を求めることは困難というような性質を持つ関数のことである。

RSA 暗号の鍵の生成法と暗号化・復号化処理の方法

RSA 暗号の鍵を作る流れを説明する。まず自然数 N と互いに素 (2 つの値の最大公約数が 1 となること) となる 1 から $N-1$ までの数値の数を求める。これを $\Phi(N)$ と表すと、N が素数のとき $\Phi(N)=N-1$ となることが分かる。従って P と Q が素数のとき、$\Phi(P\cdot Q)=(P-1)(Q-1)$ となる。次に暗号用の鍵として $\Phi(N)$ と互いに素となる自然数 E を選択し、次の式を満たす復号化用の鍵 D を求める。S は任意の整数とする。

$$D\cdot E = S\cdot\Phi(N)+1 \tag{5.1}$$

互いに素となるような 2 つの値はユークリッド互除法で求められる。これは 2 つの値の最大公約数を求められる、$O(\log_{10}N)$ の高速なアルゴリズムである。

これを用いて最大公約数が 1 となれば互いに素な値を見つけることになる。D, E, N を用いれば、平文 M、暗号文 C には以下の式が成り立つ。

$$\text{暗号化} : C = M^E (\mathrm{mod} N)$$
$$\text{復号化} : M = C^D (\mathrm{mod} N)$$

E, N を公開しても、D を秘密にすれば、復号化出来るのは D を持つ人だけということが分かる。

ただし N を素因数分解出来れば、$\Phi(N)$ の値が分かりそこからユークリッド互除法で D の値が導き出されてしまう。

大きな値を因数分解するには NFS(Number field sieve) という手法が最も効率的だが、1024 ビットの鍵を現実的な時間内に因数分解できるほどではない。そのため RSA 使った暗号化では 1024 ビット以上の鍵を使用することが推奨されている。

ここで、実際の値を用いて、本当に暗号化、復号化出来るか確かめる。

まず、ユークリッドの互除法で 7253 と 120 は互いに素であることが分かったとする。次に拡張ユークリッド互除法により、$\gcd(A,B)=R$（$\gcd(A,B)$ は A と B の最大公約数を表す）となる場合に以下の式を満たす J と K を求める。

$$J\cdot A + K\cdot B = R \tag{5.2}$$

求めると、$A=7253$, $B=120$, $R=1$ のとき、$J=2599$, $K=-43$ となる。

また、式 (5.2) を変形すると式 (5.3) が得られる。

$$(K+B)A = (A-J)B + R \tag{5.3}$$

式 (5.3) にそれぞれの値を代入すれば

$$77\cdot 7253 = 4654\cdot 120 + 1$$

となる。

式 (5.1) と比較して、$\Phi(N)=120$, $D=77$, $E=7253$ とする事が出来る。

また、$P=11, Q=13$ とすれば $\Phi(P\cdot Q)=(11-1)(13-1)=120$ より $N=11\cdot 13=143$ と出来る。

この N、D、E を用いて平文 $M=98$ を暗号化すると $C=76$ となり、復号化すると $C=76$ のとき $M=98$ となり元に戻る事がわかる。

デジタル署名

RSA を使った技術の一つにデジタル署名がある。デジタル署名とはある情報の送信者が本人であることを証明する技術である。やり方を簡単に説明する。暗号化用の秘密鍵を使って暗号化し、それに対応する公開鍵を使ってしか復号化出来ないとすれば、それは秘密鍵を持つ本人にしか出来ないことになるからである。

RSA 暗号にまつわる時事的話題

現在 (2012 年 11 月)、RSA 暗号は 1024 ビット以上が推奨されているが、暗号技術検討会の 2011 年の報告書によると、2010 年以降には 1024 ビット RSA を 1 年以内に解読可能なスパコン (10^{15}FLOPS から 10^{17}FLOPS の処理能力が必要とされる) が出現すると予想されており、2015 年以降は 1024 ビット RSA は危殆化のおそれが高まってくるとある。また 2048 ビット RSA は今後 20 年は安全とされている。

以下はニュースサイト Wired に紹介されていたもので面白かったので、かいつまんで紹介する。まず上で紹介したデジタル署名技術は、秘密鍵がばれてしまえばだれでもその人になりすませることを意味している。RSA 暗号は 1024 ビット以上の鍵が推奨されているがそれ以下の鍵を使用している有名会社が複数見つかったのである。鍵は 1024 ビット以下では 384 ビット、512 ビット、768 ビットの 3 種類が見つかった。384 ビットはノート PC で 24 時間以内で因数分解出来、512 ビットはアマゾンのクラウドサービスを使えば 72 時間以内に出来たという。768 ビットのキーは因数分解出来なかったが政府や軍などの大きなグループが行えば可能だろうと述べている (2010 年に NTT が 768 ビットの素因数分解に成功しているようだ)。

