平成 28 年 11 月 24 日
京都工芸繊維大学コンピュータ部

Lime 54

はじめに

コンピュータ部部長の尾崎です。このたびはお忙しい中、本学の学園祭にお越しいただき、我々コンピュータ部にご興味、ご関心を持っていただきありがとうございます。この度は Lime54 号を手に取ってくださったこと、部を代表いたしまして、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部員個人の趣味全開なものまで内容は多岐にわたります。これを通じて、コンピュータ部のことを知っていただき、少しでも、我々の活動に興味を持っていただければ幸いです。そして、今年の Lime 作成にあたり、編集をしてくれた矢口君、表紙の作成にあたった森下君、忙しい中記事を書き上げた部員各位、並びに、OB・ OG の方々を含め、日々我々の活動を支えてくださっているすべての方への感謝をもって始めの挨拶とさせていただきます。

平成 28 年 11 月吉日
京都工芸繊維大学コンピュータ部部長　尾崎　俊宏

目 次

1 ボールの物理演算

情報工学課程 1 回生 森下和馬

1.1 はじめに

私は今まで何度かプログラミングをしてきましたが、ウィンドウなどを使った GUI のプログラミングの経験はほとんどありませんでした。今回はその GUI プログラミングの経験を積むことも兼ねて、ボールの物理演算を行いその動きを表示するプログラムを javaapplet を使って作りました。拙いプログラムかもしれませんが、是非最後まで見ていってください。

1.2 仕様

今回のプログラムではボールをウィンドウのクリックした地点から任意の角度・速度で発射します。その後、ボールは重力の影響を受けつつ運動していきます。ウィンドウの端にぶつかったときには反発係数に従って跳ね返ります。また速度などのそれぞれの値はウィンドウ上で設定できるようします。またボールは 3 つまで同時に動かすことができるようにします。

1.3 実装

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.util.Timer;
import java.lang.Math;


/*
<applet code="Ball" width=800 height=600>
</applet>
*/
```

```
public class Ball extends Applet implements MouseListener,ActionListener
{
    int angle;
    int width;
    int height;
    int i=0;
    int a = 0;
    double x[] = {0,0,0};
    double y[] = {0,0,0};
    double G;
    double E;
    double speed = 0;
    double speedX[] = {0,0,0};//X 成分の速度
    double speedY[] = {0,0,0};//Y 成分の速度
    String str1;//文字列の一時保存領域その 1
    String str2;//文字列の一時保存領域その 2
    Label la1 = new Label("速度");
    Label la2 = new Label("角度");
    Label la3 = new Label("重力");
    Label la4 = new Label("反発");
    TextField tf1 = new TextField("40");
    TextField tf2 = new TextField("60");
    TextField tf3 = new TextField("10");
    TextField tf4 = new TextField("0.9");
    Button button1 = new Button("適用");
    Button button2 = new Button("開始");

    public void init()
    {
        addMouseListener(this);
        Dimension size = getSize();
        width = size.width;
        height = size.height;
        setLayout(null);
        add(la1);
        add(la2);
        add(la3);
        add(la4);
        add(tf1);
        add(tf2);
        add(tf3);
        add(tf4);
        add(button1);
        add(button2);
        button1.addActionListener(this);
        button2.addActionListener(this);
        la1.setBounds(10,height-40,40,40);
        tf1.setBounds(50,height-40,40,20);
        la2.setBounds(120,height-40,40,40);
        tf2.setBounds(160,height-40,40,20);
        la3.setBounds(230,height-40,40,40);
        tf3.setBounds(270,height-40,40,20);
        la4.setBounds(340,height-40,40,40);
        tf4.setBounds(380,height-40,40,20);
```

```
        button1.setBounds(440,height-40,40,20);
        button2.setBounds(500,height-40,40,20);
        str1 = tf1.getText();
        speed = Double.parseDouble(str1);
        str1 = tf2.getText();
        angle = Integer.parseInt(str1);
        str1 = tf3.getText();
        G = Integer.parseInt(str1);
        G = G/10;
        str1 = tf4.getText();
        E = Double.parseDouble(str1);
    }
```

ここでは必要なクラスをインポートしたり、各種インターフェースを実装したりしています。また init メソッドではボタンやテキストボックスを配置や、ボタンとインターフェースを関連付けなどをしています。

```
    public void paint(Graphics g)
    {

        Dimension size = getSize();
        width = size.width-10;
        height = size.height-10;
        if(i<3) a=i;
        if(i>=3) a=3;
        for(int j=0;j<a;j++)
        {
            //ウィンドウの端にぶつかった時の跳ね返り処理
            if(x[j]<=10){
                speedX[j] = (-1)*speedX[j]*E;
                x[j]*=-1;
            }
            if(x[j]>=width){
                speedX[j] = (-1)*speedX[j]*E;
                x[j]=width-(x[j]-width);
            }
            if(y[j]<=10){
                speedY[j] = (-1)*speedY[j]*E;
                y[j]*=-1;
            }
            if(y[j]>=height){
                speedY[j] = (-1)*speedY[j]*E;
                y[j]=height-(y[j]-height);
            }

            g.fillOval((int)x[j],(int)y[j],20,20);
            x[j] = x[j]+speedX[j];
            y[j] = y[j]-speedY[j];
            speedY[j] -= G;
        }
    }
```

paint メソッドでは画面端の位置を確認して、ボールが画面端にぶつかるなら跳ね返り位置を求め運動の向きを反転させます。今回のプログラムではボールの大きさは直径 20px で

固定したので画面端からその半分の 10px 離れた地点を壁だと判定します。

```
public void mouseClicked(MouseEvent me)
    {
        x[i\%3] = me.getX();
        y[i\%3] = me.getY();
        speedX[i\%3] = Math.cos(Math.toRadians(angle))*speed;
        speedY[i\%3] = Math.sin(Math.toRadians(angle))*speed;
        repaint();
        i++;
    }

    public void mouseEntered(MouseEvent me)
    {
        //何もしない。
    }

    public void mouseExited(MouseEvent me)
    {
        //何もしない。
    }


    public void mousePressed(MouseEvent me)
    {
        //何もしない。
    }

    public void mouseReleased(MouseEvent me)
    {
        //何もしない。
    }
```

mouseClicked メソッドはマウスイベントが発生した時に呼び出されます。マウスイベントの中でもクリックが起こったときには各ボールに位置とスピードを設定し、それを paint メソッドを使い表示させます。そのほかのイベントでは何の処理もしません。

```
    public void actionPerformed(ActionEvent e)
    {
        str2 = e.getActionCommand();
        if(str2.compareTo("適用")==0)
        {
            str1 = tf1.getText();
            speed = Double.parseDouble(str1);
            str1 = tf2.getText();
            angle = Integer.parseInt(str1);
            str1 = tf3.getText();
            G = Integer.parseInt(str1);
            G = G/10;
            str1 = tf4.getText();
            E = Double.parseDouble(str1);
        }
```

```
        if(str2.compareTo("開始")==0)
        {
            repaint();
            Timer t = new Timer();
            t.schedule(new TimerRunTask(),0,100);
        }
    }

    class TimerRunTask extends TimerTask
    {
        public void run()
        {
            repaint();
        }
    }
}
```

actionPerformed メソッドはボタンを押したときに呼び出されます。適用ボタンが押された時は、テキストボックスに入力されているそれぞれの値を次にクリックした時に発射されるボールに設定します。開始ボタンはタイマーを設定し、100 ミリ秒経過するごとに paint メソッドを呼び出して、ボールを動かしていきます。　最後に実際に出来上がった物のスクリーンショットを図 1.1 に示します。

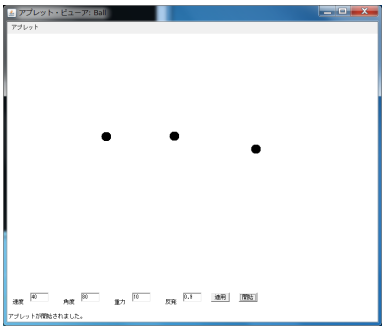


図 1.1: スクリーンショット

1.4 工夫箇所や補足

- 重力加速度は 100 ミリ秒おきに計算をするので実際に入力された値を 10 で割っています。
- 新しくボールを追加するとき配列の添字を i%3 にすることで 3 つ目以降は古い順から消えていくようになっています。
- ボールの表示位置を計算するときは出力の直前まで double 型で計算することで精度をあげています。

1.5 最後に

今回はほぼ初めての GUI のプログラミングで、いろんな本やサイトからコードを持ってきて組み合わせた結果、キメラのようなソースコードになってしまいました。初期段階での構想では、天体の動きを再現できるようにして新しい星を加えたりして遊ぶものを作りたかったのですが、勉強不足でそこまでのものは作ることが出来ませんでした。これをきっかけにもっと GUI や java について勉強をして、望んだ通りのものができるように頑張りたいです。最後まで読んでいただきありがとうございました。

2 五目並べ

情報工学課程 1 回生 古田　峻也

2.1 はじめに

C 言語で五目並べを作成しました。見にくいことに加え、禁じ手などは搭載されていないということがありますが、読んでいただけると幸いです。

2.2 操作方法

置きたい点が上から何番目の点で左から何番目の点であるかを順番に入力します。この操作を player1 と player2 で繰り返していきます。

2.3 ソースコード

```
#include <stdio.h>
int main(){

    int board[9][9] = {
            {0,0,0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0,0,0}
    };

    int suji1,suji2;
    //盤面の表示
    for(suji1=0;suji1<9;suji1++){
        for(suji2=0;suji2<9;suji2++){
        if(board[suji1][suji2] == 0)printf("・");
        else if(board[suji1][suji2] == 1)printf("○");
        else printf("●");
        }
        printf("\n");
    }
```

```
   while(1){

       //一人目の配置場所
       while(1){
           printf("player1 のターンです。上から何番目か整数のみで入力してください\n");
           scanf("%d",&suji1);
           printf("左から何番目か整数のみで入力してください\n");
           scanf("%d",&suji2);
           if(suji1<10 && suji2<10 && board[suji1-1][suji2-1] == 0){
               board[suji1-1][suji2-1] = 1;
               break;
           }else {printf("置けません\n");
           }
       }


       //盤面の表示
       for(suji1=0;suji1<9;suji1++){
           for(suji2=0;suji2<9;suji2++){
               if(board[suji1][suji2] == 0)printf("・");
               else if(board[suji1][suji2] == 1)printf("○");
               else printf("●");
           }
           printf("\n");
       }

       //一人目の縦の勝利判定
       for(suji1=0;suji1<5;suji1++){
           for(suji2=0;suji2<9;suji2++){
               if(board[suji1][suji2]+board[suji1+1][suji2]+board[suji1+2][suji2]
+board[suji1+3][suji2]+board[suji1+4][suji2] == 5){
                   break;
                   }
           }
           if(board[suji1][suji2]+board[suji1+1][suji2]+board[suji1+2][suji2]
+board[suji1+3][suji2]+board[suji1+4][suji2] == 5){
               break;
           }
       }
       if(board[suji1][suji2]+board[suji1+1][suji2]+board[suji1+2][suji2]
+board[suji1+3][suji2]+board[suji1+4][suji2] == 5){
           printf("player1 の勝ち\n");
           break;
       }

