平成 25 年 11 月 23 日
京都工芸繊維大学コンピュータ部

Lime 48

はじめに

こんにちは、部長の中川です。今年は学祭の規模も小さくなり、例年の学祭の売りの一つだった酒類の販売もなくなりましたが、本学の学祭にお 越しいただき、さらに我々コンピュータ部のもとへお越しいただき、さらにさらにこの Lime 第 48 号を手に取ってくださったこと、厚く御礼申し上げます。

この Lime は、部員の活動の一部を記したものです。日々の活動の成果や興味に対する調べ物、個人的趣味全開なものまで内容はさまざまです。これを通じてコンピュータ部のことを知っていただき、さらに内容に興味を持っていただけたり、面白いと感じていただけると幸いです。レポートや課題に追われながら記事を書きあげた甲斐があります。ぜひ、最後までお付き合いください。

では、編集担当の渡邉君、今年の始めから私を支えてくれた他の部員各位、さらには OB・OG の方々を始めコンピュータ部を支えてくださるすべての方に感謝しつつ、はじめの挨拶とさせていただきます。

平成 25 年 11 月 17 日
京都工芸繊維大学コンピュータ部部長　中川　鴻佑

目 次

1 五目並べ

情報工学課程 1 回生 木本 大介

1.1 はじめに

自分はこれまでに勉強してきた C 言語を使用し、○×ゲームを参考にして五目並べを作りました。

1.2 操作方法

このゲームを作るにあたって、升を数値で指定するタイプとカーソルを動かして升を指定するタイプの二種類の操作方法に関する関数ができました。

ここでは操作性が高くて扱いやすいカーソルで升を指定するタイプについて説明していきます。

1.3 ソース

コンソール上でカーソルを使って操作を行うには、conio.h で提供される関数を使用することで可能になります。

また、const int n=10 と最初に入力することで n という変数を 10 という定数に定義しています。この値を変更するとボードの大きさを変更する事が可能です。

○と×を指定した位置に入力した後にボードを出力する putoff 関数とプレイヤーが勝利条件を満たしているかを判定する check 関数の 2 つがあります。

check 関数は相当な量があるのでソースは省略しますが、縦・横・斜めのそれぞれで連続して並んでいる同じマークの数を調べ、最大で連続している数が 5 つ以上であればどちらのプレイヤーが勝利しているかを述べた後にゲームが終了するように main 関数へと数値を返すようになっています。

putoff 関数は入力に対応する部分と出力に対応する部分を分けて説明しますと、まず入力だと

```
void putoff(int board[],int player){
    int in,save;
    int a=0;
    save = board[0]; //指定している board[] の数値を save に保存

    board[0] = 3; //指定している board[] に 3 を代入

    //ここでの\n はエンターキーの入力を表す
    while( (in=getch() != '\n' ){
        if(in == 'w'){
            board[a] = save;//board[a] の元の値を save から取り出す
            if( a >= n) a = a-n; //カーソルを 1 つ上に移動させる
            //カーソルが一番上にある場合は一番下へ移動
            else a = a+n*(n-1);
        }

        else if(in == 'a'){
```

```
            board[a] = save;

            //カーソルが一番左にある場合は一番右へ移動
            if(a%n == 0 || a % n == n) a = a+n-1;
            else a = a-1; //カーソルを1つ左に移動させる
        }

        //w,a,s,d,Enter 以外の入力を受けた際に再入力する
        else continue;

        //新しく指定した board[] の値を save に保存する
        save = board[a];
        board[a] = 3; //新しく指定した board[] に3を代入
    }
    //指定してある升が空白なら○か×を入力する
    if(save == 0)board[a] = player;
    else{
        //指定してある升に○か×が既に入力されている場合、
          putoff 関数をやり直す
        printf("再度入力してください\n");
        board[a] = save;
}
```

本来ならカーソルの移動を上:w, 左:a, 下:s, 右:d の4つに対応させていますが、数値が違うだけで基本は同じなので上と左についてのみ書いています。

カーソルの表示によって升の中身が消えないように変数 save に board[] の中身を毎回保存し、エンターキーを押すことで while によるループを抜け出すようになっており、player の数値に関しては main 関数の方で数値を決定しています。

次に出力だと

```
    int i;
    for(i=0;i<n*n;i++){
        //board[] の値が 1 の時、○を出力
        if(board[i] == 1) printf("○");
        //board[] の値が 2 の時、×を出力
        if(board[i] == 2) printf("×");
        //board[] の値が 3 の時、●を出力
        if(board[i] == 3) printf("●");
        //board[] の値が 0 の時、_を出力
        if(board[i] == 0) printf("_");
        //これにより n*n の正方形になるよう改行される
        if(i%n == n-1) printf("\n");
    }
```

初期状態では全ての升を空白にし、カーソルとなる3が代入されている場合は●、player の数値である1や2が代入されている場合は○や×を表示しています。ボード全体が正方形で表示されるように改行をしているのもこの部分となります。

そして main 関数ですが、

```
int main(){
```

```
    int board[(n+1)*(n+1)];
    int i;

    //board[] 全てに 0 を代入
    for(i=0;i<(n+1)*(n+1);i++) board[i] = 0;

    //check 関数で勝利条件が満たされていない場合ループする
    while(check(board) == 0){
        putoff(board,1); //player=1 として board に 1 を代入する

        if(check(board) == 0){
            putoff(board,2); //player=2 として board に 2 を代入する
        }
        else break;
    }
    return 0;
}
```

board について定義して全てに 0 を代入し、putoff 関数と check 関数に必要な物を用意しています。

check 関数では勝利条件が満たされることで 0 以外の数値が返されるので、それにより while のループから抜けてゲームが終了し、勝利条件を満たしていない限りはループで延々と○と×を入力し続けます。

1.4 参考文献

○×ゲームを C 言語で作成する
http://program-study.hatenablog.com/entry/20121107/1352248936

2 ミニゲーム作成

情報工学課程 1 回生 久保 克弘

2.1 ソースコード

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(void){
	int count,x,y,n=100;
	srand(time(0));
	x=rand()%100;
	printf("NUMBERS GAME\n");
	count = 0;
	while(n){
		printf("プログラムが 0〜99 の数字を一つランダムで選びます。\n 推測して数字を入力してくださ
い:");
		scanf("%d",&y);
		count++;
			if(x<y)
				printf("答えはその数より小さいです\n");
			else if(x>y)
				printf("答えはその数より大きいです\n");
			else break;
		n--;
	}
	printf("正解 !! おめでとうございます !! 入力回数 %d 回\n",count);
	return 0;
}
```

2.2 脚注と感想

include ＜stdlib.h＞は srand 関数を使用する際に必要 srand 関数は rand 関数で発生させる擬似乱数の発生系列を変更します。while(n) は、n が 0 でない間だけ、ループに入ります。どうせなら入力回数も分かった方がいいかと思って付け足してみました。

自分でゲームを作るのは初めてなので手探り状態でしたがまずは習った範囲で出来るものをと考えました。来年はもっと応用の効いた作品を作りたいと思います。

TeX での原稿作成も前期の授業の復習もかねていい勉強になりました。

参考文献

LaTeX コマンド集　http://www.latex-cmd.com/

3 C言語でヌメロンつくってみた

情報工学課程 1 回生 小西 健太・小林 勇揮

3.1 ヌメロンとは

以下は Wikipedia からの引用です。

> Numer0n(ヌメロン) はフジテレビの「COOL TV」枠内で、毎週月曜日 23:00 - 23:30（JST）放送のゲームバラエティ番組である。
>
> マスターマインドをベースに、相手が選んだ数字を当てる推理ゲーム番組。タイトルの「Numer0n」はギリシャ語で「数字」を意味する「NUMERO」と「 - の人」という意味を表す「ON」を組み合わせた造語であり、番組では O の字を数字のゼロで表している。
>
> 2012 年 10 月から 2013 年 6 月までは 23 時から放送された。2013 年 9 月時点でゲームアプリのダウンロード数が 320 万を超えた。その後は再び単発で放送。

3.1.1 ルール

> それぞれのプレイヤーが、0-9 までの数字が書かれた 10 枚のカードのうち 3 枚を使って、3 桁の番号を作成する。カードに重複は無いので「550」「377」といった同じ数字を 2 つ以上使用した番号は作れない。
>
> 先攻のプレイヤーは相手の番号を推理してコールする。相手はコールされた番号と自分の番号を見比べ、コールされた番号がどの程度合っているかを発表する。数字と桁が合っていた場合は「EAT」（イート）、数字は合っているが桁は合っていない場合は「BITE」(バイト) となる。 例として相手の番号が「765」・コールされた番号が「746」であった場合は、3 桁のうち「7」は桁の位置が合致しているため EAT、「6」は数字自体は合っているが桁の位置が違うため BITE。EAT が 1 つ・BITE が 1 つなので、「1EAT-1BITE」となる。
>
> これを先攻・後攻が繰り返して行い、先に相手の番号を完全に当てきった（3 桁なら 3EAT を相手に発表させた）プレイヤーの勝利となる。

3.2 ソース解説

ここでは、C 言語で作ったヌメロンのソースを解説したいと思います。
ヌメロンのゲームに関わるソースだけを選んでいるので、実際に私たちが書いたソースとは異なる部分があります。
まず、main 関数でプログラムの流れを簡単に説明し、その後それぞれ作成した関数について具体的に解説をしていきます。

3.2.1 main 関数

main 関数

```
int main(){
    define();
    create_number();
    while(1){
        printf("あなたの攻撃です。\n");
        input_number();
        if(eat_bite()==3){
            printf("あなたの勝ちです。\n");
             break;
        }
        printf("CPU の攻撃です。\n");
        ai_fibonacci();
        if(reseved()==3){
            printf("CPU の勝ちです。\n");
            break;
        }
    }
    return 0;
}
```

「1.1 ルール」で説明したようにゲームを進行していきます。

1. コンピュータ側でヌメロンに必要な数字を準備します。
   ・define() では、ヌメロンで使用する 3 桁の数字の組み合わせをすべて定義します。この define() は、のちに出てくる ai_fibonacci 関数を使用するために必要です。
   ・create_number() で CPU 側の 3 桁の数字を生成します。

2. whike 関数からヌメロンのゲームがスタートします。
   ・この while の｛｝内は break が実行されるまで（勝敗がつくまで）無限ループを繰り返すことになります。

3. プレイヤー側から行動が始まります。
   ・input_number() で CPU の数字を当てるため 3 桁の数字を入力します。
   ・eat_bite() で input_number() と creat_number() で生成した 3 桁の数字を比較して eat 数と bite 数を計算します。このとき eat 数が 3 のとき「あなたのかちです」と表示されてゲームが終了します。

4. CPU 側の行動が始まります。
   ・ai_fibonacci() でプレイヤーの数字を当てるため 3 桁の数字を生成されます。
   ・その 3 桁の数字を見てプレイヤーは reseve() で eat 数と bite 数を入力します。ここでも eat 数が 3 のとき「CPU のかちです」と表示されてゲームが終了します。

5. 勝敗がつくまでプレイヤーと CPU の行動がループされることになります。

3.2.2 数字の生成・入力

1. define()

define 関数

```
int try[720];
int define(){
```

```
    int i,j,k,count=0;
    for(i=0;i<10;i++){
        for(j=0;j<10;j++){
            for(k=0;k<10;k++){
                if(i!=j&&j!=k&&i!=k){
                    try[count++]=100*i+10*j+k;
                }
            }
        }
    }
    for(i=0;i<720;i++){
        int r = GetRandom(0,719);
        int imp =try[i];
        try[i] = try[r];
        try[r] = imp;
    }
    return 0;
}
```

この define 関数は、0 から 9 の数字で同じ数字を 2 つ以上使わない 3 桁の数字を全通り作成する関数です。これらの数字は ai_fibonacci 関数で使うので、int 型のグローバル変数 try [720] を作り、そこに代入することになります。ヌメロンで使う 3 桁の数字は全部で 720 通りあります。
ai_fibonacci 関数で使う関係上ランダムに並べます。ランダムに並べるやり方は非常に単純で、try[0]〜try[719] を順番に try[r] と入れ替える（r はランダムな数字）というものです。GetRandom(0,719) は 0〜719 までの数字をランダムに手に入れてくれるというものです。

2. GetRandom()

GetRandom 関数

```
int GetRandom(int min,int max){

    static int flag;

    if (flag == 0) {
        srand((unsigned int)time(NULL));
        flag = 1;
    }

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

GetRandom 関数は min に代入された数字から max に代入された数字までの範囲でランダムに数字を手に入れてくれる関数です。
乱数のシード値に時間を使うことにより、実行毎に違った値を得ることができます。そのために
srand((unsigned int)time(NULL));
を使うのですが、これは一回宣言するだけで十分なので、上記のようなソースにしました。関数内で宣言する変数の前に static を付けると、関数が終わっても値を保持することができます。また、static int で宣言した変数 frag の初期値が 0 なので、結果的に srand((unsigned int)time(NULL)); は一回だけ宣言されることになります。また、返

り値の
min + (int)(rand()*(max-min+1.0)/(1.0+RAND_MAX))
は min に代入された数字から max に代入された数字までの範囲のランダムな数字を返します。

3. create_number()

create_number 関数

```
int cpu[3];
int create_number(){
    do{ cpu[0]=rand()%10;
        cpu[1]=rand()%10;
        cpu[2]=rand()%10;
    }while(cpu[0]==cpu[1]||cpu[0]==cpu[2]||cpu[1]==cpu[2]);
return 0;
}
```

create_number 関数は、CPU 側の 3 桁の数字を生成する関数です。
int 型のグローバル変数 cpu[3] を宣言します。ここで、cpu[0] は百の位、cpu[1] は十の位、cpu[2] は一の位です。ここでは、do-while 文を使います。まず、cpu[0]〜[2] にランダムな数字を代入します。次に条件式が 3 桁の数字のうち同じ数字を 2 つ以上使っている場合なので、これを満たす場合 cpu[0]〜[2] にランダムな数字を代入します。条件式を満たさなくなるまでこれを繰り返します。これによりヌメロンに使える数字を作れることができます。