デジタル署名技術の中に DKIM というものがある。これは DNS サーバに公開鍵を登録することによってデジタル署名を行うのだが、システムを構築したときにテスト用の鍵を作り DNS に登録する。この鍵をシステムを構築した後に消してしまえば問題ないが、それを忘れられているときがある。そしてテスト用の鍵なので十分な長さを持っていないことがあり、これを素因数分解出来れば秘密鍵が分かり、なりすますことが可能になるのである。

参考文献

[1] 「暗号技術検討会 2011 年度報告書」
<http://www.meti.go.jp/policy/netsecurity/docs/crypto/cryptrec2011.pdf>

[2] 「マイナビ記事 : NTT ら、768 ビット合成数を一般数体篩法にて完全分解に成功」
<http://news.mynavi.jp/news/2010/01/08/055/index.html>

[3] 「HACKING:美しき策謀 脆弱性攻撃の理論と実際 第 2 版」
hack: Jon Erickson (著), 村上 雅章 (翻訳) オライリージャパン

[4] 「NEWTON 2009 年 6 月号」
ニュートンプレス

[5] 「WIRED 記事 : How a Google Headhunter ' s E-Mail Unraveled a Massive Net Security Hole」
<http://www.wired.com/threatlevel/2012/10/dkim-vulnerability-widespread/2/>

6 パズルゲーム作成

情報工学課程 1 回生 木津 風真

6.1 ゲーム内容

今回作成したゲームは、ルールとしてはぷよぷよと一緒です。落ちてきたブロックを同じ色同士で 4 つ以上そろえると消え、連鎖を狙えるというやつです。

6.2 作成に使ったもの

基本的に C 言語を用いて作成しました。また、画像表示などの部分においては DX ライブラリを使用しています。DX ライブラリの基本的な部分は新・ゲームプログラミングの館を参考にしました。[1]

6.3 内容

まず、全体の大まかな流れを説明します。

```
#define CONTROL 0
#define FALL 1
#define DELETE 2
#define DECIDE 3
#define GAMEOVER 1
static int state;
static int drawflag;
static int game=0;

int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                        LPSTR lpCmdLine, int nCmdShow )
{
   ChangeWindowMode(TRUE);if( DxLib_Init() == -1 )return -1 ; //初期化
   SetDrawScreen(DX_SCREEN_BACK);
   back_init();block_init();map_init();
   state=DECIDE;
   while(ScreenFlip()==0&&ProcessMessage()==0&&ClearDrawScreen()==0&&Key_upd()==0){
      back_draw();
      drawflag=0;
      switch(state){
         case DECIDE:
            block_decide();
            //すぐに操作できるようにするため、break; は入れていません。
         case CONTROL:
            state=block_control();
```

```
            break;
         case DELETE:
            fall_map();
            state=block_delete();
            game=check_end();
            break;
         case FALL:
            state=fall();
            drawflag=1;
            break;
      }
      if(state!=FALL&&drawflag==0)
         block_draw();
      if(game==GAMEOVER)
         break;
   }
   DxLib_End() ;
   return 0 ;
}
```

ウィンドウに表示するようにするのと、最初に DX ライブラリの初期化を行います。その後に back_init(), block_init(), map_init() というのがありますが、これらは背景の画像やブロックの画像の読み込み、最初に出てくるブロックやブロックの配置を表すマップの初期化を行う関数です。

今回のゲームでは大まかに分けて 3 つの状態に分けられます。最初に宣言されている state という変数によって、どの関数が実行されるかが決まります。処理が行われた後 block_draw() によってマップにしたがってブロックの描画が行われます。

6.3.1 CONTROL もしくは DECIDE の場合

DECIDE の場合、block_decide() が呼び出されます。block_decide() では、今から表示するブロックに次のブロックの色として決められていた色を代入し、次のブロックを乱数によって決定しています。このとき同時に、落ちていくブロックの位置を初期化します。CONTROL もしくは DELETE の場合、その後に block_control() が呼び出されます。この関数では主に落ちているブロックの回転や左右移動をキーボード入力の状態によって制御しています。もし一番下まで落ちきっていない場合は CONTROL を戻り値として返し、落ちきった場合には FALL を戻り値として返します。