       //一人目の横の勝利判定
       for(suji1=0;suji1<9;suji1++){
           for(suji2=0;suji2<5;suji2++){
               if(board[suji1][suji2]+board[suji1][suji2+1]+board[suji1][suji2+2]
+board[suji1][suji2+3]+board[suji1][suji2+4] == 5){
                   break;
               }
           }
```

```
            if(board[suji1][suji2]+board[suji1][suji2+1]+board[suji1][suji2+2]
+board[suji1][suji2+3]+board[suji1][suji2+4] == 5){
                break;
            }
        }
        if(board[suji1][suji2]+board[suji1][suji2+1]+board[suji1][suji2+2]
+board[suji1][suji2+3]+board[suji1][suji2+4] == 5){
            printf("player1 の勝ち\n");
            break;
        }
        //一人目の斜めの勝利判定
        for(suji1=0;suji1<5;suji1++){
            for(suji2=0;suji2<5;suji2++){
                if(board[suji1][suji2]+board[suji1+1][suji2+1]+board[suji1+2][suji2+2]
+board[suji1+3][suji2+3]+board[suji1+4][suji2+4] == 5){
                    break;
                    }
            }
            if(board[suji1][suji2]+board[suji1+1][suji2+1]+board[suji1+2][suji2+2]
+board[suji1+3][suji2+3]+board[suji1+4][suji2+4] == 5){
                break;
            }
        }
        if(board[suji1][suji2]+board[suji1+1][suji2+1]+board[suji1+2][suji2+2]
+board[suji1+3][suji2+3]+board[suji1+4][suji2+4] == 5){
            printf("player1 の勝ち\n");
            break;
        }

        //一人目の斜めの勝利判定
        for(suji1=0;suji1<5;suji1++){
            for(suji2=0;suji2<9;suji2++){
                if(board[suji1][suji2]+board[suji1+1][suji2-1]+board[suji1+2][suji2-2]
+board[suji1+3][suji2-3]+board[suji1+4][suji2-4] == 5){
                    break;
                    }
            }
            if(board[suji1][suji2]+board[suji1+1][suji2-1]+board[suji1+2][suji2-2]
+board[suji1+3][suji2-3]+board[suji1+4][suji2-4] == 5){
                break;
            }
        }
        if(board[suji1][suji2]+board[suji1+1][suji2-1]+board[suji1+2][suji2-2]
+board[suji1+3][suji2-3]+board[suji1+4][suji2-4] == 5){
        printf("player1 の勝ち\n");
        break;
        }

        //二人目の配置場所
        while(1){
            printf("player2 のターンです。上から何番目か整数のみで入力してください\n");
            scanf("%d",&suji1);
            printf("左から何番目か整数のみで入力してください\n");
            scanf("%d",&suji2);
```

```
            if(suji1<10 && suji2<10 && board[suji1-1][suji2-1] == 0){
                board[suji1-1][suji2-1] = -1 ;
                break;
            }else {printf("置けません\n");
            }
        }

        //盤面の表示
        for(suji1=0;suji1<9;suji1++){
            for(suji2=0;suji2<9;suji2++){
                if(board[suji1][suji2] == 0)printf("・");
                else if(board[suji1][suji2] == 1)printf("○");
                else printf("●");
            }
            printf("\n");
        }

        //二人目の縦の勝利判定
        for(suji1=0;suji1<5;suji1++){
            for(suji2=0;suji2<9;suji2++){
                if(board[suji1][suji2]+board[suji1+1][suji2]+board[suji1+2][suji2]
+board[suji1+3][suji2]+board[suji1+4][suji2] == -5){
                    break;
                }
            }
            if(board[suji1][suji2]+board[suji1+1][suji2]+board[suji1+2][suji2]
+board[suji1+3][suji2]+board[suji1+4][suji2] == -5){
                break;
            }
        }
        if(board[suji1][suji2]+board[suji1+1][suji2]+board[suji1+2][suji2]
+board[suji1+3][suji2]+board[suji1+4][suji2] == -5){
            printf("player2 の勝ち\n");
            break;
        }

        //二人目の横の勝利判定
        for(suji1=0;suji1<9;suji1++){
            for(suji2=0;suji2<5;suji2++){
                if(board[suji1][suji2]+board[suji1][suji2+1]+board[suji1][suji2+2]
+board[suji1][suji2+3]+board[suji1][suji2+4] == -5){
                    break;
                }
            }
            if(board[suji1][suji2]+board[suji1][suji2+1]+board[suji1][suji2+2]
+board[suji1][suji2+3]+board[suji1][suji2+4] == -5){
            break;
            }
        }
        if(board[suji1][suji2]+board[suji1][suji2+1]+board[suji1][suji2+2]
+board[suji1][suji2+3]+board[suji1][suji2+4] == -5){
            printf("player2 の勝ち\n");
            break;
        }
```

```
       //二人目の斜めの勝利判定
       for(suji1=0;suji1<5;suji1++){
           for(suji2=0;suji2<5;suji2++){
               if(board[suji1][suji2]+board[suji1+1][suji2+1]+board[suji1+2][suji2+2]
+board[suji1+3][suji2+3]+board[suji1+4][suji2+4] == -5){
                   break;
               }
           }
           if(board[suji1][suji2]+board[suji1+1][suji2+1]+board[suji1+2][suji2+2]
+board[suji1+3][suji2+3]+board[suji1+4][suji2+4] == -5){
               break;
           }
       }
       if(board[suji1][suji2]+board[suji1+1][suji2+1]+board[suji1+2][suji2+2]
+board[suji1+3][suji2+3]+board[suji1+4][suji2+4] == -5){
           printf("player2 の勝ち\n");
           break;
       }

       //二人目の斜めの勝利判定
       for(suji1=0;suji1<5;suji1++){
           for(suji2=0;suji2<9;suji2++){
               if(board[suji1][suji2]+board[suji1+1][suji2-1]+board[suji1+2][suji2-2]
+board[suji1+3][suji2-3]+board[suji1+4][suji2-4] == -5){
                   break;
               }
           }
           if(board[suji1][suji2]+board[suji1+1][suji2-1]+board[suji1+2][suji2-2]
+board[suji1+3][suji2-3]+board[suji1+4][suji2-4] == -5){
               break;
           }
       }
       if(board[suji1][suji2]+board[suji1+1][suji2-1]+board[suji1+2][suji2-2]
+board[suji1+3][suji2-3]+board[suji1+4][suji2-4] == -5){
           printf("player2 の勝ち\n");
           break;
       }
   }

   return 0;
}
```

2.4 最後に

自分で考えて作成したのは初めてだったので、わからいことがたくさんあったり、技術的にできないこともたくさんありましたが、形にすることはできたので良かったです。これを機に、プログラミングの勉強を頑張り、より良いものを作れるようにしていきたいです。最後まで読んでくださりありがとうございました。

2.5 参考文献

Ｃ言語 入門 テトリスの作り方
http://akaguro.jp/index.shtml
初心者のためのポイント学習Ｃ言語
http://www9.plala.or.jp/sgwr-t/index.html

3 ボンバーマンのようなゲーム

電子システム工学課程 1 回生 足立 智生

3.1 はじめに

私は今回、ボンバーマンというゲームを真似た物をプログラミングで作ってみました。

なぜゲーム制作をしたかというと、この春と夏、コンピュータ部で私が主に学習させてもらったものが、C 言語を用いたプログラミング、またその延長としての C++を用いたゲームプログラミングの初歩であるからです。これらの知識を使って何を作ろうかと思い、ゲームの制作に至りました。

このような経緯があり、また私自身はプログラミング初心者ですので、今回の制作物は改善の余地が多くあり、面白みに欠けるものとなっているかもしれません。それでもご一読頂けるとありがたいです。

3.2 概要

まず、ボンバーマンとは、キャラクターを操作して一定時間で爆発する爆弾を置き、ブロックを壊して出るアイテムを取り自分を強化しつつ、敵を倒す 2D のゲームです。ボンバーマンはシリーズ作としてたくさんのタイトルが出ており、新しいものになるほどアイテムの種類が豊富であったり、様々な敵が出現したりします。また、対等な能力の NPC や友人との対戦を楽しむモードが存在することも多いです。

今回私が作ったゲームは、他人との対戦のみ遊ぶことができます（NPC は技量不足でまだ作れていません…）。3 人まで遊べます。プレイヤーは最初、画面上に自分の爆弾を同時に 1 つまでしか置けず、その爆発の届く範囲は爆弾の中心と左右上下それぞれ 1 マスずつです。また、存在するアイテムはボンバーマンの基礎アイテムである「爆発範囲 UP」「爆弾数 UP」「速度 UP」の 3 種類のみです。それぞれの効果は「爆発の範囲を 1 マスずつ広げる」「画面上に同時に置ける爆弾の数を 1 つ増やす」「キャラクターの移動速度を上げる」となっており、いずれも 8 個分まで効果を累積できます。各キャラクターは爆弾の爆発に当たってしまうとやられてしまいますが、1 回までなら復活でき、復活直後は無敵になります。しかし、2 回やられてしまった場合は消えてしまい、後は戦いを観戦するのみとなってしまいます。また、生き残りプレイヤーが 1 人になると勝利者が表示され、ゲーム終了となります。リセットもできます。

図 1 にゲーム画面を載せます。いくつかの丸がありますが、プレイヤーは黄色が 1P、青

色が 2P、紫色が 3P となっています。赤い丸は爆弾です。緑色の場所は通れなくなっており、灰色の四角形は爆弾で壊せるブロックとなっています。また、緑の枠にはステータスが書かれています。例えば上端の表示は 1P のもので、左から順に「予備残機が 1 つ」、「爆発範囲 4」、「爆弾数 3」、「速度 3」を表しています。

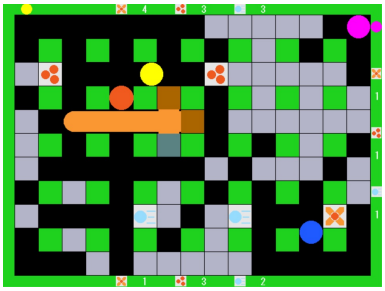


図 1　ゲームのようす

3.3　ソースコードと解説

書いたソースコードの要所を抜き取りコメントを付けながら説明していきます。なお、開発環境は Windows10 で VisualStudio を使用しており、言語は C++です。また、DX ライブラリというライブラリを用いています。

3.3.1　変数の定義と初期化

まずは GameMain 関数内で使う変数について説明します。