4. input_number()

input_number 関数

```
int user[3];
int input_number(){
    char inp[8];
    int num,len=0;
    while(len!=3){
        printf("3 桁の数字を入力してください\n");
        scanf("%s",inp);
        num=atoi(inp);
        len=strlen(inp);
        if(len!=3){
            printf("無効な数値が入力されました。入力しなおしてください。\n");
        }
    }

    user[0]=num/100;
    user[1]=(num/10)%10;
    user[2]=num%10;

    return 0;
}
```

input_number 関数は、プレイヤーがコールした 3 桁の数字を受け取るための関数です。
int 型のグローバル変数 user[3] を宣言します。ここで、user[0] は百の位、user[1] は十の位、user[2] は一の位です。まず、必要な変数を準備します。ここでは入力ミスを防ぐために while 文で入力された数字の長さが 3 文字でなかっ

たら、while の {} 内のソースを繰り返し実行します。while の {} 内では、まずプレイヤーが入力した数字を文字列で inp に代入します。次に、atoi() を利用して、num に inp の文字列を数値に変換して代入します。そして、strlen() を利用して、len に inp の文字列の長さを計算した数字を代入します。一応、3 文字ではなかった場合エラー通知を出してプレイヤーに知らせます。うまく 3 桁の数字が入力されたら、num の数字の百〜一の位をそれぞれ user[0]〜[2] に代入します。

3.2.3 ＥＡＴ数とＢＩＴＥ数を数える

1. eat_bite()

eat_bite 関数

```
int eat_bite(){
    int i=0,j=0;
    int eat=0;
    int bite=0;
    int bite_temp=0;
    for(i=0;i<3;i++){
        if(user[i]==cpu[i]){
            eat=eat+1;//eat 数カウント
        }
    }
    for(i=0;i<3;i++){
        for(j=0;j<3;j++){
            if(user[i]==cpu[j]){
            bite_temp=bite_temp+1;//eat または bite 数のカウント
            }
        }
    }
    bite=bite_temp-eat;//bite 数の割り出し

    printf("EAT:%d\nBITE:%d\n",eat,bite);

    return eat;
}
```

eat_bite 関数は、eat 数と bite 数を割り出し、それを表示する関数です。
eat 数の割り出し方は、調べる位の数が同じ場合のときを数えることです。bite 数の割り出し方は、eat または bite 数から eat 数を引くことで出します。まず、それらを数えるために変数を用意します。eat 数を数えるために eat、bite 数を数えるために bite、eat または bite 数を数えるために bite_temp を int 型で宣言します。eat 数を数えるために for 文で同じ桁の数が同じなら、eat に 1 を足します。これを 3 回繰り返すことにより、eat 数を数えます。eat または bite 数を数えるためには for 文を二重構造で使うことにより user[0] と cpu[0]〜cpu[2]、user[1] と cpu[0]〜cpu[2]、user[2] と cpu[0]〜cpu[2] を比較します。これにより、eat または bite 数を数えることができます。あとは、bite_temp から eat を引けば、bite 数を求めることができます。最後に printf 関数で eat と bite 数を表示します。返り値には eat を返します。これは、main 関数中の一つ目の if 文で効果を発揮します。

2. reseved()

reseved 関数

```
int eat_save;
```

```
int bite_save;
int reseved(){
    printf("EAT 数を入力してください:");scanf("%d",&eat_save);

    printf("BITE 数を入力してください:");scanf("%d",&bite_save);

    return eat_save;
}
```

---

reseved 関数は、プレイヤー側が入力した eat 数と bite 数を受け取ってくれる関数です。
int 型のグローバル変数で eat 数を代入するために eat_save、bite 数を代入するために bite_save を宣言します。関数内では、printf 関数と scanf 関数を使いプレイヤーが入力した eat 数と bite 数をそれぞれの変数に代入します。返り値には eat_save を返します。これは、main 関数中の二つ目の if 文で効果を発揮します。

3.2.4 コンピュータのＡＩ

ai_fibonacci 関数

```
int attack[3]={0,0,0};
int ai_fibonacci(){

    static int count=0;
    int i=1;
    if(count==0){
        int first;
        first=rand()%721+1;
        attack[0]=try[first]/100;
        attack[1]=(try[first]/10)%10;
        attack[2]=try[first]%10;

        printf("%d%d%d\n",attack[0],attack[1],attack[2]);
    }
    if(count!=0){
        for(i=0;i<720;i++){
            if(eat(attack[0],attack[1],attack[2],i)!=eat_save||bite(attac
k[0],attack[1],attack[2],i)!=bite_save){
                try[i]=-1;
            }
        }
        for(i=0;i<720;i++){
            if(try[i]!=-1){
                attack[0]=try[i]/100;
                attack[1]=(try[i]/10)%10;
                attack[2]=try[i]%10;
                printf("%d%d%d\n",attack[0],attack[1],attack[2]);
                break;
            }
        }
    }
```

```
    count+=1;

    return 0;
}
```

---

ai_fibonacci 関数は、CPU 側が自動でプレイヤー側の 3 桁の数字を求めてくれる関数です。

1. まずランダムに 3 桁の数字を作りコールします。

2. プレイヤーからの eat と bite 数（reseved()）を利用して、プレイヤー側の 3 桁の数字を絞り込んでいきます。

   define 関数で生成した 3 桁の数字を全通りを利用して、プレイヤーから eat 数と bite 数を参考に絶対に当てはまらない数字を除外していきます。これを繰り返すことによりプレイヤー側の数字を絞り込んでいきます。具体的には、まず CPU 側の 3 桁の数字を作るために int 型のグローバル変数で attack[3] を宣言します。例の通り百の位は attack[0]、十の位は attack[1]、一の位は attack[2] に代入します。次に CPU 側の最初のコールに使う 3 桁の数字はランダムに作るので、乱数を使用します。これは最初の一回だけできればよいので、int 型の前に static を宣言して count という変数を用意します。そして、if 文でもし count が 0 であればランダムに 3 桁の数字を作るようにします。これには define 関数で作った try[720] を利用しています。最後に count+=1; とすることにより、count に 1 を足すことでこの動作が一度だけ行われるようにします。2 回目以降の動作は 2 つ目の if 文のみ実行されることになります。その if(count!=0) の｛｝内ではプレイヤーから受け取った eat 数と bite 数を参考にプレイヤー側の 3 桁の数字を絞り込んでいきます。まず最初に作った 3 桁の数字とその eat 数と bite 数を参考に絶対に当てはまらない数字を除外していきます。そこで、eat() 関数と bite() 関数を利用します。これにより、if 文で、もし返された eat 数と bite 数がプレイヤーが入力した eat 数と bite 数と合わない場合、try[i] に-1 を代入することで、この値は違うということを示します。これを for 文で 720 回繰り返すことで全通り確認します。

3. CPU 側がコールする数字を決めます。
   これは簡単でプレイヤー側の考えられる 3 桁の数字は try[720] の中の 3 桁の数字なのです。よって、その 3 桁の数字が出るまで for 文を繰り返すことにより CPU 側がコールする 3 桁の数字を作ることができます。

4. 2,3 を繰り返すことによりプレイヤー側の数字を当てることができます。

eat 関数

```
int eat(int a,int b,int c,int i){
    int try_dev[3];
    int eat=0;
    try_dev[0]=try[i]/100;
    try_dev[1]=(try[i]/10)%10;
    try_dev[2]=try[i]%10;

    if(a==try_dev[0]){
        eat+=1;
    }
    if(b==try_dev[1]){
        eat+=1;
    }
    if(c==try_dev[2]){
        eat+=1;
    }
```

```
    return eat;
}
```

---

eat() 関数は、例えば eat(attack[0],attack[1],attack[2],i) というように記述します。これは attack[0]〜[2] と try[i] を比べて、その eat 数の値を返してくれる関数です。

bite 関数

```
int bite(int a,int b,int c,int i){
    int d[3]={a,b,c};
    int p,q;
    int try_dev[3];
    int bite=0;
    int bite_temp=0;
    try_dev[0]=try[i]/100;
    try_dev[1]=(try[i]/10)%10;
    try_dev[2]=try[i]%10;

    for(p=0;p<3;p++){
        for(q=0;q<3;q++){
            if(d[p]==try_dev[q]){
                bite_temp=bite_temp+1;//eat または bite 数のカウント
            }
        }
    }
    bite=bite_temp-eat(a,b,c,i);//bite 数の割り出し

    return bite;
}
```

---

bite() 関数は、例えば bite(attack[0],attack[1],attack[2],i) というように記述します。これは attack[0]〜[2] と try[i] を比べて、その bite 数の値を返してくれる関数です。

3.2.5 使用したライブラリ

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

3.3 制作を終えて・CPU と戦った感想

このソースコードを書くにあたって最も苦労した部分は CPU がいかにしてプレイヤーの数字を当てるかということでした。いろいろ思案した結果として実装したのが ai_fibonacci 関数でした。この関数は通常、人がこのゲームで数字を当てるのとは異なる当て方をしているため、実際にゲームをしてみるまでその難易度は未知数でした。実際にゲームをしてみると CPU は平均 6 手でこちら側の数字を当てており、ゲームレベルとしてはノーマル程度になったかと思います。

尚、実際のソースにはゲーム進行のログをとるための関数、ゲームが正常に作動していることを確認するため、CPU の数字を表示する関数等ここでは紹介出来ていない関数が存在しますが、紙面の都合上割愛させていただきます。

3.4 注

本文中の会社名、番組名等は各社の標章、登録標章であるものがあります。

4 テトリス

情報工学課程 1 回生 前田 竜輝

4.1 テトリス

4.1.1 はじめに

今回私は、テトリスを作りました。最初は、ウィンドウアプリケーション (GUI アプリケーション) を開発しようとしていましたが、コンソールアプリケーションに途中で変更しました。開発環境は Microsoft の VisualStudioExpress2012 開発言語は C 言語を用いて、主に「Windows プログラミング研究所 (URL は参考文献に記載)」を参考にしてつくりました。また、ソースコードが長くなってしまったため、要点に絞って書いています。

4.1.2 大まかな流れ

まず、最初に落ちてくるピースの生成をします。生成が終わると次に printf 関数でフィールドがコンソールに描かれます。その後ピースを移動させたり、回転させるたびにピースの位置情報を更新してから、さいど printf 関数でフィールドすべてを描画しなおします。そして、ピースが落ちきるとラインが消えるかの判定を行った後、次のピースを出します。主な流れはこれの繰り返しとなります。

4.1.3 ピースの生成

ピースは以下のように定義されています。

```
BYTE piece[4][4]={0};       //移動中のピースの表示
BYTE next1[4][4]={0};      //次のピース
BYTE next2[4][4]={0};      //2 番目のピース
BYTE next3[4][4]={0};      //3 番目のピース
BYTE hold[4][4]={0};        //ホールド中のピース
```

ピースはこのように 4 × 4 の二次元配列を定義し 4 × 4 のマスとして考え、どこが埋まっているかを定義し 7 種類のピースを生成しています。

実際にコードとして書くとこのようになります。

```
switch(rand()%7){
    case 0:
        next[1][1]=1; next[2][1]=1; next[1][2]=1; next[2][2]=1;
        break;
    case 1:
        next[1][0]=1; next[1][1]=1; next[1][2]=1; next[1][3]=1;
        break;
            …
            …
 }
```

このように 0〜6 の乱数を生成し、ランダムに 7 種類のピースのうちどれかひとつを生成するようになっています。たとえば case 0:が選択されたとすると空きマスを□埋まったマス (1 が代入されたマス) を■として書くと

```
  0 1 2 3
0 □□□□
1 □■■□
2 □■■□
3 □□□□
```

となり、正方形のピースが生成されます。生成されたピースはまず next3(3 番目のピースの配列) に代入され、その後 NextPiece 関数で next3 → next2（2 番目のピースの配列）→ next1(次のピースの配列) → piece(移動中のピースの配列) の順で回されます。これにより、次に出る 3 つ目までのピースを表示できるようにしています。

4.1.4 ピースの移動

移動中のピースのフィールド上での座標は次のように定義されています。

```
POINT location={0,0};    //piece[0][0] のフィールド上の座標
```

ピースの移動は MovePiece 関数でこの location の値を変えることで行います。また、このとき location の 2 つの値は location.x、location.y として定義されています。

```
BOOL MovePiece(int move){
    int x,y,left,right,bottom;
    switch(move){
        //ピースを左に 1 マス動かす
        case PIECE_LEFT:
            left=GetPieceLeft();
            if((location.x)+left<=0) return FALSE;
            for(y=0;y<PIECE_HEIGHT;y++){
                for(x=0;x<PIECE_WIDTH;x++){
                    //左にブロックまたは壁がある
                    if(piece[x][y] && (location.x)+x-1>=0
                        && (location.y)+y>=0
                        && field[(location.x)+x-1][(location.y)+y])
                           return FALSE;
                }
            }
            location.x--;
            return TRUE;
            //ピースを右に 1 マス動かす
        case PIECE_RIGHT:
            right=GetPieceRight();
            if((location.x)+right>=FIELD_WIDTH-1) return FALSE;
            for(y=0;y<PIECE_HEIGHT;y++){
                for(x=0;x<PIECE_WIDTH;x++){
                    //右にブロックまたは壁がある
                    if(piece[x][y] && (location.x)+x+1<=FIELD_WIDTH
                        && (location.y)+y>=0
                        && field[(location.x)+x+1][(location.y)+y])
                            return FALSE;
                }
            }
            location.x++;
```

```
            return TRUE;
            //ピースを下に 1 マス動かす
        case PIECE_DOWN:
            bottom=GetPieceBottom();
            if((location.y)+bottom>=FIELD_HEIGHT-1) return FALSE;
            for(y=0;y<PIECE_HEIGHT;y++){
                for(x=0;x<PIECE_WIDTH;x++){
                    //下にブロックまたは壁がある
                    if(piece[x][y] && (location.y)+y+1>=0
                        && (location.y)+y+1<FIELD_HEIGHT
                        && field[(location.x)+x][(location.y)+y+1])
                            return FALSE;
                }
            }
            location.y++;
            return TRUE;
    }
}
```