6.3.2 DELETE の場合

DELETE の場合 fall_map() と block_delete() の二つの関数が呼び出されます。fall_map() ではブロックが消えたり、ブロックが着地した際に穴ができた場合に、上にブロックがあればその穴を埋めるということをする関数です。block_delete() では関数の再帰呼び出しによって、4 つ以上つながっていた場合にブロックを消すということを行います。下の saiki() と del() はそのとき呼び出す関数で、saiki() によってつながっているブロックの個数を数え、4 つ以上だった場合に del() によって削除します。block_delete() はひとつもブロックが消えなかった場合は、次のブロックを表示させるために DECIDE を返します。もし消えた場合はブロックが空いたぶんだけ落ちる表示をするため FALL を返します。もしブロックがひとつも消えない状態で最上段の左から三番目にブロックがあると、ゲームオーバーになるので、check_end() でゲームオーバーかどうかを確かめ、GAMEOVER なら終了します。

```
void saiki(int i,int j){
   int k=map_color[i][j];
   del_color[i][j]=3;
```

```
   if(i!=11&&k==del_color[i+1][j]){
      saiki(i+1,j);
   }if(j!=11&&k==del_color[i][j+1]){
      saiki(i,j+1);
   }if(i!=0&&k==del_color[i-1][j]){
      saiki(i-1,j);
   }if(j!=0&&k==del_color[i][j-1]){
      saiki(i,j-1);
   }
   delcount++;
}
void del(int i,int j){
   int k=map_color[i][j];
   map_color[i][j]=3;
   height[j]--;
   if(i!=11&&k==map_color[i+1][j]){
      del(i+1,j);
   }if(j!=11&&k==map_color[i][j+1]){
      del(i,j+1);
   }if(i!=0&&k==map_color[i-1][j]){
      del(i-1,j);
   }if(j!=0&&k==map_color[i][j-1]){
      del(i,j-1);
   }
}
```

6.3.3 FALL の場合

FALL の場合 fall() が呼び出されます。fall() のなかでは DELETE のときに開いた穴の分を全体で 20 フレームで落ちていくモーションを表示します。この関数の中では何フレーム間呼び出されたかをカウントしており、20 フレーム経った時点で DELETE を返し、20 フレーム以内なら落ちるのを続けるため、FALL を返します。

6.4 最後に

今回ゲームを作ってみて、いろいろと詰まったところもあるのですが、とても楽しみながら作成しました。しかし、割と単純だったので次はもっと凝った感じのものを作りたいと思います。

参考文献

[1] ＤＸライブラリ置き場 http://homepage2.nifty.com/natupaji/DxLib/index.html

[2] 新・ゲームプログラミングの館 http://dixq.net/g/

7 ターミナル上でプレイするゲーム制作

情報工学課程 1 回生 出羽 裕一

7.1 はじめに

個人単位のゲーム制作では DX ライブラリというゲーム制作用ライブラリがあり、これを利用するのが最も容易にゲームを作る方法だと思う。だが今回はあえてこのツールを使わずに stdio や stdlib のような標準ライブラリを使用して、ターミナル上でプレイするゲーム制作を目指した。

この理由は DX ライブラリや OpenGL などのライブラリをインストールすることなく C 言語が使える環境が揃っていれば誰でも制作可能であるため、このゲームをプレイした人に

「これなら自分でも作れるのではないか」

「こういうものも作ることができるのか」

などの感想を持ってほしかったのである。

なお、今回 Windows 上でゲーム制作を行ったためヘッダファイルに windows.h があるが Linux の場合には stdio.h 内にある sleep 関数を使えばよい。

7.2 工夫点

今回の方針はターミナル上で動くゲーム制作なので、まず画像の表示はできない。

そこでゲーム内容は文章を読み進めていき、所々に選択肢を用意しその選択によって内容が変化するノベルゲームのようになるものの作成を目指した。

これを実現するために今回用意した関数は 2 つ。

- 文章表示関数
- 選択肢を表示し、その選択を受ける関数

まず文章表示関数の方だが、これは当初文章を 2 次元配列に格納しようとしたため

```
void String(str[][100],int n){
      printf("%s",str[n]);
      n++;
}
```

つまり、配列に文字列を格納し、1 文字ずつ読み込んでいくものとしていたが、途中から 2 次元配列よりもポインタを用いた格納の方がメモリを有効に使えるためポインタを用いた方法にした。[1]

```
void String(**str){
    while(*str){
        printf("%s",*str);
        str++;
    }
}
```

これはある文字列の先頭の文字が格納されているアドレスをポインタ str に代入し、先頭から文字を読み込んでいく、としてポインタの中には文章とその最後に NULL 文字を格納することにより、NULL までの文字列の表示を実現した。
選択肢関数は選択肢に相応する文字を入力させその文字で判別する関数にした。

```
int select(){
    char a;
    printf("選択肢を入力:");
    scanf("%*c%c",&a);
    fflush(stdin);
    if(a=='a' || a=='A')
        return 0;
    if(a=='b' || a=='B')
        return 1;
    後略
}
```

scanf の中にあるアスタリスクは変数に代入せず読み飛ばすという意味であり、[2] 今回は直前の入力の改行コードを読み飛ばすために使用した。
この 2 つの関数によって文章表示からの選択という操作が可能になりゲーム感が出てくる。今回は、特定の選択肢を選んだときのみステータスが上昇する、物語の確信に迫るような情報を得られる等のギミックを用意した。

7.3 ゲーム内容

今回の目的上、画像を使ったゲームはできないため非常にシンプルなものになり、またわかりやすいものにしないといけないという制限があった。
そこで、私が採用したゲームはじゃんけんだった。これなら画像での説明なしに誰でもゲームができると思ったからである。これを実現するためのプログラムは