```
const int lives = 2; //各キャラの全残機数

float player1_x = 40; //1P のｘ座標
float player1_y = 40; //ｙ座標
int player1_v = 1; //速度
int player1_s = 0; //状態遷移に使用
int player1_t = 0; //状態遷移に使用

int bomb1_x[9] = { 0 }; //1P の爆弾のｘ座標
int bomb1_y[9] = { 0 }; //ｙ座標
int bomb1_t[9] = { 0 }; //爆発時間に使用
int bomb1_max = 1; //爆弾数
int boom1_x[9] = { 0 }; //1P の爆弾の爆発のｘ座標
int boom1_y[9] = { 0 }; //ｙ座標
int boom1_t[9] = { 0 }; //爆発時間に使用
int boom1_length = 1; //爆発範囲
int boom1_length_temp[9] = { 0 }; //爆発範囲の維持に使用

int block[11][15] = { 0 }; //壊せるブロックおよびアイテム
for (int y = 0; y < 11; y++) {
    for (int x = 0; x < 15; x++) {
        if ((x % 2 == 0 || y % 2 == 0) && ((x > 1 && x < 13) || (y > 1 && y < 9))) {
            if (GetRand(5) > 1) block[y][x] = 1; //壊せるブロックが存在するかどうかを決める
```

```
        }
    }
}
int block_t[11][15] = { 0 }; //壊せるブロックの状態遷移に使用
```

2P,3P のものは初期座標以外 1P と同じように定義・初期化するので割愛しています。爆弾や爆発用の配列の要素数が 9 なのは、爆弾数を 9 としているためです。また、爆弾と爆発を分けているのは、例えば爆弾 1 が爆発中に爆弾 1 を置いても爆発が消えないようにするためです。爆発範囲の維持に使用する変数があるのは、これがないと爆発中に爆発範囲 UP のアイテムを取るとその爆発の範囲も増えて不自然だからです。

3.3.2 衝突の関数

プレイヤーを画面端以外の壁やブロック、爆弾の中に外から侵入させないための関数です。1 マスの障害物につき 1 回この関数を使います。

```
void coll(float *chara_x, float *chara_y, int chara_v, int who, int x1, int y1, int x2,
 int y2) {
    int col_x, col_y; //衝突する可能性の一番高い座標

    if (*chara_x < x1) col_x = x1; //左端
    else if (*chara_x > x2) col_x = x2; //右端
    else col_x = *chara_x; //端以外

    if (*chara_y < y1) col_y = y1; //上端
    else if (*chara_y > y2) col_y = y2; //下端
    else col_y = *chara_y; //端以外

    //三平方の定理を使って衝突判定
    if ((*chara_x - col_x) * (*chara_x - col_x) +
 (*chara_y - col_y) * (*chara_y - col_y) < RADIUS * RADIUS) {
        switch (who) {
        case 1:
            if (*chara_x < x1) {

                //左に移動するボタンを押していなければ勝手に左に移動（右への移動を抑制）
                if (CheckHitKey(KEY_INPUT_A) == 0) *chara_x -= chara_v;

                //上隣の行の移動で左に行き過ぎてしまわないようにする
                *chara_x = max(x1 - GAP / 2, *chara_x);
            }
            if (*chara_x > x2) {
                if (CheckHitKey(KEY_INPUT_D) == 0) *chara_x += chara_v;
                *chara_x = min(x2 + GAP / 2, *chara_x);
            }
            if (*chara_y < y1) {
                if (CheckHitKey(KEY_INPUT_W) == 0) *chara_y -= chara_v;
                *chara_y = max(y1 - GAP / 2, *chara_y);
            }
            if (*chara_y > y2) {
                if (CheckHitKey(KEY_INPUT_S) == 0) *chara_y += chara_v;
                *chara_y = min(y2 + GAP / 2, *chara_y);
```

```
            }
            break;

        case 2:
                //2P 用。省略
            break;
        case 3:
                //3P 用。省略
            break;
        }
    }
}
```

この関数はキャラクターの x 座標と y 座標を示す変数のポインタと速度、そして障害物のあるマスの 4 つの端の x,y 座標の他に、プレイヤーが誰であるか (who) を引数としています。この関数では最初に、マスのうち一番衝突している可能性のある座標を調べ、次にキャラクターがそのマスに衝突しているかどうかを判定します。衝突していた場合、switch 文を用いてプレイヤーが誰であるかを調べ、プレイヤーが壁などに入らないようにかつ離れすぎないように座標を変えます。なお、ADWS はそれぞれ 1P が左右上下に移動するためのキーです。また、GAP はオブジェクト形式マクロを用いて定数 40(1 マスの幅) を英字化してあるもの、RADIUS はその半分を英字化してあるものです。

3.3.3 爆弾の位置・タイマーの関数

爆弾の位置を決めたり爆弾用タイマーのリセットを行う関数です。爆弾を置こうとする度にこの関数が使われます。

```
void bombpoint(int player_x, int player_y, int bomb_x[], int bomb_y[], int bomb_t[],
 int bomb1_x[], int bomb1_y[], int bomb1_t[], int bomb2_x[], int bomb2_y[],
 int bomb2_t[], int bomb_max){

    int i, j;

    for (i = 0; i < bomb_max; i++) {

        //爆弾が置かれていて爆発する前であればこの爆弾の変数には何もしない
        if (GetNowCount() - bomb_t[i] < BOOMSTART) continue;

        //全てのマスの中心座標のうちキャラに一番近いものを爆弾の座標にする
        for (int y = 0; y < 11; y++) {
            for (int x = 0; x < 15; x++) {
                int point_x = x * GAP + 40;
                int point_y = y * GAP + 40;
                if ((player_x - point_x) * (player_x - point_x) + (player_y - point_y) *
 (player_y - point_y) < (player_x - bomb_x[i]) * (player_x - bomb_x[i]) +
 (player_y - bomb_y[i]) * (player_y - bomb_y[i])) {
                    bomb_x[i] = point_x;
                    bomb_y[i] = point_y;
                }
            }
        }
```

```
        int bomb_exist = 0;

        //爆弾を置こうとしている場所に他の爆弾が置かれているか判定
        for (j = 0; j < 9; j++) {
            if (GetNowCount() - bomb_t[j] < BOOMSTART && bomb_x[i] == bomb_x[j] &&
 bomb_y[i] == bomb_y[j]) bomb_exist = 1; //自分の爆弾に対して
            if (GetNowCount() - bomb1_t[j] < BOOMSTART && bomb_x[i] == bomb1_x[j] &&
 bomb_y[i] == bomb1_y[j]) bomb_exist = 1; //他の 1 人の爆弾に対して
            if (GetNowCount() - bomb2_t[j] < BOOMSTART && bomb_x[i] == bomb2_x[j] &&
 bomb_y[i] == bomb2_y[j]) bomb_exist = 1; //更に他の人の爆弾に対して
        }

        if (bomb_exist == 1) continue; //他に爆弾が置かれていたら下の行は実行しない
        bomb_t[i] = GetNowCount(); //タイマーリセット
    }
}
```

BOOMSTART はオブジェクト形式マクロで、3000 としています。爆弾を置く時間を 0 ミリ秒としたとき、爆発開始の時間である 3000 ミリ秒を表すのに使っています。一番外側のループは爆弾数の回数だけ行い、ループするごとに爆弾用の配列の要素のうち使うものが変わります。このループの中では最初に置こうとしている爆弾が他の場所に置かれていないかを判定し、置かれていればこの爆弾に変化は起きません。置かれていなければ、次に爆弾の座標をプレイヤーに一番近いマスの中心にします。そしてその座標に他の爆弾が置かれていなければ爆弾用タイマーがリセットされます。

3.3.4 爆発の描画・判定の関数

爆発を描画したり、爆発に当たった他の物の状態を変える関数です。爆発時間中にこの関数が使われます。

```
void boom(int boom_x[],int boom_y[],int boom_t[], int boom_length[], int block[][15],
 int block_t[][15], int bomb1_x[], int bomb1_y[], int bomb1_t[], int bomb1_max,
 int bomb2_x[], int bomb2_y[], int bomb2_t[], int bomb2_max,
 int bomb3_x[], int bomb3_y[], int bomb3_t[], int bomb3_max,
 int player1_x, int player1_y, int *player1_s,
 int player2_x, int player2_y, int *player2_s,
 int player3_x, int player3_y, int *player3_s, int sign, int i){

    int b;
    int j;
    int sideexist = 1; //爆発の隣に障害物があるかどうかの変数

    //横方向に、中心のマスから順番に実行
    for (int x = 0; x < boom_length[i]; x++) {

        //オレンジ色の四角形を描画
        DrawBox(boom_x[i] + sign * x * GAP - GAP / 2, boom_y[i] - GAP / 2 + 3,
 boom_x[i] + sign * x * GAP + GAP / 2 + 1, boom_y[i] + GAP / 2 - 2,
 GetColor(250, 150, 50), 1);
```

```
      //爆発範囲内にいるプレイヤーの状態を変える
      if (*player1_s % 3 == 0 && player1_y >= boom_y[i] - GAP / 2 && player1_y <=
boom_y[i] + GAP / 2 + 1 && player1_x >= boom_x[i] + sign * x * GAP - GAP / 2 &&
player1_x <= boom_x[i] + sign * x * GAP + GAP / 2 + 1) (*player1_s)++;
          //2P,3P も同様。省略

      //他の爆弾を誘爆する
      for (b = 0; b < 3; b++){
          int *bomb_x, *bomb_y, *bomb_t, bomb_max;
          bomb_x = bomb1_x;
          bomb_y = bomb1_y;
          bomb_t = bomb1_t;
          bomb_max = bomb1_max;
          if (b == 1) {
              bomb_x = bomb2_x;
              bomb_y = bomb2_y;
              bomb_t = bomb2_t;
              bomb_max = bomb2_max;
          }
          if (b == 2) {
                  //3P の物を代入。省略
          }


          for (j = 0; j < bomb_max; j++) {

              //他の爆弾が爆発範囲内にあるか?
              if(bomb_x[j] == boom_x[i] + sign*(x+1)*GAP && bomb_y[j] == boom_y[i]){

                  if(GetNowCount() - bomb_t[j] < BOOMSTART){ //その爆弾が爆発前であるか?
                      sideexist = 0;

                      //その爆弾が爆発直前であるか?
                      if (GetNowCount() - bomb_t[j] < BOOMSTART - 100) {

                          //その爆弾のタイマーを爆発する時間に変更
                          bomb_t[j] = GetNowCount() - BOOMSTART + 100;
                      }
                  }
              }
          }
      }

      if ((boom_y[i] / GAP) % 2 == 0 || boom_x[i] + sign * (x + 1) * GAP < 40 ||
boom_x[i] + sign * (x + 1) * GAP > 621) sideexist = 0; //隣に壊せない壁があるか?