Movepiece 関数はキー操作や自動落下の際に呼び出され、どの方向にピースを動かすかによって分岐します。それぞれ動かす方向にブロックまたは壁がなければ location の値を変えて TRUE を返し、ブロックまたは壁がある場合は location の値を変えず (ピースを動かさず)FALSE を返します。また、case PIECE_DOWN:で FALSE を返されると、それ以上は落とせないということなので、ピースが固定され次のピースが出現します。ハードドロップ (ピースを一番下まで強制的に落とす操作) は、FALSE が返ってくるまで MovePiece(PIECE_DOWN) を繰り返し呼び出すことで行っています。

4.1.5 ピースの回転

ピースの回転は TurnPiece 関数によって行われます。しかし、ただピースを回転させるだけだと回転後にピースの一部が壁に埋まってしまったり、フィールド上の他のブロックと干渉してしまう場合があります。そのような場合本家テトリスでは右や左や上にピースをずらして回転させます。そのためこのようにしました。

```
int TurnPiece(){
    int x,y,offsetX,offsetY;
    BYTE pTurn[PIECE_WIDTH][PIECE_HEIGHT];
    //pTurn に回転したピースを代入
    for(y=0;y<PIECE_HEIGHT;y++){
        for(x=0;x<PIECE_WIDTH;x++){
            pTurn[(PIECE_HEIGHT-1)-y][x]=piece[x][y];
        }
    }
    //回転可能かどうかの判定
    for(y=0;y<PIECE_HEIGHT;y++){
        for(x=0;x<PIECE_WIDTH;x++){
            if(pTurn[x][y]){
                offsetX=(location.x)+x;
                offsetY=(location.y)+y;
                if(offsetX<0) return 1;
                else if(offsetX>=FIELD_WIDTH) return 2;
                else if(offsetY>=FIELD_HEIGHT
                    ||offsetY>=0&&field[offsetX][offsetY])
```

```
                        return 3;
                }
            }
        }
        //回転したピースを piece に代入
        for(y=0;y<PIECE_HEIGHT;y++){
            for(x=0;x<PIECE_WIDTH;x++){
                piece[x][y]=pTurn[x][y];
            }
        }
        return 0;
}
```

まず回転させたピースの状態を新たに定義した pTurn に代入し、回転可能かどうか (回転後壁に埋まったり、他のブロックと重なったりしないか) を確認します。そして、回転可能であれば 0、左壁と重なる場合は 1、右壁と重なる場合は 2、フィールドの下端またはフィールドのブロックとかさなる場合は 3 を返します。そして、main 関数内ではこのように処理されます。

```
case ' ':  //スペースが入力された
    first:
    ref=TurnPiece();
    if(ref==1){          //左壁と重なる
        location.x++;    //ピースを右に 1 マスずらす
        goto first;
    }
    else if(ref==2){     //右壁と重なる
        location.x--;    //ピースを左に 1 マスずらす
        goto first;
    }
    else if(ref==3){     //下端または他のピースと重なる
        location.y--;    //ピースを上に 1 マスずらす
        goto first;
    }
    break;
```

まず返し値の ref を得てたとえば、返し値が 1 ならば右壁と重なっているのでピースを右にひとつずらして、再度返し値を得ます。返し値が 2 や 3 の場合も同様に処理して、返し値が 0 になるまでこれを繰り返します。これにより本家テトリスと同様の動きができます。

4.1.6 埋まったラインの削除

埋まったラインの削除は DeleteLine と ShiftLine の二つの関数で行います。まず DeleteLine 関数で埋まったラインを空にします。

```
//埋まった行の削除
int DeleteLine(){
    int x,y,delCount=0,lineCount;
    for(y=FIELD_HEIGHT-1;y>=0;y--){
        lineCount=0;
        //一番下から 1 列にいくつブロックがあるかを数えていく
        for(x=0;x<FIELD_WIDTH;x++){
```

```
            lineCount+=field[x][y];
    }
        //1列にブロックがひとつもないとき
        if(lineCount==0) break;
        //ブロックはあるがすべて埋まっていないとき
        if(lineCount!=FIELD_WIDTH) continue;
        //1列すべて埋まっているとき
        delCount++;
        levelCount++;
        //埋まっている列を消す
        for(x=0;x<FIELD_WIDTH;x++){
            field[x][y]=0;
        }
    }
    return delCount;
}
```

DeleteLine 関数はフィールドの一番下から一列ごとにいくつブロックがあるかを数えていきます。1列にブロックはあるがすべて埋まっていないときは、そのまま処理を続行します。1列すべてブロックで埋まっていれば delCount を＋1してその列を空にします。1列にひとつもブロックがない場合はゲーム上その上にブロックは存在し得ないのでそこで処理を終了し、delCount(消したラインの数) を返します。DeleteLine 関数の処理が終わると埋まった行のブロックが消え空行になっているのでその分を ShiftLine 関数で詰めます。

```
//削除した行を詰める
void ShiftLine(int delCount){
    int x,y,lineCount=0;
    for(y=FIELD_HEIGHT-1;y>=0 && delCount>0;){
        lineCount=0;
        for(x=0;x<FIELD_WIDTH;x++){
            lineCount+=field[x][y];
        }
        //空行ではない
        if(lineCount!=0){
            y--;
            continue;
        }
        //空行なので詰める
        delCount--;
        for(int iy=y;iy>=0;iy--){
            for(x=0;x<FIELD_WIDTH;x++){
                if(iy-1>=0){
                    field[x][iy]=field[x][iy-1];
                }
                else{
                    field[x][0]=0;
                }
            }
        }
    }
}
```

ShiftLine 関数は DeletLine 関数の返し値 delCount(消したラインの数) を受け取り、DeleteLine 関数と同様にフィールドの下端から 1 列ごとにブロックの数を数えていきます。 1 列にブロックがひとつもなければその行は消した行なのでそこから上を詰めて、delCount を- 1 します。これを繰り返して delCount が 0 になれば消した行の分詰めたことになるのでそこで処理を終了します。

4.1.7 自動落下

自動落下の処理は main 関数で行われます。

```
    …
    …
progress=GetTickCount()-beforeTime;
if(progress>=sleep){
    if(!MovePiece(PIECE_DOWN)){
        PieceToField();
        delcount=DeleteLine();
        ShiftLine(delcount);
        beforeTime=GetTickCount();
    …
    …
```

このように GetTickCount 関数で現在時刻を取得し、前回の自動落下から一定時間が経過すると MovePiece 関数を呼び出して、ピースを 1 マス落とします。そこで一番下まで達していればピースを固定し、ライン削除の判定処理をします。そして、再度現在時刻を取得し、次の自動落下の際の経過時間算出の目安とします。

4.1.8 キー入力と自動落下関数の組み合わせ

今回キー入力の関数にはエコーバックなし (コンソールに文字を表示せず、キーを押すと即その文字を取り込む) の getch 関数を使用しました。しかし、この関数はキーが押されるまでずっとキー入力待ち状態で待機してしまうため、キー入力待ち状態の間自動落下関数を呼び出せず、一定の間隔でピースを落とせなくなってしまいます。そこで、このようにしました。

```
//キー入力があるまで繰り返し
…
…
while(1){
    if(kbhit()){
        key=getch();
        goto move;
    }
    progress=GetTickCount()-beforeTime;
    if(progress>=sleep){
        …
        …
    }
}
//キー入力があるとここに飛ぶ
move:
switch(key){
    //左へ移動
```

```
    case 'a':
        MovePiece(PIECE_LEFT);
        break;
    //下へ移動
    case 's':
        …
        …
}
```

kbhit 関数はキー入力があったかどうかを判定する関数で、それを if 文に組み込みキー入力があったときだけ getch を呼び出すようにしました。キー入力があるとその文字を受け取って switch 文へ飛び、入力されたキーに応じた動作を行います。キー入力がない間は自動落下の処理を繰り返し、一定時間ごとにピースを落下させることができます。

4.1.9 おわりに

今までは基礎を学ぶための特に意味のないプログラムしか書いたことはなく、今回初めて目的を持ったプログラミングをして、いろいろ苦労もありましたが、1 つ形のあるものを作ることができてとても楽しかったですし、モチベーションもあがりました。しかし、もともとの目標だったウィンドウアプリでの作成はできずじまいで、まだまだ搭載したい機能もたくさんあるのでこれからも改良を加えていきたいと思っています。

参考文献

[1] Windows プログラミング研究所
http://www13.plala.or.jp/kymats/

[2] パソコン活用研究Ｃ＆Ｃ＋＋であそぼう
http://www.geocities.co.jp/SiliconValley-Bay/7437/index.html

5 コイン落としゲーム

情報工学課程 1 回生 山口 琴子・高橋 ともみ

5.1 はじめに

私たちはもともとゲームプログラマーという職業に興味があったため情報工学部に入ったので、この夏休みには以前からやってみたかったゲーム製作に挑戦したいと思い、Unity というゲームエンジンで”コイン落としゲーム”を作成することにしました。Unity は物体の動きを自動で計算してくれるゲームエンジンです。しかし、Unity に対応する言語は C ＃や JavaScript などでした。本当は、前学期にコンピュータ部の勉強会で学んだ C 言語を用いてゲーム作成をする予定だったのですが、C 言語以外のプログラミング言語もいずれ習得したいと思っていたので、今回はテーマを、「ゲーム製作を通じて新しい言語や Unity というゲームエンジンに触れる」ということに変更しました。そこで、一人は JavaScript、もう一人は C ＃でプログラムを書きほぼ同じゲームを作りました。初めて触れるエンジンや言語ということもあり、主にサイトや動画を参考にコードを書き、意味を理解し、理解したことを利用してデザインや新たな動きなどを作成しました。

5.2 製作の流れ

1. 物体の配置

   (a) 土台はじめの状態では画面には何もないので、まずは土台を作りました。
   GameObject(ゲーム中の全ての物体はここから選択して配置します。) の中の CreateOther から cube を選択、変形させ、配置し、base と名づけます。これを実体化させるために、Component の中の Physics から Rigidbody を選択し、さらに、そのまま無限に下に落ちていくことを防ぐために、Is Kinematic にチェックを入れます。

   (b) ライト
   このままだと暗いので、ライトを設置します。GameObject の中の CreateOther から Directional light を選択、配置します。これで明るくなりました。

   (c) 壁
   次に土台を囲う壁を 3 枚作ります。いずれも土台と同様の作業を行います。

   (d) プッシャー
   続いて、コインを押すプッシャーを作りました。これも土台と同じ作業を行います。

   (e) コイン
   次にコインを作ります。
   GameObject の中の CreateOther から Cylinder を選択し、薄くすることでコインの形にします。さらに実体化させるために、Rigidbody を追加します。さらに、当たり判定を球状からコイン型にするため、Component

の中の Physics から MeshCollider を選択し、CapsuleCollider と置き換えます。この MeshCollider は処理が重いので、convex にチェックを入れます。
作ったコインをプレファブ化することで、複製できるようにし、複製したコインを適当な位置に配置します。

(f) コインの発生場所
コインが出てくる場所を設置します。土台の作成と同じように作ります。

(g) 手持ちコインの枚数表示
GameObject の中の CreateOther から GUIText を選択します。FontSize を調整します。これで画面上に文字が表示されます。

(h) コインの消去
一定の位置に達したコインを消去するために、GameObject から CreateEmpty を選択し、見えないボックスを配置します。

(i) 着色
Project ビューの中の Material を選択し、好きな色に変えて、色を変えたい物体にD＆Dすれば色が変わります。

(j) その他加工 Project ビューの中の Physic Material を選択し、Bounciness の値や Friction Combine を調整します。これによって、より実際の状況に近づけることができます。

2. スクリプト (ver.JavaScript)

(a) プッシャーについて
プッシャーを動かすためのソースコードです。

```
var origin : Vector3;

function Awake(){

    origin =rigidbody.position;
}

function Update () {

  //プッシャーに z 軸方向にのみ Sin 関数の動きをさせるために offs を定義。
  var offs = Vector3(0,0,Mathf.Sin(Time.time));

  rigidbody.MovePosition(origin + offs);  //起点を保存したまま上の動きをさせる。

}
```

(b) コインの発生について
クリックされるたび、位置を変えながらコインを出現させるためのソースコードです。

```
var prefab : Transform;


function Update (){
```

```
    if(Input.GetButtonDown ("Fire1")) {

    //コインの出現位置に x 軸方向にのみ Sin 関数の動きをさせるために定義。
    var offs=Vector3(Mathf.Sin(Time.time * 2),0,0);

    //起点を保存しつつ、位置を変えながら prefab を複製させる。
   Instantiate
     (prefab,transform.position
                      + offs,transform.rotation);

    //score スクリプト中で定義された score の値を-1 する、つまり、落とすたびに手持ちコインを 1
減らす。
    score.score - = 1;

    }
}
```

(c) コインの発生口の移動

```
var origin : Vector3;

function Awake(){

   origin = rigidbody.position;  //起点を定義。

   }

function Update () {

   //コインの発生口に見立てたブロックを、コインの出現位置の動きに合わせて移動させるために offs
を定義。
   var offs = Vector3(Mathf.Sin(Time.time * 2),0,0);

   rigidbody.MovePosition(origin + offs);  //起点を保存したまま上の動きをさせる。

   }
```