1. 自分と相手のライフを格納する変数を 2 つずつ用意し片方だけ初期化
2. 乱数を取得しその数とプレイヤーが入力した数値とで評価
3. ライフポイントの変化を先ほど用意した 2 つめの変数に格納
4. どちらかのライフがゼロになるまでループ

乱数は各プレイ毎に数値を変化させるため time.h をインクルードし、時間を乱数のシードに使うことでこれを実現した。
また、今回はプレイヤー名を格納する配列を用意しライフを視覚的に表示できるようにループを使用してライフの数だけアスタリスクを表示する仕様にした。

```
int life1_o, life1_e, life2_o, life2_e;
char name[6];
int i, j;
do{
    printf("あなたの出す手を選択してください グー：1 チョキ：2 パー：3");
    scanf("%d",&i);
    j=Getrandom(1,3);
    switch((j-i+3)%3){
        case 0:
            life2_e=life1_e;
            life2_o=life1_o;
            break;
        case 1:
            life2_e=-life1_e;
            life2_o=life1_o;
```

```
            break;
        case 2:
        後略
    }
}
```

ここで、Getrandom 関数の詳細を記載する。

```
int Getrandom(int min,int max){
     return min(int)(rand()*(max-min+1.0)/(1.0+RAND_MAX));
}
```

これは引数 min から max までの中の数値で乱数を返す関数である。

7.4 最後に

今回の制作のメリットは

- プログラムが単純でこの分析は良い勉強になる
- 特殊なツールやライブラリを必要としないので手軽に制作できる
- C 言語の基本的な知識があれば作ることができる

という事柄があげられ、またデメリットして

- ターミナル上で動くので画像などが無く地味
- ゲーム性に欠ける

などがあげられる。この改善点として次回の制作では OpenGL などのゲーム制作ツールを使用した画像や音声を含んだゲーム性の高い作品の制作を目指したいと思う。

参考文献

[1] 図解 C 言語 ポインタの極意 柴田望洋著

[2] 明解 C 言語入門編 柴田望洋著

[3] 明解 C 言語によるアルゴリズムとデータ構造 柴田望洋＋辻亮介著

8 テトリスについて

情報工学課程 1 回生 中西 一人

8.1 はじめに

僕がこの半年コンピュータ部でやってきた活動は、主にゲームの制作で、今回、僕はテトリスをつくりました。

8.2 テトリスの制作

僕がコンピュータ部に入って、一番やりたいと思っていたことがゲームの制作で、特にシューティングゲームの制作に興味がありました。

しかし、実際にシューティングゲームの制作の勉強をしてみると、C++で作ってあるものばかりだったり、全然理解できないところばかりと、夏休みを十分につかって制作するという計画が一気に崩れてしまいました。

結局、簡単なゲームの制作から始めようと思い、選んだのがテトリスの制作でした。

すべてのソースコードを書くと、ページ数がかなり多くなってしまうので、一部のソースコードの紹介をします。理解できていない点もあるので、説明が変になっているところもあると思います。

なお、「コウのポケット」というサイトを参考にしています。[1]

8.2.1 ブロックの制作

まずは、ソースコードを書いておきます。なお、このソースを書くには、一番最初に、conio.h をインクルードする必要があります。

```
void ControlBlock()
{
    char key; //受け付けたキーを保存する変数

    key = getch(); //キーから一文字入力

    //キーに応じて各方向へブロックを移動したり、回転させたりする
    switch(key) {

        case 'c':
            if(!CheckOverlap(x+1, y)) {
                MoveBlock(x+1, y);
            }
            break;
        case 'z':
            if(!CheckOverlap(x-1, y)) {
                MoveBlock(x-1, y);
            }
            break;
```

```
        case 'x':
            if(!CheckOverlap(x, y+1)) {
                MoveBlock(x, y+1);
            }
            break;
        case ' ':
            TurnBlock();

    }
}
```

このソースの前に kbhit() というキーボードからの入力を判定する関数を書いてあります。

getch() はキーボードから入力された文字を Enter キーを押さなくても読み込んでくれる関数です。

getch() で読み込んだ文字を switch() で判別し、このソースとは別に書いてある MoveBlock() という関数を使って、キーに応じた方向にブロックを移動させるというソースコードです。

今回のソースは、右は c、左は z、下は x となっています。ブロックの回転は TurnBlock() というソースの上に空欄がありますが（case' ' となっているところ)、これはスペースキーを表しています。