      //隣に壊せるブロックやアイテムがあるか?
      else if(block[boom_y[i] / GAP - 1][boom_x[i] / GAP + sign * (x+1) - 1] != 0) {

          //壊せるブロックの状態を変化させる
          if (block[boom_y[i] / GAP - 1][boom_x[i] / GAP + sign * (x + 1) - 1] <= 2)
block[boom_y[i] / GAP - 1][boom_x[i] / GAP + sign * (x + 1) - 1] = 2;

          //アイテムの状態を変化させる
          else block[boom_y[i] / GAP - 1][boom_x[i] / GAP + sign * (x + 1) - 1] = 6;
```

```
            //壊せるブロックやアイテム用のタイマーをリセット
            block_t[boom_y[i] / GAP-1][boom_x[i] / GAP+sign*(x+1) - 1] = GetNowCount();
            sideexist = 0;
        }

        if (sideexist == 0) break; //障害物があるなら爆発を止める
    }
    if (sideexist == 1) { //最後まで障害物がなければ端の爆発へ
        DrawBox(boom_x[i] + sign * boom_length[i] * GAP, boom_y[i] - GAP / 2 + 3,
 boom_x[i] + sign * (boom_length[i] - 1) * GAP, boom_y[i] + GAP / 2 - 2,
 GetColor(250, 150, 50), 1);
        DrawCircle(boom_x[i] + sign * boom_length[i] * GAP, boom_y[i], RADIUS - 3,
 GetColor(250, 150, 50), 1, 1);

        if (*player1_s % 3 == 0 && player1_y >= boom_y[i] - GAP / 2 &&
 player1_y <= boom_y[i] + GAP / 2 + 1 &&
 player1_x >= boom_x[i] + sign * boom_length[i] * GAP - GAP / 2 &&
 player1_x <= boom_x[i] + sign * boom_length[i] * GAP + GAP / 2 + 1) (*player1_s)++;
            //2P,3P も同様。省略

    sideexist = 1; //縦方向のために 1 に戻す

        //縦方向は省略
}
```

少し長くて、引数も多いですね…。引数には爆発させたい爆弾の爆発に使う配列の他に、ブロックに使う配列、全キャラクターの座標の値と状態の変数のアドレス、全キャラクターの爆弾に使う配列、さらに+方向か-方向かを示す値 (sign = 1　または　-1)、元々のループが何回目のものでどの爆発かを示す値 (i) があります。分かりにくいですね。

この関数では 1 マスずつ動作を行います。例えば sign = 1 で横方向の場合、まずは爆弾のあるマスに爆発を描画し、そこにいるキャラクターの状態を変化させ、右隣の爆弾のタイマーを爆発開始時間に変えます。爆弾ではなく壊せるブロックやアイテムがあればその状態を変化させます。また、右隣に爆弾、壁、ブロック、アイテムのいずれかがある場合ここで爆発の動作が終了します。逆にそうでない場合は右隣のマスで同じことをします。これを爆発範囲の値の回数だけ繰り返した後、右端の爆発マスの右隣にやはり障害物がなければ、右隣マスで端の丸まった爆発の描画を行い、そこにいるキャラクターの状態を変化させます。しかし爆発はそこで終わりなので、ここでは右隣に何かがあっても何も起こしません。ところで爆発範囲の値には「変数の定義と初期化」の項で説明した＜爆発範囲の維持に使用する配列＞の値を用いるため、爆発範囲の引数が配列となっています。

3.3.5 キャラクターと爆弾・爆発のプログラム

以下では、プログラムの中心となる、GameMain 関数のループ内のソースコードの説明をします。まずはキャラクターの移動や状態変化をさせたり、爆弾に関係するソースコードです。

```
//プレイヤー 1
```

```
switch (player1_s % 3) {
case 0: //通常状態
    if (CheckHitKey(KEY_INPUT_Q) != 0 || CheckHitKey(KEY_INPUT_R) != 0)
    bombpoint(player1_x, player1_y, bomb1_x, bomb1_y, bomb1_t, bomb2_x, bomb2_y,
 bomb2_t, bomb3_x, bomb3_y, bomb3_t, bomb1_max); //爆弾の関数を実行

case 2: //復活時の無敵状態
    if (CheckHitKey(KEY_INPUT_A) != 0) player1_x -= player1_v / 2.0 + 0.5; //左に移動
    if (CheckHitKey(KEY_INPUT_D) != 0) player1_x += player1_v / 2.0 + 0.5; //右に
    if (CheckHitKey(KEY_INPUT_W) != 0) player1_y -= player1_v / 2.0 + 0.5; //上に
    if (CheckHitKey(KEY_INPUT_S) != 0) player1_y += player1_v / 2.0 + 0.5; //下に

    if (player1_s % 3 == 2 && player1_s < 3 * lives - 1 &&
 GetNowCount() - player1_t > 6000) player1_s++; //倒された時点から 6 秒後に状態遷移

case 1: //倒された後
    for (int i = 0; i < bomb1_max; i++) {
        if (GetNowCount() - bomb1_t[i] <= BOOMSTART) { //爆発する前かを判定

            //1P が通常状態であれば 1P に対して爆弾は衝突判定を持つ
            if (player1_s % 3 == 0) coll(&player1_x, &player1_y, player1_v, 1,
 bomb1_x[i] - GAP/2, bomb1_y[i] - GAP/2, bomb1_x[i] + GAP/2, bomb1_y[i] + GAP/2);
                //2P,3P も同様。省略

            DrawCircle(bomb1_x[i], bomb1_y[i], RADIUS,GetColor(240, 90, 30));//爆弾の描画

            //爆弾が爆発する 1 秒前以降のとき、爆発の座標とタイマーに爆弾のものを代入
            if (GetNowCount() - bomb1_t[i] > BOOMSTART - 1000) {
                boom1_x[i] = bomb1_x[i];
                boom1_y[i] = bomb1_y[i];
                boom1_t[i] = bomb1_t[i];
            }
        }

        //爆発時間中なら爆発の関数を実行
        if (GetNowCount() - boom1_t[i] > BOOMSTART && GetNowCount() - boom1_t[i] <
 BOOMEND) {

            if (GetNowCount() - boom1_t[i] < BOOMSTART + 20)
 boom1_length_temp[i] = boom1_length; //爆発直後にその時点での爆発範囲の値を保存

            boom(boom1_x, boom1_y, boom1_t, boom1_length_temp, block, block_t,
 bomb1_x, bomb1_y, bomb1_t, bomb1_max, bomb2_x, bomb2_y, bomb2_t, bomb2_max,
 bomb3_x, bomb3_y, bomb3_t, bomb3_max, player1_x, player1_y, &player1_s,
 player2_x, player2_y, &player2_s, player3_x, player3_y, &player3_s, -1, i);

            //上の行の関数呼び出しの、後ろから 2 つ目の引数を 1 に変えて再度関数を呼び出す。省略
        }
    }

    //キャラが倒されて 2 秒後に座標移動と状態遷移
    if (player1_s % 3 == 1) {
        if (GetNowCount() - player1_t > 3000) player1_t = GetNowCount();
        if (GetNowCount() - player1_t > 2000) {
            player1_x = 40;
```

```
            player1_y = 40;
            player1_s++;
        }
    }
}
```

2P,3P に対してもやっていることは同じなので割愛。この部分ではキャラクターの状態の変数の値によって実行の開始位置が変わります。値は 0 から始まり、通常状態→（爆発に当たると）倒された状態→（2 秒経過で）無敵状態→（さらに 4 秒経過で）通常状態→…の順に状態が変わっていき、その度に値が 1 ずつ増えます。ただしこの値が 4 を超えると無敵状態→通常状態の遷移が行われなくなるため、最終的に値は 5 で止まってしまい、爆弾の設置ができなくなります。

爆弾に関しては、まず BOOMEND はオブジェクト形式マクロで 4000 としています。爆弾は置いた瞬間を 0 ミリ秒とすると、2001 ミリ秒〜3000 ミリ秒の間は爆発の変数がこの爆弾の爆発に備えて準備され、3001 ミリ秒〜3999 ミリ秒の間は爆発を行います。爆発時間が 1 秒ほどであるため、爆発が終わるまでに爆発の変数が新しく置かれた爆弾用に準備されることはありません。

3.3.6 ブロックとアイテムのプログラム

壊せるブロックとアイテムに関するソースコードです。壊せない壁があるところや 4 隅とその隣を除いたすべてのマスでそれぞれ独立な動きをします。

```
//壊せるブロック、アイテム
for (int y = 0; y < 11; y++) {
    for (int x = 0; x < 15; x++) {
        switch (block[y][x]) {
        case 0: break; //何もなし

        case 1: //壊せるブロック
            coll(&player1_x, &player1_y, player1_v, 1, x * GAP + 21, y * GAP + 21,
 x * GAP + 60, y * GAP + 60);
            coll(&player2_x, &player2_y, player2_v, 2, x * GAP + 21, y * GAP + 21,
 x * GAP + 60, y * GAP + 60);
            coll(&player3_x, &player3_y, player3_v, 3, x * GAP + 21, y * GAP + 21,
 x * GAP + 60, y * GAP + 60);
            break;

        case 2: //ブロックが破壊されている最中
                //衝突判定の関数を用いるコードがあるが、case 1 の部分とほぼ同じなので省略
            if (GetNowCount() - block_t[y][x] > 10) {
                int rand = GetRand(4);
                if (rand < 2) block[y][x] = 0; // ４０％の確率で０になりアイテムは出ない
                else block[y][x] = rand + 1; //３〜５の値になる
            }
            break;

        case 3: //爆発範囲ＵＰアイテム
            if (player1_s % 3 == 0 && player1_x >= x * GAP + 20 &&
```

```
 player1_x <= x * GAP + 61 && player1_y >= y * GAP + 20 && player1_y <= y * GAP + 61){
                block[y][x] = 0; //状態遷移
                if (boom1_length < 9) boom1_length++; //1P の爆発範囲+1
            }
                //2P,3P も同様。省略
            break;