(d) 手持ちコインの枚数表示について
手持ちコインの枚数を表示するためのソースコードです。

```
static var score : int ;

function Awake () {
```

```
    score = 15;  //ゲーム開始時のコイン枚数を 15 枚に設定。
    }

function Update () {
    //"YOUR COIN : 手持ちコインの枚数"を表示。
    guiText.text = "YOUR COIN : " + score.ToString();

    if(score<=0){
        //score の値、つまり、手持ちコインの枚数が 0 未満になったとき、"GAME OVER..."と表示する。
        guiText.text ="GAME OVER...";
        }
    }
```

(e) 落ちたコインの処理について
落ちたコインを消去し、手持ちコインの枚数に加えるためのソースコードです。

```
function OnTriggerEnter (other : Collider) {

    Destroy(other.gameObject);  //落ちたコインが remover に当たったとき、そのコインを消去する。
    score.score +=1;  //上の処理が行われるごとに、手持ちコインの枚数に 1 加える。

    }

function Update () {

    }
```

3. スクリプト (ver. C #)

(a) プッシャーについて

```
using UnityEngine;
using System.Collections;

public class Pusher : MonoBehaviour {
 z
private Vector3 origin;

// Use this for initialization
void Start () {

origin = rigidbody.position;//起点を保存しておく
}
```

```
// Update is called once per frame
void Update () {


    //z 軸方向に一定の速度で前後運動させる
   Vector3 offset
          =new Vector3 (0,0,Mathf.Sin (Time.time));


    rigidbody.MovePosition (origin + offset);
    //起点を保存したまま動かす

}
}
```

(b) コインの発生について

```
 using UnityEngine;
using System.Collections;

public class SpawnerScript : MonoBehaviour {
public Transform prefab;

void Update(){
if(Input.GetButtonDown("Fire1")){//click
//x 軸方向に一定の速度でコイン生成口を運動させる ((y,z)=(16,2))
Vector3 offset
      =new Vector3(Mathf.Sin(Time.time*4),16,2);

  //コインを生成
  Instantiate
  (prefab, transform.position
                  + offset,transform.rotation);

  //前に記述した ScoreScript の score の値をボタンを押した回数分-1 していく
          ScoreScript.score--;
}
}
}
```

(c) コインの発生口の移動

```
 using UnityEngine;
using System.Collections;

public class cubeScript : MonoBehaviour {
private Vector3 origin;

// Use this for initialization
void Start () {

origin = rigidbody.position;
}

// Update is called once per frame
void Update () {
   Vector3 offset
        =new Vector3 (Mathf.Sin (Time.time*4),0,0);
rigidbody.MovePosition (origin + offset);

}
}
```

(d) 手持ちコインの枚数表示について

```
 using UnityEngine;
using System.Collections;

public class ScoreScript : MonoBehaviour{

public static int score;

void Awake(){
score=10;//初期のスコアの値
}

void Update(){
    //「SCORE:数値」を画面上に出力
guiText.text="SCORE:"+score.ToString();
if(score=<0){
guiTeXt.text="GAME OVER"//もし、score の値が 0 になったら、GAMEOVER と表示
}
}
```

(e) 落ちたコインの処理について

```
using UnityEngine;
using System.Collections;

public class Remover : MonoBehaviour {

// Use this for initialization
void OnTriggerEnter(Collider collider){
Destroy(collider.gameObject);//受け取り口にコインが入ったらをコインを消去

//前に記述した ScoreScript の score の値を入った回数分+1 していく
        ScoreScript.score++;
}
}
```

5.3 おわりに

今回のゲーム製作を通じて、さまざまなスクリプトの例を見ることで JavaScript や C ＃に少しは慣れることができたように思います。今後は書籍を読むなどして確実な知識を身につけ、いずれは今回学んだ言語を用いて、自身で一からゲーム製作をしたいと考えています。

参考文献

[1] http://www.slideshare.net/MakotoYamahira/unity1unity

[2] http://d.hatena.ne.jp/ke-16/20120113/1326450644

[3] http://www.nicovideo.jp/watch/sm12948504

[4] ttp://www.slideshare.net/takuya0429/unitydena-20121130

6 パソコンを動かす要素

電子システム工学課程 1 回生 多田 晧之

6.1　始めに

　この文書は今まで当たり前のように使っていたパソコンの性能や機能を生み出すいくつかの部品を一覧という形で紹介していきます。

6.2　マザーボード

概要　その名のとおり母のようにすべてを受け入れてくれるパーツです。
　この基盤にＣＰＵやメモリーなど、この文書にあるようなパーツがつくことになります。
　逆にいうと、マザーボードによってつけられるパーツも違ってくるので、後々の拡張性などの面で一番基本となるパーツであるということです

・各部の役割

- ＣＰＵソケット
  ＣＰＵを取り付けるところです。ソケットにもいろいろな種類があり、インテル系ならＬＧＡ１１５０ LGA2011、ＡＭＤ系ならやＡＭ３＋,FM2 が有名どころです。

  ちなみに購入した際このソケットにソケットカバーがついてきますが、もし不具合があって返品する際、ソケットカバーも必要になることがあるので、間違えて捨ててしまわないように注意しましょう。

- メモリスロット
  メモリを差し込むところです。ここもメモリスロットの種類によって 差し込むことのできるメモリが変わってきます。

  メモリスロットの種類についてはメモリーの種類と対応しており、DDR　SDRAM（ディーディーアールエスディーラム） DDR ２　SDRAM（ディーディーアールツーエスディーラム） DDR ３　SDRAM とあります。

  現在主流は DDR ３　SDRAM です。

- ＰＣＩスロット
  ビデオボードやサウンドボードなど拡張カードを差し込むところです。もうかれこれ１０年以上も前から使われています。最大転送速度は１３３ＭＢ／秒。

- ＰＣＩエクスプレス
  拡張カードを差し込むところです。一昔前にはＡＧＰスロットという拡張カードスロットもありましたが、現在はＡＧＰスロット自体ないマザーボードも増えてきています。

  ですので、現在の主流はこのＰＣＩエクスプレス×１６です。最大転送速度はデータ送信４ＧＢ／秒、データ受信４ＧＢ／秒で、合計８ＧＢ／秒。

- ＩＤＥインターフェース
  ハードディスクや光学ドライブなどのＩＤＥケーブルを接続するところです。 別名 ”Ultra ATA ”や ”E-IDE ”とも呼ばれることがあります。

  コネクタ側とケーブル側の切り欠きをあわせて接続します。ＡＴＡ／１００（最大転送速度は１００MB／秒）とＡＴＡ／１３３（最大転送速度は１３３MB／秒）の２種類の規格があります。

  現在では主流になっているＳＡＴＡに移行しています。

- シリアルＡＴＡインターフェース

  シリアルＡＴＡのケーブルを持つハードディスクや光学ドライブを接続するところです。ＳＡＴＡ（サタ）とも呼ばれます。

  これにおいて一般的だったＳＡＴＡ２の最大転送速度が３Ｇｂｉｔ／秒となっていて、最大転送速度が１３３MB／秒のＩＤＥよりさらに高速です。次世代のＳＡＴＡでは最大転送速度が６Ｇｂｉｔ／秒となっており、これから自作する方はシリアルＡＴＡで接続した方が速いです

- ＦＤＤコネクタ
  フロッピーディスクドライブのケーブルを差し込むところです。ＩＤＥのインターフェースと似ていますが、ＦＤＤの方が少しサイズが小さいです。

  ＦＤＤケーブルもＩＤＥケーブルと似ていますが、ＦＤＤケーブルは一部がねじれているという違いがあります。何十年後かにはＦＤＤコネクタもなくなっていくでしょうね

- チップセット
  チップセットは基盤上に接続される各部分をコントロールする重要な部分です。

  現在のチップセットは１つです。以前は２つあり、CPU に近い方からノースブリッジ、サウスブリッジと呼ばれていました。

  それぞれコントロールする部分が違い、ノースブリッジはＣＰＵ、メモリ、ＰＣＩエクスプレス、ＡＧＰなどをコントロールしていました。サウスブリッジは シリアルＡＴＡ、ＰＣＩスロット、ドライブ類、各種Ｉ／Ｏポートなどをコントロールしていました。

  これらは発熱するので、ヒートシンクや冷却ファンがついているものが一般的です。このチップセットの機能で、マザーボードの機能が決まるいうことになります。

- 主電源２４ピン

  電源からマザーボードへ電源を供給するメインのコネクタです。一昔前は２０ピンのコネクタが主流でしたが、現在は２４ピンが主流です。

  電源コネクタの中には２４ピンの４ピンだけ分割できて、両方のコネクタに対応しているものもあります。この主電源のみならず８ピンのＣＰＵ補助電源と

  も存在する

- ＣＰＵファン用電源

  冷却用のファンの電源ケーブルを接続するところです。マザーボードによって配置されている個数は異なります。

- バッテリー

  ＢＩＯＳの設定を保存するためのバッテリーです。このバッテリーがあるおかげで、電源ＯＦＦのときも時計の時刻を刻んでくれたりします。

バッテリーの電池切れのときは時計はリセットされてしまうことが多いです。ちなみにＢＩＯＳ設定を間違えたときなんかは、このバッテリーを一度取り外せ

ば初期設定にもどせます。

- ＢＩＯＳ－ＲＯＭ（ＣＭＯＳ）

  ＢＩＯＳの設定や情報などが記憶されています。パソコンが起動できるのはこのＲＯＭがあるおかげです。

  またＢＩＯＳのアップデートなどをする場合は、このＲＯＭの中のＢＩＯＳ情報をアップデートするということになります。アップデートを失敗したときに備えて、２つのＲＯＭを搭載しているものもあります。

6.3 ハードディスク

ハードディスクとはファイルやデータを記憶保存しておくところです。ここはパソコンで見ることのできるすべてのデータを保存しています。

あなたの家にもあるであろうパソコンの画面も、インターネットサーバーからあなたのパソコンのハードディスクに一時的に保存されているから映っていると

いうことです。（ただ、画面出力自体はビデオカードの仕事です。）

一度データをいれると消去や壊れない限りずっと記憶しておいてくれます。 むちゃくちゃ記憶力のいい人っていうイメージですね

- 磁気ヘッド

  磁気ディスクにデータを読み書きする部分です。

  ディスクとのすき間はなんと！10 数 nm（ナノメートル）！１ｎｍ＝１０億分の１メートル＝１００万分の１ミリメートルですので、 目に見えないほこりよりさらに小さい隙間といえます！

  ハードディスクに衝撃を与えると、磁気ヘッドの先が磁気ディスクに触れて傷をつけ、データエラーを起こしてしまうので、やさしく扱わないといけないというわけです。

- アクセスアーム

  アクチュエーターとも呼ばれ、磁気ヘッドの動きはこのアクチュエーターによって作られます。

  ディスクとのすき間は ディスクが回転することによって生じる風圧によってつくられるという、これまた極小のすき間です。

- ステッピングモーター

  磁気ヘッドやアクセスアームを要求されているディスクの位置（トラック・セクター）にもっていく為のモーターです。かなり精密な動きをする高性能モーターです

- スピンドルモーター

  ディスクを回転させる為のモーターです。７２００ｒｐｍや１００００ｒｐｍなどがありますが、回転数はこのモーターによります。ディスクがブレてしまわないように かなりの高い精度をもったモーターです。

6.4 ＣＰＵ

ＣＰＵとは中央演算処理装置（ちゅうおうえんざんしょりそうち）とも呼ばれ、頭を回転させることが仕事です。いうばればパソコンの頭脳です。

- ＣＰＵコアヒートスプレッダーの裏にある。 ＣＰＵの要ともいえる演算部分です。 デュアルコアではコアが２つ、クアッドコアでは４つあります。現在では６つのヘキサコア、８つのオクトコアもあります。どれも分業による処理速度の向上が目的です。
- ヒートスプレッダーＣＰＵコアを保護するとともに、 熱をヒートシンクに伝える部分です。 デスクトップＰＣ用ＣＰＵでは昔はこれがなかったので、 コアが破損してしまうことがよ

  く、ありました。 （コア欠け）
- ＬＧＡパッケージ

  とげとげのピンではなく、 平らな金属接点がたくさんついている形状のパッケージ形式のこと。 ２０１３年現在では、ＬＧＡ１１５０,LGA2011 が主流。

  LGA1155、LGA1156、LGA1366、は旧式になっている。
- ランド丸くて平たい点々のこと。 マザーボード側のＣＰＵソケットに とげとげのピンがでている。

6.5 メモリー

- 接続端子

  メモリーをマザーボードに接続する部分です。デスクトップＰＣ用のメモリであれば DDR3-SDRAM DIMM の２４０ pin（ピン）。 ノートＰＣ用のメモリであれば DDR3-SDRAM SO-DIMM ２０４ pin（ピン）が主流です。最近では メモリーのデータ転送を安定させるために設置されています。
- 切り欠き

  メモリーをマザーボードに接続したときに メモリーの種類や方向を挿し間違えないためにあります。

- ＳＰＤ（Ｓｅｒｉａｌ　Ｐｒｅｓｅｎｃｅ　Ｄｅｔｅｃｔ）

  メモリーチップの種類・容量・動作周波数などの情報を記憶している部分です。パソコンを起動させると、マザーボードのＢＩＯＳがメモリーのＳＰＤの情報を読み込んで、きちんと使えるようにしてくれます。

  メモリー起動の基本情報が入っているので、メモリー版ＢＩＯＳといったところです。

- スタブ抵抗

  データを高速でやりとりする際に発生する信号のノイズを抑制する部分です。一列に並んでいますね。大昔のメモリではこれがなかったのですが、ＤＤＲになってからはメモリーのデータ転送を安定させるために設置されています。

6.6 光学ドライブ

光学ドライブとはカンタンにいいますと、ＣＤやＤＶＤなどを再生や書き込みなどができる装置です。CD ドライブや DVD ドライブともいいます。

- コンボドライブ

  コンボドライブとは、CD と DVD の双方に対応しているものを言います。まあ最近のパソコンのドライブはほぼ全て CD と DVD の双方に対応していますから、わざわざ「コンボドライブ」と呼ばれることも少なくなりつつありますね。