8.2.2 ブロックの回転

先に、ソースコードを書いておきます。

```
int TurnBlock()
{
    int i, j; //for ループ制御用
    //ブロックを一時保存するための配列　回転させたブロックを入れるために一度 0 に初期化する
    int temp[4][4] = {0};

    //ブロックを回転する前に temp にセーブ
    for(i = 0; i<4; i++) {
        for(j = 0; j<4; j++) {
            temp[i][j] = block[i][j];
        }
    }

    //ブロックを回転
    for(i = 0; i<4; i++) {
        for(j = 0; j<4; j++) {
            block[i][j] = temp[3-j][i];
        }
    }
    //重なってるブロックが出てしまったらブロックを回転前に戻して中止
    if(CheckOverlap(x, y)) {
        for(i = 0; i<4; i++) {
            for(j = 0; j<4; j++) {
                block[i][j] = temp[i][j];
            }
        }
        return 1;
    }
```

```
    //一旦フィールドからブロック消して回転後のブロックを再表示
    for(i = 0; i<4; i++) {
        for(j = 0; j<4; j++) {
            field[y+i][x+j] -= temp[i][j];
            field[y+i][x+j] += block[i][j];
        }
    }

    ShowGameField();

    return 0;
}
```

ブロックの回転には、TurnBlock() という関数を使っています。

今回のプログラムでは、ブロックの配列の x と y を逆にし、y 軸を逆に回すと、ブロックの配列の中身を９０度右に回転できるようになっています。

とりあえず、ブロックのキーボードでの操作について書きました。もっと知りたい人は、「コウのポケット」で調べれば、出てきます。

8.3 終わりに

以上が、テトリスのソースの一部です。

分からないところを教えてくれた友達や先輩方、ゲームづくりに利用したサイトに感謝しています。

まだまだテトリスも未完成なので、シューティングゲームの制作等と一緒にテトリスにも完成させていきたいです。

参考文献

[1] コウのポケット プログラミング講座
http://www.nhk.or.tv/kow/program/index.php

9 数当てゲーム

情報工学課程 1 回生 山岡 孝史

9.1 概要

私はコンピュータ部に入ってから半年あまり経ちました。この間、私は C 言語を勉強し、いくつかのプログラムを書きました。他に、C++や Java に触れる機会もありましたが、やはり、この半年間での成果で一番の成果は基本的なことながらも C 言語を読み書きができるようになったことだと思っています。そこで、C 言語を使いゲームを作ることを思い立ちました。しかし、初めて作るゲームですから、まずは簡単なものから始めるのが良いでしょう。数当てゲームというのを考えてみました。数当てゲームとコンピュータが設定した数字を自力で見つけるというゲームです。

9.2 ソース

```
void subm(void);
void sukoshichiisai(void);
void chiisai(void);
void sukoshiookii(void);
void ookii(void);
void oshii(void);
void atari(void);
int count;
int a;
```

この void や int で始まっている関数や変数はこの先のプログラムで使うため自分で考えたものです。自分で考えたものですから、コンピュータにこれからこういう関数や変数を使うということを伝えなければプログラムは正常に動きません。なので、main 関数というものの前にこれからプログラムで使う関数や変数を必要があれば宣言します。これをプロトタイプ宣言といいます。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

main()
{
    count=1;
    printf("今からコンピュータが 1〜100 の数を作るのでいくつか当ててください！\n");
    srand((unsigned int)time(NULL));
    a=1+(int)(rand()*100/(1+RAND_MAX));
    printf("では今からその数を当てていって下さい。\n");
```

```
    subm();
}
```

コンピュータにおける乱数は疑似乱数というもので、計算することで得られる乱数に近いものです。この疑似乱数を作る関数は rand という関数です。これを使用するために#include<stdlib.h>として、ヘッダファイル stdlib.h をインクルードしなければなりません。また rand 関数を含む式は乱数の範囲指定のためのものです。rand() % (最大値-最小値)+最小値とすることで、最小値以上、最大値以下の範囲の数を得ることができます。RAND_MAX は rand 関数で得ることができる最大数を表す定数です。今回は 1〜100 が範囲なので最小値＝ 1、最大値＝ 100 としています。

srand 関数は乱数の計算に使う元の数字を変えるための関数です。なぜこの関数がいるかというと rand 関数だけでは乱数は作られますが、計算結果で何回やっても同じ値ができるのです。なぜかといえば、rand 関数の作るのは疑似関数、つまり計算して乱数を作っているために計算に使う元の数が同じならば、計算結果は同じになってしまうのです。このため srand 関数にランダムの数字を入れてプログラムが実行される毎に違う計算結果になるようにします。関数に入れるランダムな数字として現在時刻を選びました。現在の時刻を得るには time.h の time 関数を用います。srand((unsigned int)time(NULL)) とすることで、以降の rand が返す値は現在時刻を元に計算されるようになります。

```
void subm(void)
{
    int b;
    printf(" %d 回目\n" , count);
    scanf("%d", &b);
    if (b>a && b<=a+3)
        oshii();
    else if (b<a && b>=a-3)
        oshii();
    else if (b>a && b<a+7)
        sukoshichiisai();
    else if (b<a && b>a-7)
        sukoshiookii();
    else if (b==a)
        atari();
    else if (b>a)
        chiisai();
    else if (b<a)
        ookii();
}
```

この subm という関数には入力した数字と先程の処理で出来た乱数との大小関係で分岐する指示があります。if の後の括弧の中に書いてあることが分岐の条件です。例えば、上の方の条件を満たさない上で b¡a ならば ookii 関数の処理をし、b=a ならば atari 関数の処理をするという内容です。