            //case 4,5 では case 3 と同様に爆弾数ＵＰ、速度ＵＰのアイテムのコードがあるので省略

        case 6: //アイテムが破壊されている最中
            if (GetNowCount() - block_t[y][x] > 10) {
                block[y][x] = 0;
            }
            break;
        }
    }
}
```

それぞれのマスに何があるかをブロックの変数の値で決め、switch 文を用いて分岐させています。なお、case 0 以外ではブロックやアイテムの描画も行っていますが割愛しています。爆発の関数において、ブロックやアイテムに爆発が当たると状態が遷移するので破壊中になり、破壊中に爆発が当たっている間はタイマーがリセットされ続けます。そして触れている爆発が無くなった直後に再び状態が遷移し、アイテムが出てきたリ何もないマスになったりします。そのため、例えばわずかに時間を置いて 2 つの爆発が当たった場合、1 つの爆発だけが当たった場合より破壊されている状態の時間が少しだけ長くなります。

3.4 おわりに

ひとまずは、初めてプログラミングを習って初めて何かを作って完成までたどり着けて嬉しかったです。しかし、とても満足ができると言える出来ではありません。不具合が少しあり、ゲーム自体も浅い内容なので、もっと要素を追加できればいいなと思います。NPC の作成も行いたいです。

それとは別に、ソースコードを書く力も全く足りないなと痛感しました。今回のコードは思いつくままにどんどん書き込んでいったために、煩雑になったり同じようなコードを繰り返すことになってしまいました。ゲーム作成を通して、例えば関数を増やしたり構造体を利用するなどして、最低でも自分が編集したり読みやすいようにコンパクトなものを作るべきだなと感じました。次回に何かをプログラミングで作成ときは、そのあたりを特に気を付けたいと思います。

3.5 参考文献

「ＤＸライブラリ置き場」, http://dxlib.o.oo7.jp/, 2016/9/15 アクセス.

4 内部から見るゲーム

情報工学課程 1 回生 上北裕也

4.1 はじめに

今回、Lime に記事をいただいたのですが、作品のコードはぐちゃぐちゃで自信がないのであえてコードにはあまり触れないでおこうと思います。では何を書くのかといいますと、私のゲームの中のある一定の部分を取り上げ、どのような仕組みになっているのかを主に説明していきます。そのため、高度なコード（ハイセンスギャグ）を求める人はがっかりするかもしれません。

4.2 私の作ったゲーム ガンダルク

今回私が制作したのは、ゼルダ風のアクション RPG。といっても、グラフィックは○のみ、背景も地形もないファミコン以下の代物ですが…

敵を倒して経験値とお金をためるのですが、乱獲が過ぎると剣にかかった呪いにむしばまれるので、そうなる前に聖なる泉で剣を清めるか、動きを止めて呪いに打ち勝たなければなりません。敵を倒してレベルアップ。お金はマップのどこかにあるショップでアイテムを買うのに使えます。

おそらくコンピュータ部で展示してあります。よければぜひ一度。

4.3 ステップ 0 ゲームの骨組み

それでは、ゲームの解説をしていくのですが、そもそも、ゲームはどういう構造になっているのでしょうか。

一言でいうと、ゲームのプログラムはある一定の繰り返し（ループ）をずーっと行っており、その間にプレイヤーが何かボタンを押したりすると、それに応じて画面を変化させるのです。なので、ゲームの骨組みはループであるともいえます。といっても、少しわかりにくいかもしれないので、次のステップで具体例を示していきます。

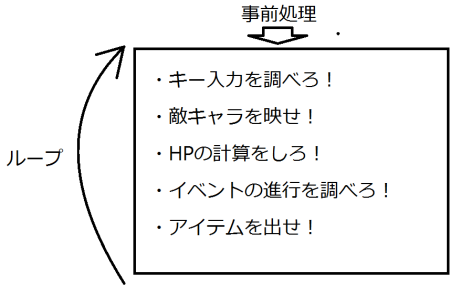


図 4.1: これがゲームの屋台骨だっ！

4.4 ステップ1 キャラを動かす

私の作ったゲームを見てみましょう。主人公はキー操作で自由に動けますが、これはどうやって動いているのでしょう。簡単に説明すると、ゲームはアニメと同じで、高速でキャラの絵を何度も表示してそれをうごかしているのです。例えば、スーパーマリオブラザーズのクリボーは、一定時間ごとに左右を反転させながら少しずつ表示の位置を左にずらして、まるで歩いているかのように見せているのです。

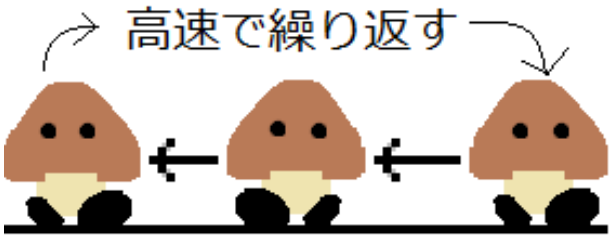


図 4.2: 某ザコの歩き方

では、少しずつキャラの位置を動かすにはどうすればいいのでしょう。ここで登場するのが、先ほど登場したループです。ループを一周するごとにキャラを表示する位置を少しだけ変えるのです。基本的に敵キャラも主人公も、この仕組みで動いています。主人公の場合、特別な理由がなければキーを押したときだけ動くようにしないといけません。このためには、ループの中に「このキーが押されていればキャラの位置をこの方向にずらす」というコードを仕込んでやればいいのです。これで、「キーの方向に動く主人公」ができました。

以下、ソースコードの「移動」に関する部分です。

```
    /*キー取得*/

    GetHitKeyStateAll(KeyBuf);

    if (KeyBuf[KEY_INPUT_LSHIFT] == 1 || KeyBuf[KEY_INPUT_RSHIFT] == 1) {
        if (item[2].flug == 1) {
```

```
            movebonus = 2;
        }
    }
    else {
        movebonus = 0;
    }
    if (KeyBuf[KEY_INPUT_LEFT] == 1) {
        player_x -= 2 + movebonus;
        player_status = 3;
    }
    else if (KeyBuf[KEY_INPUT_RIGHT] == 1) {
        player_x += 2 + movebonus;
        player_status = 1;
    }
    else if (KeyBuf[KEY_INPUT_UP] == 1) {
        player_y -= 2 + movebonus;
        player_status = 0;
    }else if (KeyBuf[KEY_INPUT_DOWN] == 1) {
        player_y += 2 + movebonus;
        player_status = 2;
    }
```

4.5　ステップ2　敵キャラのしくみ

先ほどは主人公を作りましたが、それでは、周りを歩き回る敵たちはどうでしょう。

移動の方法は主人公と同じですが、敵は主人公とは逆に勝手に動かさなければなりません。このゲームでは、少々単純ですが、骨組みのループ（以下メインループ）の回数をカウントしておき、それが一定の倍数になるたび、ランダムに上下左右のうちから向きを決定し、その方向にしばらく歩かせて、また向きを変える…ということを繰り返しています。

さて、これで動かしていいかというと、残念ながら答えはノーです。これでは、せっかくの敵キャラが勝手に画面の外に消えていくのです。これは、敵の位置を表す「x 座標（横の位置）」と「y 座標（縦の位置）」の数値が大きくなりすぎたり、小さくなりすぎたために起こるのです。パソコンのウインドウに限界はありますが、これらの数値にはありません。なので、限界を作ってあげればいいのです。このゲームのウインドウは横 640、縦 480 なので「x 座標が 640 以上なら x 座標を 640 に、0 以下なら 0 にする」と書き込んでおくのです。このゲームでは敵の向きの決め方の問題で、これだけでは敵がしばらく画面端で棒立ちしてしまうので、同時に「向きを反転させる」命令も書き込んでおきます。同様のことを y 座標でも行うと、ようやく敵を動かせます。

```
    for (i = 0; i < 4; i++) {
        //移動
        if (time >= 200) {
            enemy[i].status = GetRand(3);
        }
        if (enemy[i].x >= 624) enemy[i].status = 3;
        if (enemy[i].x <= 16) enemy[i].status = 1;
        if (enemy[i].y >= 432) enemy[i].status = 0;
        if (enemy[i].y <= 16) enemy[i].status = 2;
```

```
switch (enemy[i].status) {
case 0:
    enemy[i].y -= 2 + enemydata[enemy[i].number].speed;
    break;
case 1:
    enemy[i].x += 2 + enemydata[enemy[i].number].speed;
    break;
case 2:
    enemy[i].y += 2 + enemydata[enemy[i].number].speed;
    break;
case 3:
    enemy[i].x -= 2 + enemydata[enemy[i].number].speed;
    break;
}
```

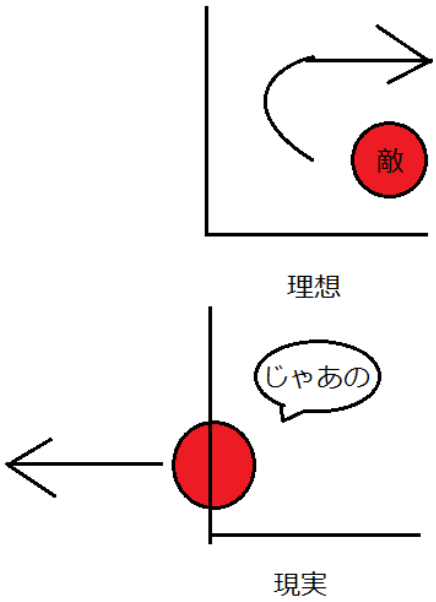


図 4.3: これが現実ッ…！

4.6 ステップ3 攻撃判定、あたり判定

さて、これで敵キャラと主人公ができました。あなたがこのゲームの製作者なら、つぎは何をしますか？おそらく、敵に当たったらダメージ（ないし何らかのイベント）を受ける

ようにしたいのではありませんか？逆に、主人公が敵をやっつけられるようにもしたいですよね。

まず敵との衝突があったかどうかの判定（あたり判定）について話します。これについてですが、私は敵キャラと主人公を正方形と考え、２つの正方形が重なれば敵と主人公が衝突した、とすることとしました。では、２つの正方形が重なったかどうかはどう判定するのかというと、まず正方形の中心の座標をお互いに出し、ｘ座標、ｙ座標同士の差を求め、その絶対値がどちらもお互いの正方形の長さの和の半分より小さければ重なっています。分かりづらかったかもしれませんが、これで敵にあたったことをコンピュータに認識させられます。

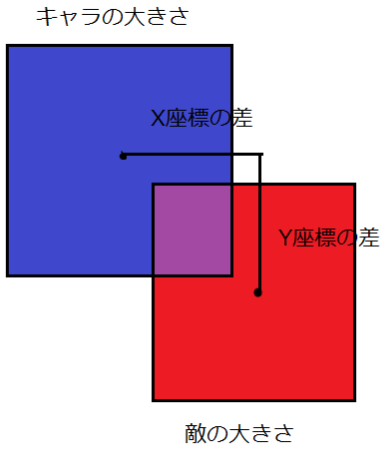


図 4.4: 当たり判定のしくみ

では、次に攻撃判定のある剣を出させます。SPACE キーを押している間、自分の向いている方向に小さな細長い三角形を描写します。そしてこの三角形を長方形とみなし、同じ要領であたり判定を作ります。これで、敵に剣があたったら敵を消す、といったことが可能になるのです。

```
       if (KeyBuf[KEY_INPUT_SPACE] == 1) {
           tate = 0; //剣を出す
           switch (player_status) {
           case 0:
               DrawTriangle(player_x, player_y - (16 + swordsize), player_x + 4,
player_y - 16, player_x - 4, player_y - 16, GetColor(255, 0, 0), TRUE);
               for (i = 0; i < 4; i++) {
                   if (abs(enemy[i].x - player_x) < 20 &&
abs(enemy[i].y - (player_y - 16 - swordsize/2)) < 32) {
                       enemy[i].hp -= hero.atk;
                       enemy[i].y = max(enemy[i].y - 80, 16);
                   }
               }
```