- マルチドライブ

  DVD-R、DVD-RW、DVD-RAM の３種類に対応したドライブを言います。まだ DVD+R/RW の規格がなかった頃は、これが最多対応のドライブでした。もちろん、これ以降のドライブは CD にも対応しているのが普通です。

- スーパーマルチドライブ DVD-R/-RW/RAM に加え、さらに DVD+R/+RW にも対応した DVD ドライブで、+R/+RW の規格が登場して以後に現れました。表記では、「DVD ± R/RW/RAM」と、±で表記される事も多いです。

- ハイパーマルチドライブスーパーマルチに加えてさらに DVD-RDL や DVD+RDL にも対応しているもので、現在最多対応のドライブと言えます。

6.7 ＳＳＤ

ＳＳＤ（ソリッド・ステート・ドライブ）は、次世代のＨＤＤ（ハードディスク）ともいわれ現在とても注目されています。

役割はＨＤＤと同じで、ファイルやデータを記憶保存しておくところです。ＳＳＤはＨＤＤと構造が全く違い、ＨＤＤにはなかったメリットがたくさんあります。

- SLC

SSD の中にはデータを保存するスペース（セル）がたくさん用意されているのですが、ＳＬＣは１つのセルに１ビット（１単位）の情報のみを書き込むができます。そのため、SLC の方がシステムが単純な分、高速で、耐久性が高く、消費電力も少なくてすみます。

- MLC

ＳＳＤの中で、１つのセルに２ビット以上（２単位以上）の情報を書き込むのが MLC です。MLC の方がデータをたくさん保存する事ができ、それにより容量が大きくても価格が安くなります。ちなみに、SSD の最大の欠点はデータの記録容量が少ない事です。ですから容量を多くできる MLC の方が重宝がられており、技術開発も MLC の方が進んでいます。その結果、MLC でも SLC に負けないデータの読み込み・書き込み速度が出るようになりつつあります。

6.8 GPU

3D グラフィックスの表示に必要な計算処理を行う半導体チップ。従来 3D グラフィックスアクセラレータと呼ばれていたチップの発展形で、3D グラフィックスアクセラレータと比べて担当する処理が多くなっている。

- コアクロック　　ＣＰＵ のクロック数と同じで、グラフィックカードの中心にあるグラフィックチップ「GPU」のクロック数を表し、「コアクロック」とも呼びます。

参考文献

[1] http://www.pc-info.sakura.ne.jp/maza-kibankaisetu1.html

[2] http://www.pc-info.sakura.ne.jp/maza-kibankaisetu1.html

[3] http://homepage2.nifty.com/kamurai/index.htm

7 へびにょろ制作

電子システム工学課程 1 回生 林 拓哉

7.1 はじめに

前学期中に C 言語の基礎を学んだので、夏休みに簡単なゲームを一つ作ってみることにしました。

今回は、へびにょろと言うゲームを作りました。題材にするゲーム選びに悩みましたが、プログラミングを作るときにゲーム内容からプログラミングをイメージしやすそうなパズルゲームを選びました。

へびにょろ (正式名称は知らないです) は、正方形のマス、連なったブロック（蛇)、ランダムな位置に現れるブロック（エサ）からなります。

プレイヤーは、前進し続ける蛇をエサの位置まで操作しエサを食べさせます。そのエサを 1 回食べるごとに蛇の体は 1 ブロック分長くなり、新しく現れるエサに対して続けることでどんどん長くなっていきます。蛇の頭が、壁や蛇自身の体に当たるとゲームオーバーになります。ゲームオーバーになるまでの蛇の長さが点数となります。

ゲーム製作には、無料で公開されているコンピューターゲーム開発用ライブラリである DX ライブラリを使用しています。

用意する素材は 70 × 70 ピクセルの画像 4 種類（マス、カベ、ヘビ、エサ用）です。

7.2 プログラム

今回、ゲームのマス目は 9 × 9 にしました。プログラミング内では周囲の壁を合わせた 11 × 11 のマス目に、1 段目左 → 1 段目右→ 2 段目左→ 2 段目右 ･･･ 、の順番で、0〜120 の数字を割り当てています。ヘビの体の位置を格納する変数『block[81]』、エサの位置を格納する変数『esa』には、この数字が入れられます。

7.2.1 関数

それぞれの関数について簡単に説明します。

（main 関数）

Player_Initialize ･･･････ ゲームの初期化をする

while 文のループ- -
Player_End ····· 壁などにぶつかったか判定する
Player_Update_Get ···· ヘビがエサを手に入れたかどうか判定する
Player_Update_Go ····· ヘビが 1 マス進む処理をする
Player_Draw ····· ゲームを描写する
- -
Player_Finalize ····· 終了処理をする
———————————————————————————

上記以外にも、キーボードからの入力状態を得るための関数も用意しました。
(Keyboard_Update、Keyboard_Get)
この中から一部のコードを説明します。

7.2.2 エサの位置決め

Player_Initialize 関数、Player_Update_Get 関数にあります。
ゲームのスタート時とヘビがエサを取ったときに使います。変数「esa_rand」に GetRand(120) で取得した値を格納します。(DX ライブラリを使用した場合 GetRand() は毎回異なる乱数を返します)
その値がヘビの体やカベの数字と一致していればもう一度違う値を格納しなおし、一致していなければその値を変数「esa」に代入します。(1 つめの for 文がヘビの体と一致したかどうか、2 つ目の for 文が壁と一致したかどうかを判定します。

```
int i;
int esa_y;
int esa_rand;

while(esa_y==0){
    esa_y=1;
    esa_rand = GetRand(120);

    for(i=0;i<80;i++){
        if(esa_rand==*(block+i)){
            esa_y=0;
        }
    }
    for(i=0;i<10;i++){
        if(esa_rand==i —— esa_rand==i*11+11 —— esa_rand==i*11+21 —— esa_rand==i+110){
            esa_y=0;
        }
    }
}
*esa=esa_rand;
```

7.2.3 ヘビの進行

Player_Update_Go 関数にあります。
ヘビの体を 1 マスずつ進める処理をします。ヘビの先頭（block[0]）と他の部分（block[1]〜）とで行う処理が異なります。
先頭以外の block[] は一つ先頭に近い block[] の値を代入することで一つ前に進みます。

一方先頭の場合、キー入力した方向によって進むマスが変わるので、方向を表す値を格納した変数「Direction」を使います。違う方向に進むときは、Keyboard_Get（）関数で現在のキーボードの状態を取得し、Direction の値を変えます。(Direction に格納する値は、上 ‥ 1、左 ‥ 2、右 ‥ 3、下 ‥ 4 です)

1 マスずつ進んでいくと体がどんどん伸びていくので、この処理の最初と最後に必要な長さより後ろの block[] に 0 を代入します。(たとえば現在の長さが 5 のときは、block[5] 以降の block[] は全て 0 になり、ゲーム内では左上の壁の中に隠れることになります。)

```
int i,j;

for(i=(*n);i<80;i++){
    block[i]=0;
}

for(i=80;i>0;i–){
    block[i]=block[i+1];
}

if(Keyboard_Get(KEY_INPUT_UP)>0){
    *Direction = 1;
}
if(Keyboard_Get(KEY_INPUT_LEFT)>0){
    *Direction=2;
}
if(Keyboard_Get(KEY_INPUT_RIGHT)>0){
    *Direction=3;
}
if(Keyboard_Get(KEY_INPUT_DOWN)>0){
    *Direction=4;
}

if(*Direction==1){
    block[0]-=11;
}else if(*Direction==2){
    block[0]-=1;
}else if(*Direction==3){
    block[0]+=1;
}else if(*Direction==4){
    block[0]+=11;
}

for(i=(*n);i<80;i++){
    block[i]=0;
}
```

7.2.4 終了判定

Player_End 関数にあります。

ヘビの先頭が壁や体にぶつかったかどうかを判定します。1 つ目の for 文が体とぶつかったどうか、2 つ目の for 文が壁とぶつかったかどうかを判定します。

```
int i;
int end_x=0;

for(i=1;i<81;i++){
    if(block[0]==block[i]){
        end_x=1;
    }
}
for(i=0;i<10;i++){
    if(block[0]==i —— block[0]==i*11+11 —— block[0]==i*11+21 —— block[0]==i+110){
        end_x=1;
    }
}
```

この処理が書かれている Player_End 関数は end_x を戻り値にしています。

7.3 終わりに

今回作ったへびにょろは非常に単純なゲームであるにもかかわらず製作にてこずってしまい、プログラミングの難しさを改めて実感しました。また、作ったのはゲームとして動かすための最低限のものだけなので、今後はメニュー画面や記録のセーブなどを取り入れたより遊びやすいゲームの製作を目指そうと思います。

今回のゲーム製作において、DX ライブラリの導入方法や使い方、基本のコードなどが掲載されている『新・ゲームプログラミングの館』を参考にさせていただきました。

参考文献

[1] 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/）

[2] LaTeX コマンド集 （http://www.latex-cmd.com/）

8 Androidで数当てゲーム

電子システム工学課程 1 回生 森 龍太郎

8.1 はじめに

数当てゲームとは、コンピュータがランダムで設定した数字を当てるゲームです。自分の入力した数字が正解より大きいか小さいかをコンピュータが言ってくれるので、それをもとに正解を導き出していきます。今回は Android ということで言語は Java を使用しています。

8.2 内容

8.2.1 ゲーム画面

まず、ゲーム画面について説明します。今回のアプリの構成は以下のようにしました。

```
RelativeLayout
    -ScrollView
        -LinearLayout
        -TextView
    -Button
    -TextView
    -EditText
```

Android アプリのユーザーインターフェイスを構成する部品には、ビューやビューグループなどがあります。ビューグループはビューを子として持つことができ、上記での RelativeLayout, ScrollView, LinearLayout がビューグループにあたります。RelativeLayout は部品同士の相対的な位置関係を指定して配置、LinearLayout は部品を縦または横の直線上に配置するときに使います。ScrollView は中身が長くなった時にスクロールすることができます。次に、上記での TextView, Button, EditText はビューと呼ばれ、それぞれテキストを表示するビュー, ボタン, テキストを入力できるビューとなっています。

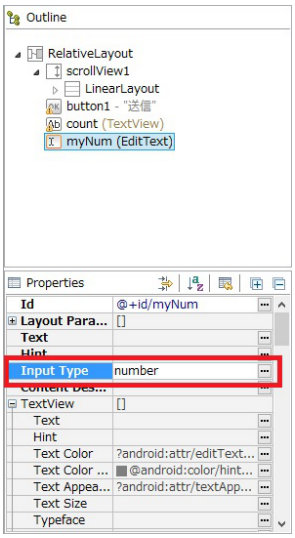
今回のゲーム内での使われ方を少し具体的に説明します。まず EditText に自分の思う答えを入力し、Button を押してその数字を確定します。ScrollView には、自分の入力した数字とそれに対する判定を表示していきます。また、EditText には数字しか入力できないようにレイアウト側から設定します。

長くなるので省略して書きます。

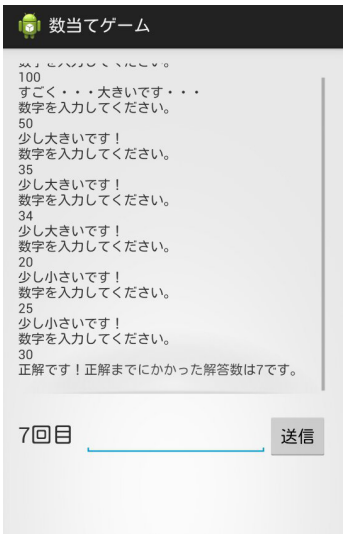
```
public class MainActivity extends Activity implements OnClickListener{
        int myAnswer;
        int cmpAnswer;
        int count;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button)findViewById(R.id.button1);
        btn.setOnClickListener(this);
        startGame();
    }省略

}
```

リスト 8.1: 起動直後に呼び出される onCreate メソッド



(a) 数字のみに指定



(b) ゲーム画面

ソース内の上から 5〜8 行目は Android アプリでの定番（？）の流れです。ここで少し onCreate について説明します。Android にはアクティビティのライフサイクルというものがあって、アプリが起動されるとまず onCreate() メソッドが呼び出されます。アクティビティとは簡単にいえばアプリケーションの画面です。アクティビティの状態が変化した時に呼び出されるメソッドがライフサイクルメソッドです。そのライフサイクルメソッドをオーバーライドして処理を記述します。それが 5 行目です。11 行目は activity_main という 8.2.1 で作成したゲーム画面を setContentView() でセットしています。11,12 行目でボタンをセットします。そして 13 行目で startGame() メソッドを呼び出しゲームを開始します。

```
1  private void startGame(){
2          cmpAnswer = 0;  //で初期化 0
3          createNum();
4          count = 0;  //初期化
5          input();  //に「数字を入力してください」と表示する ScrollView
6          nTimes();  //現在の解答数を表示する
7      }
```

```
1  private void createNum(){
2          Random rnd = new Random();
3          cmpAnswer = rnd.nextInt(100)+1;
4      }
```

リスト 8.2: ゲーム開始と乱数生成

3 行目で createNum() メソッドを呼び出し、下の createNum() メソッド内で変数 cmpAnswer に 1〜100 までのランダムの数字を代入しています。

```
1  @Override
2      public void onClick(View v){
3          editText();
4          count++;      //現在の解答回数を増やす 1
5          judge(myAnswer, cmpAnswer);
6      }
```

```
1  private void editText() {
2          EditText myNum = (EditText)findViewById(R.id.myNum);
3          Editable getText = myNum.getText();
4          myAnswer = Integer.parseInt(getText.toString());
5          TextView tvTalk = (TextView)findViewById(R.id.ScrollView);
6          if((getText.toString()).length() > 0 ){
7              tvTalk.append(getText.toString() + "\n");
8              myNum.setText("");
9      }
10 }
```