```
void chiisai(void)
{
    printf("もっと小さい！\n");
    count++;
    subm();
}
```

```
void sukoshichiisai(void)
{
    printf("もう少し小さい！\n");
    count++;
    subm();
}

void ookii(void)
{
    printf("もっと大きい！\n");
    count++;
    subm();
}

void sukoshiookii(void)
{
    printf("もう少し大きい！\n");
    count++;
    subm();
}

void oshii(void)
{
    printf("惜しい！\n");
    count++;
    subm();
}

void atari(void)
{
    printf("当たり！\n");
}
```

subm 関数で分岐させた関数の処理の内容がここに書かれてあります。atari 関数は　当たり！　を表示させて終わりですが、その他の関数は文字を表示させた後、数字の入力回数に 1 足して再び subm 関数の処理に移るようにしてあります。つまり atari 関数に辿り着くまでは終わらないプログラムになっています。

9.3 まとめ

今回数当てゲームを手がけて、自分でも簡単なゲームを作ることができるということが分かり自信がつきました。乱数は自分で勉強しましたが、それ以外の部分は勉強した C 言語の知識のおかげで戸惑うことなく進めることができました。

最初は弾幕風の縦シューティングを作る予定でしたが、私の知識が足りないために作ることができませんでした。シューティングは作るプログラムの量も数当てゲームと比べるとかなり多く、より複雑です。来年までにはしっかりと必要な知識を自分のものそして、ちゃんとしたシューティングゲームをを作りたいと思っています。

参考文献

[1] 脱初心者 C 言語　http://www33.ocn.ne.jp/˜ oreley/FSB/c03.html

[2] 乱数　http://homepage3.nifty.com/mmgames/c_ guide/21-02.html

10 オセロ作成

情報工学課程 1 回生 吉村 健志

10.1 初めに

今回は、初めて本格的にゲームを作るということでオセロを作ることにしました。ライブラリは DX ライブラリを使用しています。ゲームを作るのは初めてだったので読みにくいところが多いですがおおむね思っていたものが作れたと思っています。

10.2 main 関数

最初は main 関数のループ部分の説明をします。なるべく、コンパクトにまとめようとはしたのですが大分ごちゃごちゃした感じになってしまいました。state_ini は初期化、ロード、次の 3 つの state_choice では choice1 関数で対人か AI か選択、choice4 関数で先攻後攻、choice3 関数で AI のレベルを選択するようになっています。state_play は対戦となっていて cpu の方は AI との対戦になっています。その中の calc_flame 関数は選択用の枠の移動の計算をしており、graph_turn 関数は、どちらの番かを表示しています。裏返せるかどうかは state_play の input 関数で判断しています。input 関数内には、put 関数という関数があり、その中にある関数で裏返せるかどうかを判断しています。次に state_play_cpu の変数 pre は AI と戦う場合、どちらが先攻後攻かを表しています。また、checkEnd 関数で置ける場所がなければ、相手の番に変えたり state_check_win に移動させたりします。全体を通して使用している graph 関数は、ゲーム画面の背景、オセロの駒の表示をしています。最後に、state_check_win で勝者判定になっています。choice2 関数では再戦かタイトルに戻るかどうかの選択をするようになっています。

```
while(ScreenFlip()==0 && ProcessMessage()==0 &&
        ClearDrawScreen()==0 && ProcessLoop()==0){
    switch(state){
        case state_ini :
            ini();
            load();
            state = state_title;
            break;

        case state_title :
            title();
            break;

        case state_choice1 :
            ini();
            graph();
            choice1();
            break;
```

```
        case state_choice2 :
            graph();
            choice4();
            break;

        case state_choice3 :
            graph();
            choice3();
            break;

        case state_play :
             calc_flame();
             if(Key[KEY_INPUT_ESCAPE] == 1)
                 state = state_title;
            graph();
            graph_turn();
            input();
            switch(checkEnd(turn)){
                case 1:
                    turn = (turn+1)%2;
                    break;
                case 2:
                    state = state_check_win;
                    break;
            }
            break;

        case state_play_cpu :
            calc_flame();
            if(Key[KEY_INPUT_ESCAPE] == 1)
                state = state_title;
            graph();
            graph_turn();
            switch((pre ? 1 : 0)){
                case 0:
                    input();
                    break;
                case 1:
                    if(cpu==0)
                        cpu_0();
                    else
                        cpu_1();
                    break;
            }
            switch(checkEnd(turn)){
                case 1:
                    turn = (turn+1)%2;
                    break;
                case 2:
```