```
            for (i = 0; i < 2; i++) {
                if (abs(boss[i].x - player_x) < 36 && abs(boss[i].y -
(player_y - 16 - swordsize / 2)) < 48 && boss[i].flug == 1 && boss[i].muteki != 1) {
                    boss[i].hp -= hero.atk;
                    boss[i].mtime = (time + 20) % 200;
                    boss[i].muteki = 1;
                }
            }
            break;
        case 1:
            DrawTriangle(player_x + (16 + swordsize), player_y, player_x + 16,
 player_y + 4, player_x + 16, player_y - 4, GetColor(255, 0, 0), TRUE);
            for (i = 0; i < 4; i++) {
                if (abs(enemy[i].x - (player_x + 16 + swordsize / 2)) < 20 &&
abs(enemy[i].y - player_y) < 32) {
                    enemy[i].hp -= hero.atk;
                    enemy[i].x = min(enemy[i].x + 80, 624);
                }
            }
            for (i = 0; i < 2; i++) {
                if (abs(boss[i].x - (player_x + 16 + swordsize / 2)) < 36 &&
 abs(boss[i].y - player_y) < 48 && boss[i].flug == 1 && boss[i].muteki != 1) {
                    boss[i].hp -= hero.atk;
                    boss[i].mtime = (time + 20) % 200;
                    boss[i].muteki = 1;
                }
            }
            break;
        case 2:
            DrawTriangle(player_x, player_y + (16 + swordsize), player_x + 4,
player_y + 16, player_x - 4, player_y + 16, GetColor(255, 0, 0), TRUE);
            for (i = 0; i < 4; i++) {
                if (abs(enemy[i].x - player_x) < 20 && abs(enemy[i].y - (player_y + 16
+ swordsize/2)) < 32) {
                    enemy[i].hp -= hero.atk;
                    enemy[i].y = min(enemy[i].y + 80, 432);
                }
            }
            for (i = 0; i < 2; i++) {
                if (abs(boss[i].x - player_x) < 36 && abs(boss[i].y - (player_y + 16
+ swordsize / 2)) < 48 && boss[i].flug == 1 && boss[i].muteki != 1) {
                    boss[i].hp -= hero.atk;
                    boss[i].mtime = (time + 20) % 200;
                    boss[i].muteki = 1;
                }
            }
            break;
        case 3:
            DrawTriangle(player_x - (16 + swordsize), player_y, player_x - 16,
player_y + 4, player_x - 16, player_y - 4, GetColor(255, 0, 0), TRUE);
            for (i = 0; i < 4; i++) {
                if (abs(enemy[i].x - (player_x - 16 - swordsize/2)) < 20
&& abs(enemy[i].y - player_y) < 32) {
                    enemy[i].hp -= hero.atk;
                    enemy[i].x = max(enemy[i].x - 80, 16);
```

```
                }
            }
            for (i = 0; i < 2; i++) {
                if (abs(boss[i].x - (player_x - 16 - swordsize / 2)) < 36 &&
abs(boss[i].y - player_y) < 48 && boss[i].flug == 1 && boss[i].muteki != 1) {
                    boss[i].hp -= hero.atk;
                    boss[i].mtime = (time + 20) % 200;
                    boss[i].muteki = 1;
                }
            }
            break;
        }
    }
```

4.7　ステップ 4　RPG らしく

上記の段階では敵は剣が当たれば一撃で倒せますが、私のゲームでは敵に剣を当てても一撃で倒せるとは限りません。もちろん逆もしかりです。これはお互いに耐久力と攻撃力のパラメータが存在するからです。これは一見すると攻撃力と耐久力を設定して、ダメージを受けた時耐久力から攻撃力分の値を引くだけでよさそうですが、そうは問屋が卸してくれません。これで動かすと、おそらく敵はよほど HP を高くしない限り一撃で倒れます。

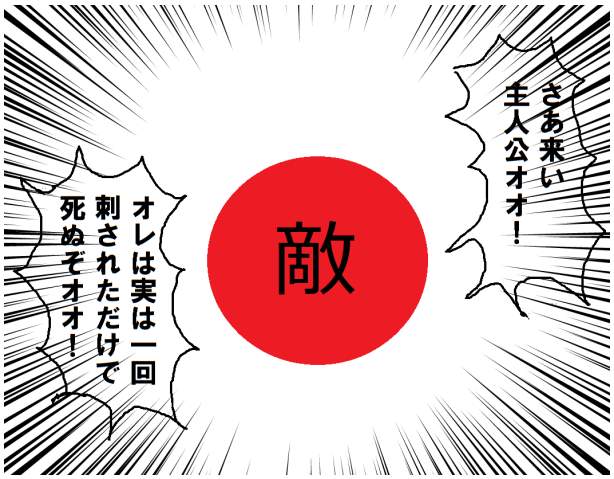


図 4.5: 無敵時間がないとこんなことに

この原因は、メインループが一周するたびにダメージ判定が行われるからです。つまり、剣から敵が離れるまで、敵は何度もダメージを食らうわけです。メインループが一周するのは非常に速いので、たとえ攻撃力が 1 でもすぐに何十ものダメージが入ってしまいます。ゆえに敵一体一体につき「無敵時間」をつくる必要があります。これはダメージを受けた後、次にダメージが通るようになるまでに必要な時間のことで、この間はあたり判定をスキップします。マリオでダメージを食らったとき、点滅してしばらく無敵になりますよね？あれが

無敵時間です。でも実はこのゲームの雑魚にはこの無敵時間を作らずに、敵をノックバック(後ろに吹っ飛ぶ) させ、剣から離すことでごまかしています。ゆえに敵を壁際に追い詰めると…（プレイヤー側には用意してあるのでご安心を。）

```
          if (abs(enemy[i].x - player_x) <32 && abs(enemy[i].y - player_y) < 32
&& enemy[i].exsit == 1 && muteki != 1) {
              hero.hp -= enemy[i].atk;
              mtime = (time + 100) % 200;
              muteki = 1;
          }
```

4.8 ステップ5 敵弾

さて、同じ敵ばかりでも面白くないので、ここらで敵に種類を作ります。HP や攻撃力はもちろんのこと、移動が早い敵や、弾（たま）を撃ってくる敵も作ります。ここでは、弾を打つ敵の仕組みを解説していきたいと思います。

弾は以前雑魚の移動方向を決めるのに使った「メインループのループ回数」を再利用しています。これが一定の倍数を取ると、雑魚が弾を放ちます（死亡時は放たない)。弾も雑魚のように座標と向きを決めて、あたり判定を作ります。ただし、これだけでは次の弾を撃つとき前の弾が画面から消えてしまうので、連射できるよう複数弾のデータを用意するか、弾が画面に残っている間は新しい弾を発射しないようにしなければなりません。また、弾が画面外に出たときはちゃんと弾が消えたことにしないといけません。あとは、キャラと同じようにメインループ一周ごとに少しずつ動かします。

```
          for (g = 0; g < 4; g++) {
              if (tama[i][g].exsit == 1) {
                  switch (tama[i][g].status) {
                  case 0:
                      tama[i][g].y -= 3 + enemydata[enemy[i].number].speed;
                      break;
                  case 1:
                      tama[i][g].x += 3 + enemydata[enemy[i].number].speed;
                      break;
                  case 2:
                      tama[i][g].y += 3 + enemydata[enemy[i].number].speed;
                      break;
                  case 3:
                      tama[i][g].x -= 3 + enemydata[enemy[i].number].speed;
                      break;
                  }
                  DrawCircle(tama[i][g].x, tama[i][g].y, 8, Cr[enemy[i].number], FALSE);
                  if (abs(tama[i][g].x - player_x) < 24 && abs(tama[i][g].y - player_y)
< 24 && muteki != 1 ) {
                      hero.hp -= enemy[i].atk;
                      mtime = (time + 100) % 200;
                      muteki = 1;
                      tama[i][g].exsit = 0;
                  }
                  if (tama[i][g].x >= 640 || tama[i][g].x <= 0 || tama[i][g].y >= 448
```

```
|| tama[i][g].y <= 0) {
                    tama[i][g].exsit = 0;
                }
            }
        }
```

4.9　ステップ6　世界を広げる

感のいい方はもう気付いていらっしゃったかもしれませんが、私は敵が画面から出ないようにはしていましたが、主人公は出れるようにしていました。ここまで放っておいた理由は、複数の部屋を作り、そこをプレイヤーが行き来できるようにしたかったからです。

部屋は8掛ける8の64部屋つくり、それぞれに0〜64の番号がつけました。左上から部屋が正方形に並んだ（ように見せる）ようにして、画面から主人公が出ると抜けた方向の部屋に移動するようにします。また、データを用意していない場所に移動されないように、番号で移動に制限を設けます。難しく聞こえますが、要するに大きな正方形の4辺からそとに出ないようにするだけです。

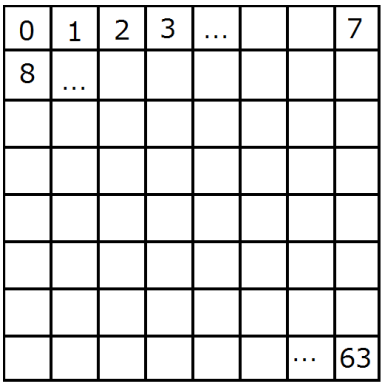


図 4.6: よくわかるルームのつながり

また、部屋を移動すると、敵が最初の配置に戻って復活します。これは、敵の配置戻しや復活などを部屋移動のたびに行っているからです。これをしないと、前の画面の敵が襲ってきます。またこの時、部屋に応じた強さの敵が出るように敵を変えましてす。

```
    //部屋の切り替え
    if (player_x > 640) {
        if (nowroom % 8 != 7 && boss[0].flug !=1 && boss[1].flug != 1) {
            player_x = 0;
            nowroom += 1;
            syokika = 1;
        }
        else {
            player_x = 640;
        }
```

```
}else if (player_x < 0) {
    if (nowroom % 8 != 0 && boss[0].flug != 1 && boss[1].flug != 1) {
        player_x = 640;
        nowroom -= 1;
        syokika = 1;
    }
    else {
        player_x = 0;
    }
}
if (player_y > 448 ) {
    if (nowroom < 56 && boss[0].flug != 1 && boss[1].flug != 1) {
        player_y = 0;
        nowroom += 8;
        syokika = 1;
    }
    else {
        player_y = 448;
    }
}
else if (player_y < 0) {
    if (nowroom >= 8 && boss[0].flug != 1 && boss[1].flug != 1) {
        player_y = 448;
        nowroom -= 8;
        syokika = 1;
    }
    else {
        player_y = 0;
    }
}