リスト 8.3: クリック時の挙動と文字入力

onClick() メソッドは、Button がクリックされた時に呼び出されます。3 行目で editText() メソッドを呼び出し入力された数字を確定し、現在の解答回数を表す count に 1 を加えます。そして myAnswer と cmpAnswer の値を引数として judge() メソッドを呼び出します。

下の editText() メソッドでは、4 行目で入力された数字を int 型に変換して変数 myAnswer に代入、6,7 行目で入力欄に 1 文字以上入力されていたら ScrollView に myAnswer の値を表示します。最後に 8 行目で入力欄をリセットします。

```
1  private void judge(int myAnswer, int cmpAnswer) {
2         if( myAnswer == cmpAnswer){
3             bingo();
4         }else if( myAnswer < cmpAnswer && myAnswer + 25 >= cmpAnswer ){
5             small();
6             input();
7             nTimes();
8         }else if( myAnswer < cmpAnswer && myAnswer + 25 < cmpAnswer ){
9             tooSmall();
10            input();
11            nTimes();
12        }else if( myAnswer > cmpAnswer && myAnswer - 25 <= cmpAnswer ){
13            big();
14            input();
15            nTimes();
16        }else if( myAnswer > cmpAnswer && myAnswer - 25 > cmpAnswer ){
17            tooBig();
18            input();
19            nTimes();
20        }
21    }
```

リスト 8.4: 判定

```
1  private void bingo() {
2         TextView result = (TextView)findViewById(R.id.ScrollView);
3         result.append(getString(R.string.result1) + count + getString(R.string.result2)
4                                                                     + "\n");
5     }
```

リスト 8.5: 判定結果の一例。当たり！

judge() メソッドは以下の様な流れです。myAnswer と cmpAnswer が等しいかどうか判定し、等しかったら bingo() メソッドを呼び出します。違ったらそれぞれの条件ごとに大きいか小さいかなどを判定し、判定結果を表示後、input() で「数字を入力してください」と表示してから、現在の解答回数を更新します。

small() や big() などの中身は、上に書いた bingo() メソッドと同じような感じで記述します。ここで、いままで何度か出てきている R.string. ○○は string.xml という別の場所に記述してあります。

これまでの流れをまとめると、onCreate() → startGame() → createNum() →「数字を入力してください」→入力欄に数字を入力し、ボタンをクリック→ onClick() → editText() → judge() で判定→「数字を (ry」...（以下ループ）となります。

8.3 終わりに

今回は初めてということでなんの面白みもないゲームになってしまいましたが、今回実際に作ってみることによってほんの少しですが Android プログラミングが理解出来たと思います。来年までにはもっと勉強し、シューティングなどのもう少し面白いゲームを作れるようになりたいです。

8.4 おまけに

おまけとして、Android アプリ開発環境の構築とアプリのテスト方法について書きます。

8.4.1 開発環境構築

今回は Windows ユーザー向けに書きます。必要なソフトウェアは、JDK(Java Development Kit), AndroidSDK, Eclipse の 3 つです。

JDK のインストール

1）http://www.oracle.com/technetwork/java/javase/downloads/index.html にアクセスし、[DOWNLOAD] アイコンをクリックします。

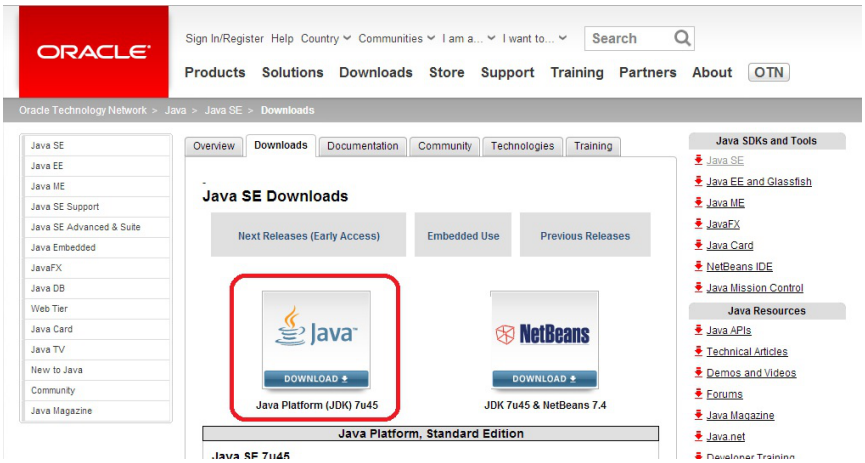
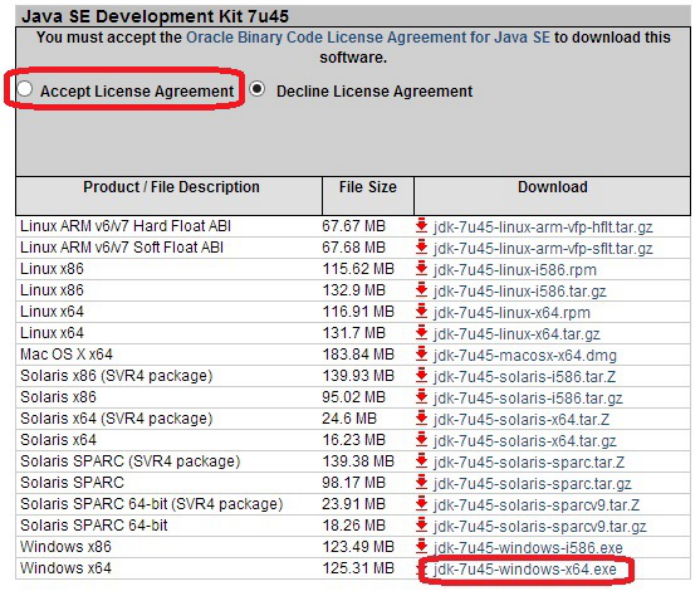


図 8.1: ダウンロード 1

2）[Accept License Agreement] を選択し、自分の PC に合わせたファイルをクリックしてダウンロードを開始します。今回僕は Windows の 64bit ですので [jdk-7u45-windows-x64.exe] を選択しています。自分の PC が何 bit なのかは、[コントロールパネル] → [システムとセキュリティ] → [システム] で出てきた画面の [システムの種類] を見ればわかります。32bit の場合は [Windows x86] の [jdk-7u45-windows-i586.exe] を選択してください。



Java SE Development Kit 7u45

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Accept License Agreement Decline License Agreement

Product / File Description	File Size	Download
Linux ARM v6/v7 Hard Float ABI	67.67 MB	jdk-7u45-linux-arm-vfp-hflt.tar.gz
Linux ARM v6/v7 Soft Float ABI	67.68 MB	jdk-7u45-linux-arm-vfp-sflt.tar.gz
Linux x86	115.62 MB	jdk-7u45-linux-i586.rpm
Linux x86	132.9 MB	jdk-7u45-linux-i586.tar.gz
Linux x64	116.91 MB	jdk-7u45-linux-x64.rpm
Linux x64	131.7 MB	jdk-7u45-linux-x64.tar.gz
Mac OS X x64	183.84 MB	jdk-7u45-macosx-x64.dmg
Solaris x86 (SVR4 package)	139.93 MB	jdk-7u45-solaris-i586.tar.Z
Solaris x86	95.02 MB	jdk-7u45-solaris-i586.tar.gz
Solaris x64 (SVR4 package)	24.6 MB	jdk-7u45-solaris-x64.tar.Z
Solaris x64	16.23 MB	jdk-7u45-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	139.38 MB	jdk-7u45-solaris-sparc.tar.Z
Solaris SPARC	98.17 MB	jdk-7u45-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.91 MB	jdk-7u45-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	18.26 MB	jdk-7u45-solaris-sparcv9.tar.gz
Windows x86	123.49 MB	jdk-7u45-windows-i586.exe
Windows x64	125.31 MB	jdk-7u45-windows-x64.exe

図 8.2: ダウンロード 2

3）ダウンロードしたインストーラの指示に従ってインストールします。設定はデフォルトで問題ありません。

4）インストールが終わったら環境変数の設定をします。先ほど bit を確認した [システム] を開きます。画面左にある [システムの詳細設定] をクリックし、[環境変数] をクリックします。[システム環境変数] の中に Path と書かれたものがあると思うので、それを選択し [編集] ボタンをクリックします。次に [変数値] のところに値を入力するのですが、元から入力されている値は”消さないように”注意してください。肝心の入力する値は、インストールした JDK の bin フォルダを開き、図 5 の四角で囲んだ [bin] を右クリックしてアドレスのコピーをします。そして [元の値; コピーした値;] となるようにペーストします。

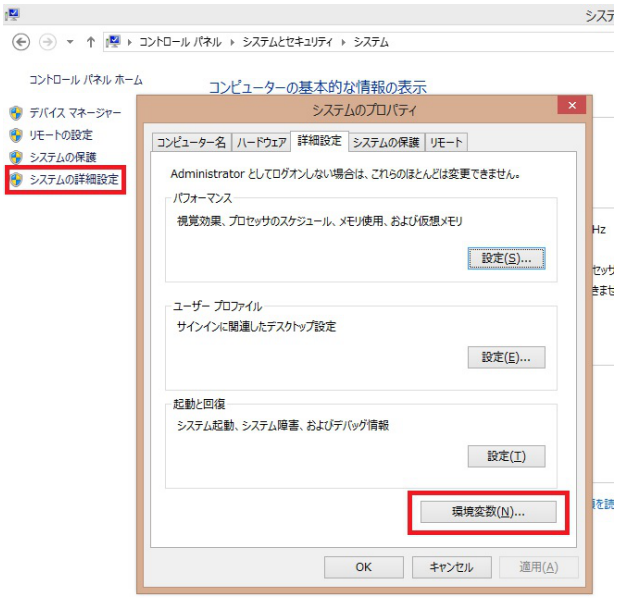


図 8.3: 環境変数

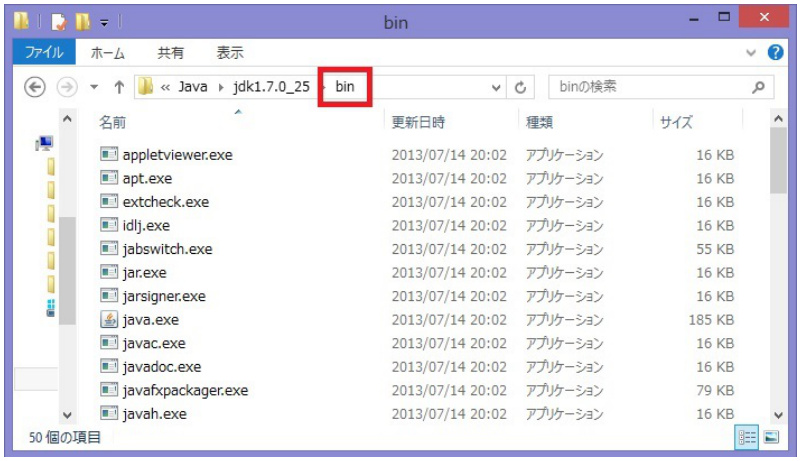


図 8.4: アドレスをコピー

5）同様に今度は [システム環境変数] の [JAVA_HOME] を設定します（もし項目が無ければ作ってください）。元から入力されている値は消さないように注意しながら先ほどコピーしたアドレスを同じようにペーストしてください。

AndroidSDK のインストール

1）http://developer.android.com/intl/ja/sdk/index.html にアクセスし、[Download the SDK] をクリックします。規約に同意し、bit を選択しダウンロードを開始してください。

2）ダウンロードしたファイルを適当な場所で解凍して完了です。

Eclipse の日本語化

8.4.1 でダウンロードした中に Eclipse が入っています。Eclipse とは統合開発環境の一つで、色々と便利な開発ツールです。今回これを日本語化するために Pleiades プラグインを導入します。

1）http://mergedoc.sourceforge.jp/にアクセスし、Pleiades プラグインの最新版をダウンロード、解凍します（All in One の方ではない）。

2）解凍してできた [features] と [plugins] のフォルダを、Android SDK を展開したフォルダーのすぐ下にある eclipse フォルダーの中に移動します。eclipse フォルダーにはすでにこの 2 つのフォルダが存在するので、確認メッセージが表示されますが、置き換えを選択します。

3）eclipse フォルダーの中にある [eclipse.ini] ファイルを編集します。eclipse.ini の末尾に [-javaagent:plugins/jp.sourceforge.mergedoc.pleiades/pleiades.jar=default.splash] と入力してください。以上で日本語化は完了です。

8.4.2 アプリのテスト

作成したアプリの動作確認テストをする方法は、エミュレータと実機の 2 つがあります。エミュレータは Eclipse に入っています。

エミュレータでのテスト

Eclipse を起動し、[ウインドウ (W)] メニューから [Android 仮想デバイス・マネージャー] をクリックし起動します。新規の仮想デバイスを作成するために [新規...] ボタンをクリックします。出てきたダイアログの各項目に、自分の仮想する値を入れていきます。入力し終わったら [OK] ボタンを押して作成完了です。

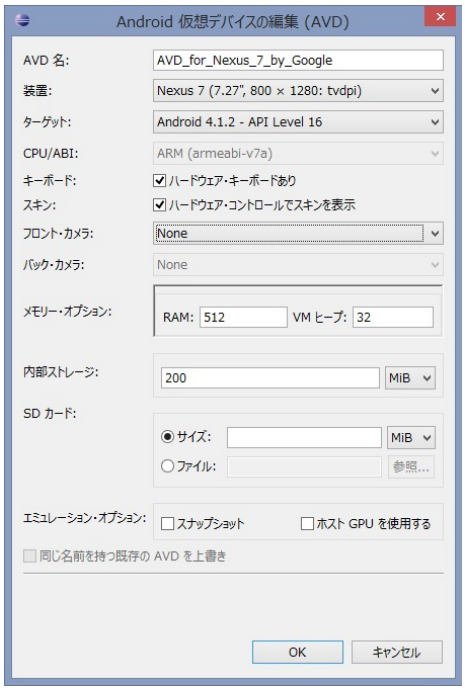


図 8.5: 仮想デバイス例

作成したエミュレータを使用するには、[実行 (R)] → [実行構成 (N)] から画面中央上部の [ターゲット] タブをクリックし、先ほど作成したエミュレータを選択します。そして実行するとエミュレータにプログラムが送られます。