```
                    state = state_check_win;
                    break;
            }
            break;

            case state_check_win :
                graph();
                Winner();
                choice2();
                break;
        }
        if(state == 9999)
            break;
    }
    DxLib_End();
    return 0;
}
```

10.3 AI 関数

2 つほど作ったのですがなかなかの弱さを誇っています。一つ目の方はおける場所にランダムで置くだけのものです。rand 関数の値の余りを取り、その値を配列に代入し put 関数で判断しおける場合は置きます。そして二つ目は、角と内側に置く優先度を上げています。check 関数は、駒がおけるかどうかのチェックをします。もう少し強いものが作りたかったのですが、実装が難しく 2 つだけになってしまいました。

```
void cpu_0(){

    int i,j;

    i = rand()%8;
    j = rand()%8;

    if(put(j,i,turn)==1){
        turn = (turn+1)%2;
        pre = (pre+1)%2;
    }
}

int check1(int turn){

    int i2,j2;

    for(i2 = 2 ; i2 < 6 ; i2++){
        for(j2 = 2 ; j2 < 6 ; j2++){
            if(board[i2][j2] == 0 && check(i2,j2,turn) == 1)
                return 1;
        }
        return 0;
```

```
    }
    return 0;
}

void cpu_1(){

    int i1,j1;

    for(i1 = 0 ; i1 < 8 ; i1+=7){
        for(j1 = 0 ; j1 < 8 ; j1+=7){
            if(put(i1,j1,turn)==1){
                turn = (turn+1)%2;
                pre = (pre+1)%2;
            }
        }
    }
    i1 = rand()%8;
    j1 = rand()%8;

    if(check1(turn)==1){
        if(j1>=2 && j1<=5 && i1>=2 && i1<=5)
            if(put(j1,i1,turn)==1){
                turn = (turn+1)%2;
                pre = (pre+1)%2;
            }
    }
    else{
        if(put(j1,i1,turn)==1){
            turn = (turn+1)%2;
            pre = (pre+1)%2;
        }
    }
}
```

10.4 終わりに

今回は、動作自体はしたのですが、中身が至らないところや関数は参考にさせてもらった部分が多く、次回は自分の手でできるだけ作っていきたいと思いました。また、ファイル数が無駄に多くなってしまった感じがしてそこも次回は直したいです。最後に、このオセロ作成で関数のゲームでの使い方が少し理解できたのが良い点だったと思います。次は、これを生かして格ゲーが作れればと思います。

参考文献

[1] 簡単オセロ制作 http://idehideout.fc2web.com/p/rev/00.html

[2] 新・C 言語 ～ゲームプログラミングの館～ [DX ライブラリ] http://dixq.net/g/

11 さめがめを作ってみた

情報工学課程 1 回生 渡邉 雄也

11.1 さめがめについて

さめがめというのは、くっついている 2 つ以上の同じ色のブロックを選択すると、それらのブロックが消えるゲームである。そうすると、上にあるブロックや右にあるブロックが隙間を埋める。その行動を繰り返しどんどんとブロックを消していくゲームである。一回で多くのブロックを消すことが出来れば獲得点数は増加する。また、最後に残ったブロックの数によってボーナスポイントが得られる。今回は 3 回行い、その合計点数を競い合うゲームとなる。なお、確実に全てが消えるということはないので注意が必要である。なので、多少運ゲーは否めない。

11.2 DX ライブラリについて

DX ライブラリとは、DirectX を使った Windows ソフトの開発に付いて回る DirectX や Windows 関連のプログラムを使い易くまとめた形で使うことが出来るようにした C++言語用のライブラリのことである。なお、C 言語の知識だけでももちろん使うことができる。機能には、グラフィックス機能、サウンド機能、入力関係（ジョイパッド入力機能やマウス入力機能など）などゲームに必要なものがある。今回はこの DX ライブラリを学びながら、且つ存分に使わせてもらってゲームを作る。

11.3 プログラムについて

全てを説明すると長くなるのでセイムパズルに関係するプログラムを中心に抜粋する。もちろん、セイムパズルに関係ないものもある。

11.3.1 ウィンドウズモードかフルスクリーンモードか選択

ウィンドウズモードかフルスクリーンモードにするかを起動時にメッセージボックスを出し、プレイヤーが選択できるようにする。

```
void ask_window_mode(){
    int flag;
    //メッセージボックス起動
    flag=MessageBox(NULL , TEXT("フルスクリーンモードで起動しますか？\n\n「はい」推奨\n") ,TEXT("
スクリーン設定") , MB_YESNO | MB_ICONQUESTION );
    if(flag != IDYES)
        ChangeWindowMode( TRUE );//ウィンドウモードに変更
}
```

プレイ中にモードの切り替えは出来ない

11.3.2 ブロックを落とす、または左に移動する

セイムパズルはブロックを消すと上に乗っていたブロックが落ちてくる。