//出現敵選択
enemy[0].number = int(nowroom / 8);
enemy[1].number = int(nowroom / 8);
enemy[2].number = int(nowroom % 8);
enemy[3].number = 1 + int(nowroom / 10);

//初期化
if (syokika == 1) {
    for (i = 0; i < 4; i++) {
        if (nowroom != 27 && nowroom != 64) {
            enemy[i].exsit = 1;
            enemy[i].hp = enemydata[enemy[i].number].hp;
            enemy[i].atk = enemydata[enemy[i].number].atk;
            enemy[i].x = 160 + (320 * (i % 2));
            enemy[i].y = 120 + (240 * int(i / 2));
        }
        else {
            enemy[i].exsit = 0;
        }
        for (g = 0; g < 4; g++) {
            tama[i][g].exsit = 0;
        }
    }
}
```

4.10　作った感想

さて、ここからは思いつくままに要素を増やしていきました。タイトル画面、ゲームオーバー、ショップ…中でも苦労したのは剣のゲージです。ゲージが減っていくにつれて緑→黄→赤と変わるようにしたかったからです。黄色は赤と緑の中間だったので、ゲージのちょうど半分で赤も緑も強くなるように工夫してようやく完成しました。また、ショップ画面ではアイテムの横にカーソルが出ますが、これも早く動かしすぎないようにするのに苦労しました。でもなんやかんやで結構楽しんでました。

4.11　おわりに

私はプログラム歴が長いとは言えませんが、この作品は私の中で一番手の込んだものとなりました。あまり出来がいいとは言えないかもしれませんが、私はとにかく自分の思いを形にできたので本当に満足しています。Lime は初投稿なので、至らぬ点が多かったと思いますが、最後までお付き合いいただき誠にありがとうございました。

図 4.7: アクション RPG　ガンダルク

5 ダンジョン攻略ゲームとホッケーゲーム

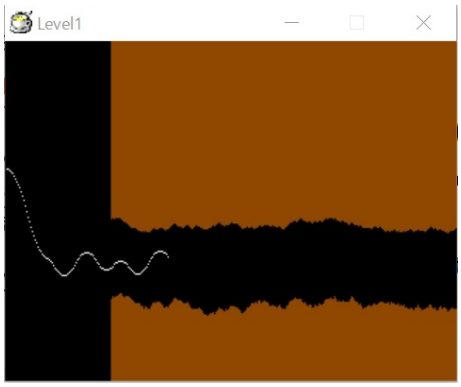
情報工学課程 1 回生 馬杉康平

5.1 はじめに

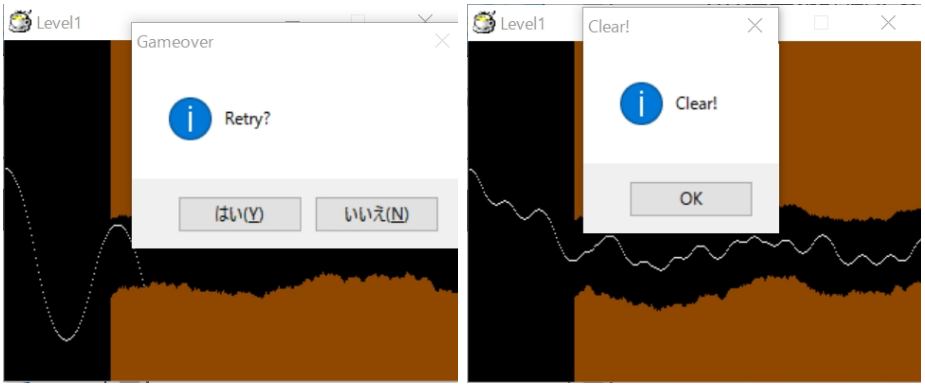
今回私は Hot Soup Processor(以下 HSP) を使ってプログラミング初心者にも簡単に作れるゲームを 2 つ作りました。HSP とは C 言語や Java などと比べて文法が簡潔で覚えやすく、プログラミング初心者でもある程度しっかりしたプログラムを作ることができる言語です。私は Lime 用に「洞窟通り抜けゲーム」と「ホッケーゲーム」を作りました。Lime では各ゲームのソースコードを書き、それらについて解説をします。ソースコードの左端には行番号を振ってありますが、便宜上のものであり、実際にソースコードを書く際には行番号を振る必要はないのでご注意ください。

5.2 洞窟通り抜けゲーム

洞窟通り抜けゲームとは、自機 (白い点) を操作して洞窟の右端を目指すゲームです。
自機は、スペースキーを押している間は上昇し、スペースキーを離している間は下降します。



自機が画面の外に出るか、洞窟の壁にぶつかるとゲームオーバーです。無事に洞窟の右端にたどり着けたらクリアとなり、次のレベルへ進めます。



以下は洞窟通り抜けゲームのソースコードです。

```
1:  level=1                          //level... 現在のレベル
2:  //セッティング
3:  *setting
4:     randomize                     //乱数パターン初期化
5:     screen 0,640,480
6:     my_x=0.0                      //my_x... 自機の x 座標
7:     my_y=180.0                    //my_y... 自機の y 座標
8:     my_dy=0.25                    //my_dy... 自機の y 座標の増加量
9:     kabe_x=150                    //kabe_x... 壁を形成する線の x 座標
10:    uekabe=250                    //uekabe... 上の壁を形成する線の長さ
11:    sukima=110                    //sukima... 隙間の広さ
12:    color 0,0,0                   //塗りつぶす色を黒にする
13:    boxf
14:
15:  //レベル表示
16:  *leveldisplay
17:    title "Level"+level
18:
19:  //コース作成
20:  *coursemake
21:     a=rnd(level*2+1)-level        //a...uekabe の変化量
22:     b=rnd(level*2+1)-level       //b...sukima の変化量
23:     uekabe=uekabe+a              //uekabe,sukima を変化させる
24:     sukima=sukima+b
25:     if uekabe<10:uekake=10      //壁の長さを制限
26:     if uekabe>350:uekabe=350
27:     if sukima<50:sukima=50
```

```
28:     if sukima>110:sukima=110
29:     color 144,72,0
30:     line kabe_x,uekabe,kabe_x,0                  //上壁の描画
31:     line kabe_x,480,kabe_x,(uekabe+sukima)       //下壁の描画
32:     if kabe_x>=640:goto *main     //右端まで描画し終えたらループを抜ける
33:     kabe_x=kabe_x+1
34:     goto *coursemake
35:
36:  //メインループ
37:  *main
38:
39:     redraw 0
40:
41:     stick spacekey,16
42:     if spacekey=16{            //スペースキーが押されているかを判定
43:        my_dy=my_dy-0.25
44:     }else{
45:        my_dy=my_dy+0.25
46:     }
47:
48:     my_x=my_x+2.0             //自機の座標を変化させる
49:     my_y=my_y+my_dy
50:
51:     pget my_x,my_y
52:     if ginfo_r=144:goto *gameover     //ゲームオーバー判定
53:     if my_y>480:goto *gameover
54:     if my_y<0:goto *gameover
55:     if my_x>640:goto *clear          //クリア判定
56:     color 255,255,255
57:     pset my_x,my_y                  //自機を描画
58:     redraw 1
59:
60:     await 50
61:     goto *main
62:
63:  //ゲームオーバーの処理
64:  *gameover
65:     wait 100
66:     dialog "Retry?",2,"Gameover"
```

```
67:     if stat=6{
68:        level=1
69:        goto *setting
70:     }else{
71:        end
72:     }
73:  //クリア処理
74:  *clear
75:     dialog "Clear!",0,"Clear!"
76:     level=level+1
77:     if level>20:level=20
78:     goto *setting
```

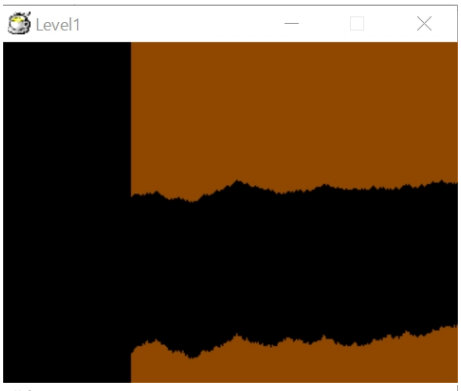
5.3 洞窟通り抜けゲームのソースコードの解説

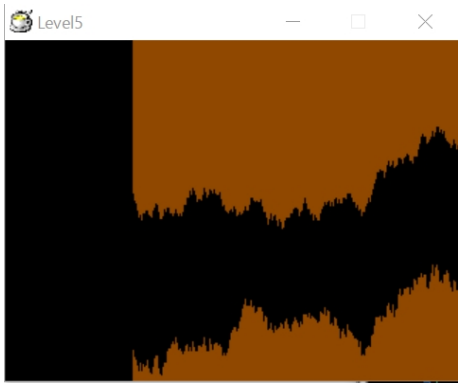
- 1 行目
  ゲーム起動時にレベルの初期化を行います。

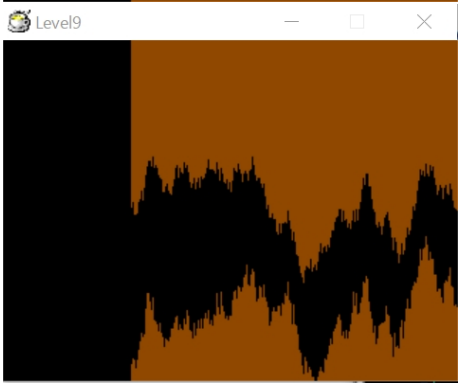
- 3 行目～13 行目
  *setting ラベルで, 乱数パターンの初期化、変数の初期化、背景の描画を行っています。
  最初に randomize 命令で乱数のパターンを不定値で初期化しています。rnd 関数を使用すると乱数を発生させることができますが、そのパターンは常に一定となってしまいます。そこで、randomize 命令をパラメータなしで入れておくと乱数のパターンが不定値で初期化されるので毎回まったく違う乱数を発生させることができます。
  また、boxf 命令はパラメーターをすべて省略すると画面全体を塗りつぶします。本プログラムでは画面全体を黒で塗りつぶしています。

- 16 行目～17 行目
  タイトルバーに現在のレベルを表示します。

- 20 行目～34 行目
  *coursemake ラベルでコースを作成します。「線を引く→線を引く位置の x 座標を 1 ずらす→線を引く→ … 」という具合にして作成します。ここでは変数 uekabe、sukima の増加量としてそれぞれ変数 a、b を用意し、*coursemake ラベルのループ内で線を引く度に a、b の値をそれぞれランダムで変え、uekabe に a を加算、sukima に b を加算しています。これにより洞窟のような絵を描くことができます。また、a、b の値はレベルが高いほどランダムで変化する値の範囲が大きいです。このことはレベルが高いほど通りにくいコースになるということを意味します。(次ページ参照)







- メインルーチン

37〜61 行目 (*main ラベル) はメインルーチンになります。

(1) 39 行目、58 行目
redraw 命令は画面の描画方法を指定する命令です。「redraw 0」で画面の書き換えを反映させないモードにして、自機を描画する処理を行ってから「redraw 1」で画面の書き換えを反映 (画面を更新) させています。そうすると、画面のちらつきを無くし、処理速度が速くなります。

(2) 41〜46 行目

`stick` 命令で変数 `spacekey` にキー入力判定の役割を持たせています。スペースキーが押されている間、`spacekey` には `16` が代入されます。`spacekey` が押されている間、変数 `my_dy` は減少し続け、スペースキーが離されている間は `my_dy` が増加し続けます。

(3) 48、49 行目

`my_x` に `2.0` を、`my_y` に `my_dy` を加えています。

(4) 51〜52 行目
このプログラムでは、現在の自機の座標の色情報を取得し、その色情報について

調べることで壁と当たっているかどうかを判定しています。pget 命令で現在の自機の座標の色情報を取得しています。このとき自機が壁にぶつかった時、すなわち自機の座標の赤の輝度が 144 になったときにゲームオーバーにします。

(5) 53〜55 行目
53〜55 行目では現在の自機の座標を調べています。自機が洞窟に入る前に画面の外に出てもゲームオーバーにしています。また、自機が右端に到達したらステージクリアにしています。

(6) 56〜61 行目
color 命令で白を指定し、pset 命令で白い点を描いています。そして*main ラベルにジャンプし、*main 内に書かれていることを繰り返します。

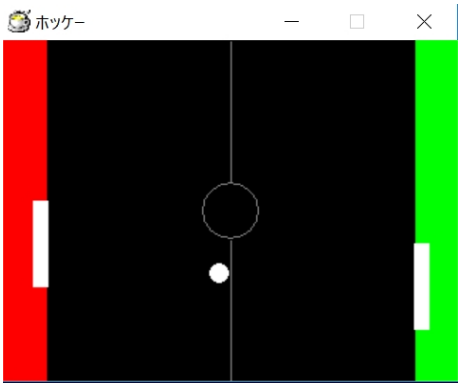
`*main` 内のプログラムを繰り返し実行することで白い点が右に進むように描画されます。`my_dy` の値は常に変化し続けるので、白い点は曲線を描きながら進みます。

- 64 行目〜72 行目
ゲームオーバー時の処理を行います。dialog 命令でリトライするかどうか尋ねるダイアログボックスを表示させています。ここで「はい」を選ぶとシステム変数 stat に 6 が、「いいえ」を選ぶと 7 が代入されます。「はい」を選ぶとレベルが 1 にリセットされラベル*setting にジャンプします。

- 74 行目〜78 行目
ステージクリア時の処理を行います。「Clear!」と書かれたダイアログボックスを表示し、OK ボタンがクリックされたらレベルを 1 上げ、*setting にジャンプします。ただし、レベルは 21 以上にはならないようにしています。

5.4　ホッケーゲーム

ゲームセンターでお馴染みのホッケーゲームです。2 人対戦用ゲームとなっています。ボールをお互いに打ち返し合い、相手のゴールにボールを入れたプレイヤーの勝ちです。プレイヤー 1(左側) は A、Z キーを使い、プレイヤー 2(右側) は矢印キーの上と下を使います。また、ボールがバーに当たるたびにボールの x 軸方向の速さが速くなります。
以下はホッケーゲームのソースコードです。