実機でのテスト

エミュレータは重いため、細かい確認は基本的に実機で行います。また実機テストは、加速度センサーなどを使ったテストも行え、実際に端末に表示した時のレイアウトなどが見られることも大きな利点です。ただし、実機でテストを行うには少しだけ準備が必要です。

1）まず Android 端末側での準備です。[設定] → [開発者向けオプション] と開き、USB デバッグをオンにします。次に [設定] → [セキュリティ] と開き、提供元不明のアプリのインストールを許可してください。※ Android の端末やバージョンによって多少異なるかもしれません。

2）次は PC 側の設定です。Android 端末の USB ドライバをインストールする必要があります。これは各メーカーによって異なるため「(機種名)　USB ドライバ」などで検索してみてください。

3）実機と PC を USB ケーブルで接続します。

4）エミュレータと同様に、Eclipse から [実行 (R)] → [実行構成 (N)] と開き [ターゲット] タブから [デバイスを選択するときに常にプロンプトを表示] にします。この状態でプログラムを実行するとターゲットを選択する画面が出てくるので、Android 端末を選択し [OK] を押します。以上です。

参考文献

[1] スッキリわかる Java 入門　中山清喬/国本大悟（著）

[2] 作ればわかる! Android プログラミング 第 2 版　金宏和實（著）

[3] ”改めて、Android の開発環境をセットアップしたのでメモ。 - Qiita [キータ]”.(http://qiita.com/yu_naka0607/items/7877e491c6adca145d3b)

9 C言語で画像処理

情報工学課程 2 回生 渡邉 雄也

9.1 はじめに

本記事では、処理をする前の画像は 256 色ビットマップ形式の画像を用いている。そして、閾値 (しきいち) を使って 2 値の画像、すなわちモノクロに変換する。多くの情報を捨てることになるが、単純で明快である。このように画像を 2 値画像にする方法のことを 2 値化処理と呼ぶ。この 2 値化したものを利用すれば、画像に描かれている人物や物体の輪郭を抜き出すことなどに役立つ。この 2 値化というのは、画像処理を理解するうえで基本となる重要な概念である。本記事では、2 値化したものを利用するのではなく、それより前の段階である画像の 2 値化のみを扱う。

9.2 画像データの格納法

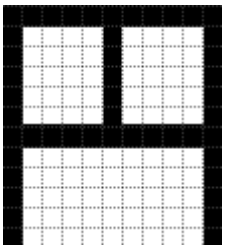


図 9.1: 円の画像

上記のような画像で画像データの格納法を考えてみる (グリッド線は便宜上、つけている)。ディジタル画像では左上隅の画素を (0,0) 番目の画素と考えている。ここから始めて横方向に i 画素、縦方向に j 画素進むと (i,j) 番目の画素の値を得られる。これを二次元配列に格納していけば良い。ここでは、image_in[j][i] とする。

9.3 閾値処理

閾値処理というのは入力画像の各画素についての明るさがある値 (ここを閾値と呼ぶ) 未満の場合は 0, それ以外の場合は 1 とするものである。ただし、実際には 0 の場合は LOW = 0,1 の場合は HIGH = 255 となっている (要するにモノクロ)。数式で書くと以下のようになる。

$$g(x,y) = \begin{cases} 1 & f(x,y) \geq t \\ 0 & f(x,y) < t \end{cases}$$

f(x,y) は処理前の、g(x,y) は処理後の濃度値を表す。t は閾値である。

9.4 閾値処理を実際にやってみた

図 9.2: 処理前の画像 1

上記の画像（kitcc とは Kyoto Institute of Technology Computer Club のことで京都工芸繊維大学にあるコンピュータ部のことである) を閾値の値を変化させながら処理していく。

図 9.3: 閾値 = 220

閾値が 220 では kitcc は闇にうもれる。閾値を減らし、このまま調べていく。

図 9.4: 閾値 = 200

図 9.5: 閾値 = 150

以上のように閾値によって画像は変化していき、閾値が 150 の際には綺麗に kitcc の文字がくり抜かれている。余談ではあるが、閾値を 0 にすると真っ白な画像が生成され、256 にすると真っ黒な画像が生成される。このことは上記にある数式に代入すれば理解できる。

9.5 閾値処理のソースコード

それでは、閾値処理のソースコードを解説する。先にコードを記述しており、後に説明が書かれている。

閾値処理の関数

```
/*-------------------- 閾値処理 --------------------------
image_in: 入力画像配列
image_out: 出力画像配列
thresh: 閾値
------------------------------------------------------------*/
void threshold(unsigned char image_in[Y_SIZE][X_SIZE],
    unsigned char image_out[Y_SIZE][X_SIZE], int thresh)
{
      int i, j;

      for (i = 0; i < Y_SIZE; i++) {
            for (j = 0; j < X_SIZE; j++) {
                  if ((int)image_in[i][j] >= thresh)
                        image_out[i][j] = HIGH;
                  else
                        image_out[i][j] =  LOW;
            }
      }
}
```

この時、X_SIZE や Y_SIZE は画像の大きさである (単位はピクセル)。このソースコードの意味は上記にある数式である。つまり、入力画像配列の情報を判定し、出力画像配列に HIGH , LOW を格納する。そうして 2 値の画像に変換される。なお、if 文の条件を逆 (不等号の向きを変える) にすれば以下のような画像となる。この際も閾値は 150 に設定している。

図 9.6: 図 9.5 の反転画像

9.6 閾値を決定する

ここまで閾値は私が手動で決めている。これは非常に面倒くさい。もっと簡単にスマートに自動で決定する方法を使う。それはヒストグラム (頻度分布) を使う方法である。要するに濃度値 i の画素が何個あるか調べることである。そうすれば、背景と文字の部分で濃度値が変化しているのでヒストグラムにも特徴が出てくる。

9.7 ヒストグラムの概要

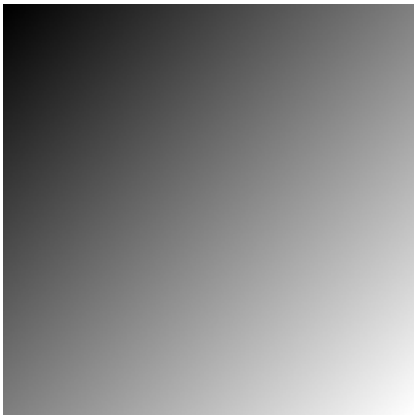


図 9.7: ヒストグラム実験の元画像

ここでヒストグラムがどのように表されるか理解しやすいようにグラデーションの画像を用意した。この画像でヒストグラムを作成してみる。この画像ならば最も左上の画素と最も右下の画素が少なくなる。反対に右上から左下にかけての画素 (右上と左下の濃度値は等しい) が多くなる。推察はこのぐらいにして実際にヒストグラムを見てみる。

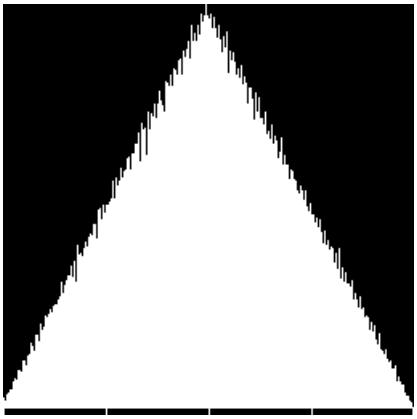


図 9.8: ヒストグラム画像

結果を見てみると図 9.8 のように山なりのヒストグラムとなった。なお、濃度値は左端が 0, 右端が 255, となっている。おおまか、予想通りの結果となっている。

9.8 ヒストグラムのソースコード

ヒストグラムを求めるソースコードを解説する。先ほどと同様に、後に説明が書かれている。

ヒストグラムを求める関数

```
/*---------ヒストグラムを求める処理----------------------
image_in: 入力画像配列
hist: ヒストグラム
-------------------------------------------------------*/
void histgram(unsigned char image_in[Y_SIZE][X_SIZE],
              long hist[256])
{
      int i, j, n;

      for (n = 0; n < 256; n++) hist[n] = 0;
      for (i = 0; i < Y_SIZE; i++) {
            for ( j = 0; j < X_SIZE; j++) {
               n = image_in[i][j];
               hist[n]++;
            }
      }
}
```

大体分かると思うが、hist[] という配列は濃度値 n を添字として受け取る。そして、hist[n] をインクリメントすることで濃度値が何画素あるか分かる。

9.9 平滑化

図 9.8 を見て分かるようにガッタガッタなものである。このままでは、境界が分かりにくい (この画像では境界なんてない)。よって平滑化してみる。つまり、ヒストグラム上で近い濃度値同士を平均化する。

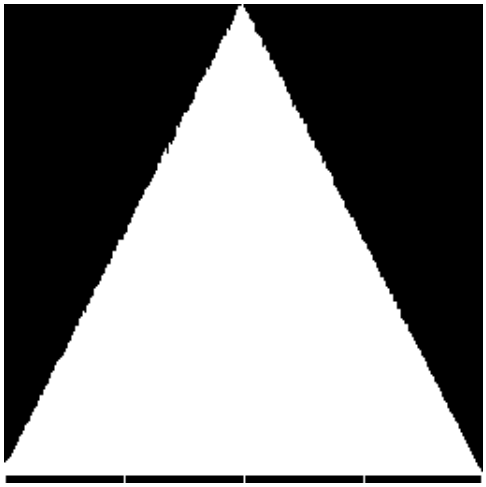


図 9.9: ヒストグラム平滑化画像

当然であるが白い部分をみると山なりを描いている。今回は濃度値 i-2,i-1,i,i+1,i+2 の平均として計算している。

9.10 平滑化のソースコード

平滑化するソースコードを解説する。先ほどと同様に、後に説明が書かれている。

ヒストグラムを平滑化する関数

```
/*--------------ヒストグラムを平滑化する-------------------
hist_in: ヒストグラム 平滑化前
hist_out: ヒストグラム 平滑化後
-----------------------------------------------------------*/
void histsmooth(long hist_in[256], long hist_out[256])
{
      int  m, n, i;
      long sum;

      for (n = 0; n < 256; n++) {
         sum = 0;
         for (m = -2; m <= 2; m++) {
            i = n + m;
            if (i <   0) i =   0;
            if (i > 255) i = 255;
            sum += hist_in[i];
         }
         hist_out[n] = (long)((double)sum / 5.0);
      }
}
```

先ほど、濃度値 i-2,i-1,i,i+1,i+2 の平均化としていたが、この考えならば i の値によって濃度値が限界を超える場合が生じる (0 未満や 256 以上の場合)。よって、if 文が追加されている。なお、この平滑化は 1 回だけしか使わないというわけではなく、デコボコが消えるまで使用する必要がある。

9.11 モード法

先ほど、ヒストグラムに特徴が出ると記述したが、これは文字の部分と背景の部分で谷が生じるということである。このようにヒストグラムの谷を使って閾値を決める方法をモード法と呼ぶ。なお、このモード法というのは山が 2 個でその間に谷があることを前提とした方法である。そのため、平滑化しなければ谷が何個もあるとうまくいかない。その平滑化も 3、4 回しなければデコボコが消えない場合もある。

9.12 モード法のソースコード

モード法のソースコードを解説する。先ほどと同様に、後に説明が書かれている。

モード法で閾値を決定する関数

```
/*---------モード法で閾値を決定する-----------------------
hist: ヒストグラム
-------------------------------------------------------*/
int threshmode(long hist[256])
{
      int m, n;
      long max, min;

      max = 0;
      for (m = 0; m < 256; m++) {
            if (max <= hist[m]) max = hist[m];
            else break;
      }
      min = max;
      for (n = m; n < 256; n++) {
            if (min >= hist[n]) min = hist[n];
            else break;
      }
      return n - 1;
}
```

初めの、for ループで max という変数はヒストグラム上で表される最初の山の最も多い濃度値の個数を獲得する。次の for ループで、min という変数はヒストグラム上の最初の山と次の山の谷の部分の濃度値の個数を獲得する。そして、その濃度値を戻り値として扱う。この戻り値こそが、閾値である。

9.13 モード法による画像処理

図 9.10: モード法施行前

ラストとしてモード法で画像処理をしてみる。

図 9.11: モード法施行後

綺麗に処理できる結果となった。

9.14 終わりに

画像がやたらと多くなりましたが、画像処理だもの。しようがない。まだまだ、楽しいこと、面白いこと、興味深いことがあるのでこのまま画像処理について勉強していく所存である。以上、ここまで読んでくださった皆様ありがとうございます。

参考文献

[1] 井上誠喜ほか,C 言語で学ぶ実践画像処理, オーム社,2010,391p

10 Kinectでできること

情報工学課程 3 回生 中川 鴻佑

10.1 Kinectとは

Kinect とは 2010 年 11 月に Microsoft 社から発売された Xbox 360 用のコントローラです。ゲームのコントローラといっても手に持って主に両手の親指だけで操作するものではなく、カメラや距離センサ、マイクを搭載しており、身振りや声で操作するものです。「何も持たずにゲームができる」、「体全体を使って操作できる」という点で家庭用ゲーム機のコントローラとして画期的なものです。

図 10.1: Kinect

さらにこの Kinect には Xbox との互換性のため USB コネクタが付属されたためコンピュータと接続でき、また Kinect と Xbox 360 の間で転送されるデータは暗号化されていないため、Kinect を PC で用いようとする試み、すなわち Kinect ハック[1]も盛んに行われてきました。その流れを受けて Microsoft 社も Windows で開発できる SDK として Kinect for Windows SDK を発表し、Kinect の全ての機能が利用できるようになりました。さらに Windows 用の Kinect(Kinect for Windows) も発売され、この Kinect for Windows によって商用利用に対する制限もなくなりました。