```
for(x=0;x<FDX;x++){
    for(i=0;i<FDY-1;i++){
        for(y=0;y<FDY-1;y++){
            if(block[y][x].flag==1 && block[y+1][x].flag==0){
                block[y+1][x].bcolor=block[y][x].bcolor;
                 block[y  ][x].flag=0;
                block[y+1][x].flag=1;
            }
        }
    }
}
```

この時の flag というのは 0 か 1 でブロックのあるなし (0 の時はブロックが無く、1 の時はブロックがある状態）を判断するものである。また、x 軸が横軸で、y 軸が縦軸である。なお、y 軸の正負は上が負、下が正となっている。なので、「落とす」という表現をしているが、実際には y 軸の正の方向に移動している。左に移動する場合もほぼ同じであるのでプログラムは割愛する。

11.3.3 マウス操作

マウス操作で行えるようにする。つまり、消したいブロックのところでクリックするとブロックを消すようにすることである。これは DX ライブラリで非常に簡単にできる。以下の様に mouse.x と mouse. y にマウスの位置を取得することができる。また、クリックの判定も以下のようにできる。

```
int GetMouseState(){
    int MouseInput,input=0;
    GetMousePoint(&mouse.x,&mouse.y);
    MouseInput = GetMouseInput() ;
    if((MouseInput & MOUSE_INPUT_LEFT) == 1)
        mouse.cnt++;
    else
        mouse.cnt=0;
    return 0;
}
```

これを利用すると、mouse.cnt==1 の時にクリックしたことを判定できる。

11.3.4 ４方向のブロックのつながりをカウント

セイムパズルはブロックを消さなければならない。長くなるが以下のとおりである。

```
void block_delete(int bx, int by, int col, int flag){
    if(cpblock[by][bx].flag==1){
        if(bx-1>=0  && cpblock[by  ][bx-1].flag==1 && col==cpblock[by  ][bx-1].bcolor)
            delete_calc(bx,by,col,flag), block_delete(bx-1,by,col,flag);
            //左
        if(bx+1<FDX && cpblock[by  ][bx+1].flag==1 && col==cpblock[by  ][bx+1].bcolor)
```

```
                delete_calc(bx,by,col,flag), block_delete(bx+1,by,col,flag);
                //右
            if(by-1>=0  && cpblock[by-1][bx  ].flag==1 && col==cpblock[by-1][bx  ].bcolor)
                delete_calc(bx,by,col,flag), block_delete(bx  ,by-1,col,flag);
                //上
            if(by+1<FDY && cpblock[by+1][bx  ].flag==1 && col==cpblock[by+1][bx  ].bcolor)
                delete_calc(bx,by,col,flag), block_delete(bx  ,by+1,col,flag);
                //下
        }
    }
```

マウスクリックは違う関数で判定している。なお、flag は上記には書いていないがブロックが消えるときのエフェクトの動作で利用している。

11.3.5 ハイスコアについて

スコアの計算はブロックが消えたり、最後に残ったブロック数に応じてのボーナス加算などで計算している。そして、ハイスコアの計算は以下のようにしている。

```
    if(highscore<score){
        highscore = score;
    }
```

要するに現在のスコアがハイスコアを超えると自動的にハイスコアとして計算されていく。

11.4 終わりに

色々と偉そうに書いていますがインターネットや人のソースをものすごく参考にしながら作成しました。なので、自分のソースとはまだまだ言えないです。しかし、人のソースを見て色々と勉強でき、非常に楽しかったです。また、DX ライブラリに非常に助けられました。これを使えば、初心者の私でも画像や音声などを組み込めるので、もっと勉強をして楽しいゲームを作りたいです。しかし、いつまでも頼ってばかりではいけないので DX ライブラリを使わない作品にも挑戦したいです。興味が尽きず、やりたいことばかりです。一歩ずつ進めていきます。次回、どんな作品になるか分かりませんが今回の作品は本当に楽しく作れました。参考にしたサイトや大学の皆様、本当にありがとうございました。それでは。

参考文献

[1] 新・ゲームプログラミングの館　http://dixq.net/g/

[2] ＤＸライブラリ置き場　http://homepage2.nifty.com/natupaji/DxLib/index.html

[3] ハンゲーム　http://www.hangame.co.jp/

編集後記

Lime46 号はいかがでしたでしょうか。部長の峯岡です。今回とある事情により編集も担当しておりまして、こちらでもお会いする機会がございました。

実は今回の Lime は「これで学祭に間に合うのか?」というほどにギリギリの日程での作業となってしまいました。(実はこの文章を書いている私は内心ヒヤヒヤです。) 一重に私の時間管理の至らぬが故でございます。迷惑をかけた後輩部員たち、心配して下さった先輩部員・OB の方々、この場を借りて謝罪させて頂きます。誠に申し訳ありませんでした。

今頃自分は今回も無事 Lime を発行することが出来たということで一安心している頃でしょう。次号の Lime にも是非ご期待ください。では、では。

平成 24 年 11 月 16 日
編集担当 峯岡 健人

Lime Vol.46

平成 24 年 11 月 23 日 発行 第 1 刷

発行 京都工芸繊維大学コンピュータ部
http://www.kitcc.org/