この記事では Kinect の概要や、Kinect ハックの流れ、主な用途と共に、Kinect を用いたアプリケーションで筆者がすごいと思ったものを紹介します。

10.2 Kinectの概要

Kinect にはカメラと赤外線による距離センサ、内蔵マイクが搭載されています (図 10.2)。距離センサは Kinect の前方の空間を立体的に認識することができ、また 4 つのマイクを並べたマイクアレイは音源の方向を感知することができます。これらの機能を用いて、プレイヤーの位置や動き、さらには骨格を認識・追跡したり、雑音を除去して音声入力を受け取ったりといったことを始めとして、様々なことが実現できます。また、チルトモーターを内蔵していて、センサの上下の角度をアプリケーション側で調整することもできます。

価格は 20000 円程度ですが、従来ではさらに高価な機材を買わなければできなかった三次元計測なども可能であるため、特に距離センサは注目され様々なアイデアに応用されているように思います。

[1]ハッキングと聞くと犯罪行為だと思われる方も多いかもしれませんが、ハッキングのもともとの意味は高い技術力を駆使してシステムを操ることであり、不正アクセスなどの行為は正式にはクラッキングと呼ばれます。ハッカーという言葉も優秀な技術者に対する敬称として使われる場合もあります。

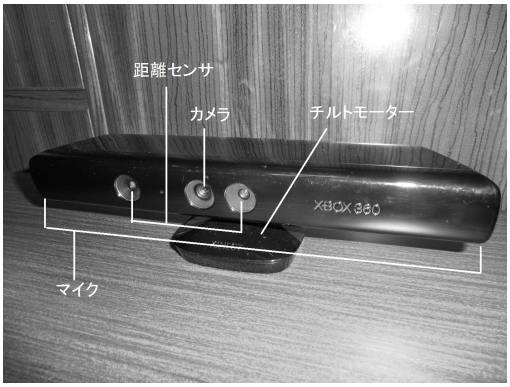


図 10.2: Kinect のセンサ・マイク

10.3 Kinect ハックの流れ

Kinect を思いのままにコントロールしようとする試みである Kinect ハックは、Kinect の発売当初にはもう始まっていました。ここでは Kinect ハックのさきがけとなった Open Kinect と、それに対する Microsoft の反応についてまとめました。

10.3.1 Open Kinect

アメリカの Adafruit Industries 社は全米で Kinect が発売された 2010 年 11 月 4 日に Kinect を Xbox 360 以外のデバイスにも応用できるようにすることを目的に Open Kinect Project を立ち上げ、ハッキングコンテスト Open Kinect Challenge を開始しました。[1] このコンテストでは、オープンソースのドライバを開発できた者に賞金 1000 ドルを授与すると発表しました。[2] この直後に Microsoft は

> Microsoft does not condone the modification of its products.
> With Kinect, Microsoft built in numerous hardware and software safeguards designed to reduce the chances of product tampering. Microsoft will continue to make advances in these types of safeguards and work closely with law enforcement and product safety groups to keep Kinect tamper-resistant.
> Microsoft は製品の改変を許可してない。Kinect にも、不正改変される可能性を減らすべく設計されたハードウェア・ソフトウェア両面の安全策をいくつも講じている。この安全策をこれからも推し進めていき、法令を遵守して Kinect を改変から守る

とアメリカのメディア CNET に声明を寄せました [3] が、これに対し Adafruit Industries 社はその日のうちに

> Ok fine, the bounty is now double, $2,000.
> よろしい、ならば賞金は倍の 2000 ドルだ。

と発表し、賞金を倍の 2000 ドルに引き上げました。[4]

そうした中 11 月 7 日、NUI Group の掲示板に Kinect からリアルタイムで RGB カメラのステータスを取得して Windows7 マシン上のウィンドウに表示している様子が映っている Youtube 動画が投稿されました。[5]11 月 8 日、GameSpot に投稿されたこれについての記事に対し Microsoft は

> Kinect for Xbox 360 has not been hacked-in any way-as the software and hardware that are part of Kinect for Xbox 360 have not been modified. What has happened is someone has created drivers that allow other devices to interface with the Kinect for Xbox 360. The creation of these drivers, and the use of Kinect for Xbox 360 with other devices, is unsupported.
> Kinect のパーツであるソフト・ハード両方とも改変されてないため、Kinect はいかなるハッキングもされてない。Kinect とほかのデバイスを繋ぐドライバが作られただけだ。このようなドライバの開発や使用はサポートされてない。

と発表しましたが、これに対し Adafruit Industries 社は

> Microsoft, the Kinect is going to get ”hacked” and ”reverse engineered” and a lot of people are going to do cool things you never imagined. Kids will use this in their FIRST robotics projects, artists will turn them in to musical instruments and art. That ’ s the way it is, the way it always was, and will be ? it ’ s a good thing.
>
> Microsoft よ、Kinect はいずれハッキングやリバースエンジニアリングが行われ、多くの人が Microsoft が思いもしないようなクールなことをするだろう。こどもたちは初めてのロボット工学プロジェクトでこれを使い、アーティストは楽器やアートとして使うだろう。これはとても良いことだ。

という声明と共に賞金を 3000 ドルに引き上げ、挑発するとともに Kinect がさまざまな環境下で使えるようになることの重要性を訴えました。[6]

ハッキングコンテストの結果は、Hector Martin 氏がオープンソースのライブラリ libfreenect を作り、賞金を受け取ることになりましたが、Open Kinect によって多くの人が Kinect を利用できるようになりました。[7]

10.3.2 Microsoft の反応

上記のように Kinect ハックに対し不快感を示していたかに見えた Microsoft ですが、11 月 19 日 Kinect の開発者の一人 Alex Kipman 氏が米のラジオ局 NPR の番組で

> The first thing to talk about is Kinect was not actually hacked. Hacking would mean that someone got to our algorithms that sit on the side of the Xbox and was able to actually use them, which hasn’t happened. Or it means that you put a device between the sensor and the Xbox for means of cheating, which also has not happened. That’s what we call hacking, and that’s why we have put a ton of work and effort to make sure it doesn’t actually occur.
>
> まず第一に、Kinect はハッキングされていません。このハッキングという言葉が意味するのは、Xbox 内部のアルゴリズムにアクセスすること、もしくはチート行為の為に Kinect と Xbox の間に別のデバイスを接続することであり、このようなことは起こっていません。またこれらの防止に努めていきます。
>
> What has happened is someone wrote an open-source driver for PCs that essentially opens the USB connection, which we didn’t protect by design, and reads the inputs from the sensor. The sensor again, as I talked earlier, has eyes and ears and that’s a whole bunch of, you know, noise that someone needs to take and turn into signal.
>
> PC で USB 接続を開いてセンサーからの入力を読み取るオープンソースのドライバを作った人がいましたが、USB 接続に対して設計上の保護はしていません。

と発言し、独自のプログラムを作ったからといってトラブルになることはないと明言しました。[8]Microsoft の懸念は別のところにあったようです。

そうして 2011 年 4 月 13 日 Microsoft は自社のイベント MIX11 で SDK のリリースを発表し、同年 6 月 17 日にはベータ版である Kinect for Windows SDK from Microsoft Research をリリースしました。

10.4 Kinect の用途

前章のように Kinect には距離センサとマイクが搭載されており、それを様々な用途に利用すべく開発が進んでいます。ここではその主な用途を挙げていきたいと思います。

10.4.1 ユーザインタフェース

Kinect はもともとはゲーム機のコントローラとして登場したものであることから、ユーザとコンピュータの接点であるユーザインタフェースとして使用しているアプリケーションも多く存在します。例えば右手をマウスの代わりにして PC を操作するアプリケーションなどです。また、画面にプレイヤーの姿を映してジェスチャーで操作する自作ゲームなども SDK を使えば簡単に作れますから、Kinect の用途としてはこれが一番の主流かもしれません。

10.4.2 センサー

Kinect の距離センサとマイクアレイを生かして、センサーとして実装している例もあります。その一例として、デジタルサイネージ (デジタル技術を用いた広告) と連携し、通行人に反応してその人に対して商品の宣伝をするシステムが挙げられます。また変わった例として、警備ロボットに Kinect を搭載して不審者を検知したら通報するというものもありました。[10]

10.4.3 三次元計測

さらに距離センサの機能を生かして三次元計測に重点を置いた例もあります。例としてはプロジェクションマッピングの実装があります。距離センサで物体の形や大きさを認識し、投影する映像に反映するというものです。プロジェクタが別で必要であり規模が大きくなると Kinect で対応できませんが、小規模なものならば三次元のデータの計測は Kinect で行うことができます。[10]

10.5 高度な Kinect の応用例

では、主な用途を紹介したところで筆者がすごいと感じた Kinect の応用例を紹介します。

10.5.1 ホログラム映像視聴

まず、Kinect を用いてホログラム映像を見せている例を紹介します。[13] 仕組みとしては映像の視聴者を Kinect で認識し、その動きに合わせて映像を動かしてあたかもホログラムに見せるというものですが、その発想には驚きました。

10.5.2 非接触呼吸心拍測定

次に紹介するのは Kinect を用いたアプリケーション開発のコンテストである Kinect for Windows Contest 2013 で技術賞を受賞した作品です。[9] それは Kinect の前に座るだけで呼吸数と心拍数を計ってくれるというもので、驚くべきことに服の上からでも、(体と密着させていれば) タオルケットや布団の上からでも計測できます。呼吸・心拍数と胸部の動きの関係について詳しくは割愛しますが、センサの前に居る人の顔と胸部を追跡し、距離の変化から計測するそうです。[14] 心拍計測は 2014 年登場予定の次世代 Kinect に搭載される予定のものですが、この作品は現行版の Kinect を用いてそれを実装している点でも驚くべきものです。

10.5.3 三次元フェイスライン測定システム

最後に、これも同じく Kinect for Windows Contest 2013 でグランプリを獲得した作品を紹介します。これは美容ビジネスにおいて、施術前後の顔の形状を Kinect を用いて計測し、その効果を数値とビジュアルで示すシステムです。Kinect の三次元計測を存分に生かしたもので、Kinect の可能性を感じました。(図**??**)

10.5.4 おわりに

ここまで目を通して頂いたならば、Kinect が一時期だけ流行っただのゲーム機用コントローラだという印象は変わったのではないでしょうか。私自身も Kinect を使った簡単なゲーム開発をやってみて、そのついでで Kinect について調べてみると、たくさんのこういった面白い使い方が見つかって驚きました。ここで紹介したような高度なものを作ろうとすると並大抵でない興味と努力と技術が必要となりますが、Kinect for Windows でなくても Xbox 360 があれば、無料の SDK を使って簡単なアプリケーションは十分に作れます。この記事を読んで少しでも Kinect に興味を持ち、かつ家に Kinect があり、かつプログラミングの経験が少しでもあるのなら、一度 SDK を用いて何か作ってみてください。[16]

参考文献

[1] 賞金付きオープンソース ”Kinect ”ドライバ開発コンテスト！
http://ascii.jp/elem/000/000/568/568632/

[2] The Open Kinect project ? THE OK PRIZE ? get $3,000 bounty for Kinect for Xbox 360 open source drivers
http://www.adafruit.com/blog/2010/11/04/

[3] Bounty offered for open-source Kinect driver
http://news.cnet.com/8301-13772_3-20021836-52.html

[4] “ Microsoft isn ’ t taking kindly to the bounty offer ”
http://www.adafruit.com/blog/2010/11/04/

[5] Windows Kinect Driver/SDK - Xbox NUI Audio, NUI Camera, NUI Motor and Accelerometer
http://nuigroup.com/forums/viewthread/11154/

[6] The bounty is now $3k ? “ Software giant says engineer ’ s linking of camera-based system to Windows 7 PC does not constitute hacking ”
http://www.adafruit.com/blog/2010/11/08/

[7] OpenKinect Main Page
http://openkinect.org/wiki/Main_Page

[8] How The X-Box Kinect Tracks Your Moves
http://www.npr.org/2010/11/19/131447076/how-the-x-box-kinect-tracks-your-moves

[9] Kinect for Windows Contest 2013 リポート（1）
http://monoist.atmarkit.co.jp/mn/articles/1309/24/news052.html

[10] Kinect for Windows Contest 2013 リポート（2）
http://monoist.atmarkit.co.jp/mn/articles/1309/30/news056.html

[11] Microsoft　Kinect 公式ページ
http://www.xbox.com/ja-JP/kinect

[12] Kinect が切り開く“ 夢の近未来 ”中村　薫
http://www.atmarkit.co.jp/fdotnet/special/kinectfuture_01/kinectfuture_01_01.html

[13] ホログラム映像視聴
http://www.youtube.com/watch?v=9xMSGmjOZIg&feature=related

[14] 非接触バイタル・センシング
http://www.youtube.com/watch?v=xQrN5d1rYFE

[15] 『KINECT for Windows SDK プログラミング C++編』秀和システム
中村　薫／斉藤　俊太／宮城　秀人　著

[16] Kinect for Windows SDK 公式 Web ページ:
http://www.microsoft.com/en-us/kinectforwindows/

編集後記

Lime48 号はお楽しみいただけましたでしょうか。編集担当の渡邉です。今回の Lime は、1 回生の記事が大半を占めていました。2 回生以上の記事を割と楽しみにしていた者として少々複雑な気分です。

さて、内容としては今年の Lime はゲームの記事が多く占めていました。また、ゲームを製作したといっても、同じ趣旨のゲームを違う言語で製作するという面白い試みもあり、一味違った記事もありました。ゲームが好きな自分としては、多種多様なゲームの記事を読むことができて楽しかったです。

とりあえず、大勢の助けを借りて無事に発行できたことを喜んでいる現状です。それでは、次号の Lime を是非ご期待ください。

平成 25 年 11 月 17 日
編集担当 渡邉 雄也

Lime Vol.48

平成 25 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/