```
1:  //セッティング
2:   *setting
3:      randomize          //乱数パターンの初期化
4:      title "ホッケー"
5:
6:      screen 0,320,240   //画面サイズ変更
7:      r=rnd(9)-4         //r...ball_dy を決める乱数
8:      minus=rnd(2)       //minus... 最初にボールが移動する方向を決める乱数
9:      ball_x=160.0       //ball_x... ボールの x 座標
10:     ball_y=120.0       //ball_y... ボールの y 座標
11:     ball_dx=1.0        //ball_dx... ボールの x 座標の増加量
12:     ball_dy=(1.0*r)/2.0          //ball_dy... ボールの y 座標の増加量
13:     if minus=1:ball_dx=-ball_dx //minus が 1 なら 1P 側へ、0 なら 2P 側へ行く
14:     left_y=rnd(10)+110           //left_y... 左バーの y 座標
15:     right_y=rnd(10)+110          //right_y... 右バーの y 座標
16:  //メインループ
17:  *main
18:     redraw 0
19:     color 0,0,0                    //背景の描画
20:     boxf
```

```
21:
22:     color 255,0,0                   //←赤を指定
23:     boxf 0,0,30,240
24:     color 0,255,0                   //←緑を指定
25:     boxf 290,0,320,240
26:
27:     color 128,128,128               //←灰色を指定
28:     line 160,0,160,100
29:     line 160,140,160,240
30:     circle 140,100,180,140,0        //円を描く
31:
32:     gosub *leftbar
33:
34:     gosub *rightbar
35:
36:     gosub *judge
37:
38:     gosub *ball
39:
40:     redraw 1
41:     await 10
42:     goto *main
43:
44:  //ボール処理
45:  *ball
46:     if ball_y<=0.0 or ball_y+13.6>=240.0:ball_dy=-ball_dy
47:     ball_x+=ball_dx                 //ボールの座標を変化させる
48:     ball_y+=ball_dy
49:     pos ball_x,ball_y               //ボールを描画
50:     color 255,255,255
51:     font "MS gothic",16
52:     mes "●"
53:
54:     return
55:
56:  //左バー処理
57:  *leftbar
58:     getkey left_keyA,65          //left_keyA...A キーの押下判定
59:     getkey left_keyZ,90          //left_keyZ...Z キーの押下判定
```

```
60:     if left_keyA=1:left_y-=3
61:     if left_keyZ=1:left_y+=3
62:     if left_y<0:left_y=0
63:     if left_y>180:left_y=180
64:     color 255,255,255          //左のバーを描画
65:     boxf 21,left_y,31,left_y+60
66:
67:     return
68:
69:  //右バー処理
70:  *rightbar
71:     getkey right_keyUP,38     //left_keyUP... ↑キーの押下判定
72:     getkey right_keyDN,40     //left_keyDN... ↓キーの押下判定
73:     if right_keyUP=1:right_y-=3
74:     if right_keyDN=1:right_y+=3
75:     if right_y<0:right_y=0
76:     if right_y>180:right_y=180
77:     color 255,255,255                  //右のバーを描画
78:     boxf 289,right_y,299,right_y+60
79:
80:     return
81:
82:  //当たり判定
83:  *judge
84:     pget ball_x,ball_y+6.8             //ボールの左端の色情報を取得
85:     if ginfo_r=255 and ginfo_g=0{      //1P ゴールとぶつかったとき 2P の勝ち
86:        dialog "Retry?",2,"2p wins"
87:        if stat=6:goto *setting
88:     end
89:     }
90:     if ginfo_r=255 and ginfo_g=255{    //1P バーとぶつかったら跳ね返る
91:        ball_dx=-ball_dx                //ボールを跳ね返らせる
92:        ball_dx+=0.1                    //ボールを加速させる
93:        r=rnd(17)-8
94:        ball_dy=(1.0*r)/2.0
95:     }
96:
97:     pget ball_x+13.6,ball_y+6.8        //ボールの右端の色情報を取得
98:     if ginfo_r=0 and ginfo_g=255{      //2P ゴールとぶつかったとき 1P の勝ち
```

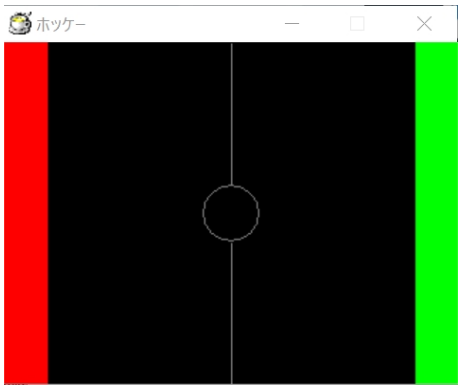
```
99:         dialog "Retry?",2,"1p wins"
100:        if stat=6:goto *setting
101:    end
102:    }
103:    if ginfo_r=255 and ginfo_g=255{    //1P バーとぶつかったら跳ね返る
104:        ball_dx=-ball_dx               //ボールを跳ね返らせる
105:        ball_dx-=0.1                   //ボールを加速させる
106:        r=rnd(17)-8
107:        ball_dy=(1.0*r)/2.0
108:    }
109:
110:    return
```

5.5 ホッケーゲームのソースコードの解説

- 2 行目～15 行目

  *setting ラベルで乱数パターンの初期化、画面サイズ変更、変数の初期化を行っています。
  変数 minus は一番最初にボールが向かう方向を決めています。

- メインルーチン
  17 行目～42 行目はメインルーチン (*main ラベル) となります。19 行目～30 行目はホッケーゲームの背景を描画しています。また、32、34、36、38 行目は gosub 命令でサブルーチンに移動しています。



- 45 行目～54 行目　サブルーチン*ball

  *ball ではボールの描画を行っています。ボールの描画には、mes 命令を用いています。また。ボールが上壁または下壁にぶつかったときには、ball_dy の符号を逆にすることでボールを跳ね返らせています。

- 57 行目～67 行目　サブルーチン*leftbar

  プレイヤー 1 のバーを描くサブルーチンです。まず getkey 命令でキーが押されているかどうかを調べる変数を用意します。本プログラムでは変数 left_keyA に A キーの判定をする役割、変数 left_keyZ に Z キーの判定をする役割を持たせています。指定したキーが押されていれば変数に 1 が、押されていなければ 0 が代入されます。キーが押されている間、バーが動きますが、壁を突き抜けることはありません。

- 70 行目～80 行目　サブルーチン*rightbar
  プレイヤー 2 のバーを描くサブルーチンです。*leftbar と同様の処理をします。

- 83 行目～110 行目　サブルーチン*judge
  ボールとゴールとの当たり判定と、ボールとバーとの当たり判定を行っています。
  ボールの左端および右端の座標の色情報を読み込んで、当たり判定を行います。

  (1) 84 行目～95 行目

  ボールとプレイヤー 1 のゴール (赤い領域) との当たり判定と、ボールとプレイヤー 1 のバーとの当たり判定を行っています。pget 命令でボールの左端の色情報を取得して、赤色が検出されたらプレイヤー 2 の勝ちにします。
  また、白色が検出されたらボール跳ね返らせ (ball_dx の値をプラスにし) ます。このとき ball_dx の値を増加させ、ball_dy の値をランダムで変えています。

(2) 97 行目～108 行目

ボールとプレイヤー 2 のゴール (赤い領域) との当たり判定と、ボールとプレイヤー 2 のバーとの当たり判定を行っています。pget 命令でボールの右端の色情報を取得して、緑色が検出されたらプレイヤー 1 の勝ちにします。
また、白色が検出されたらボール跳ね返らせ (ball_dx の値をマイナスにし) ます。このとき ball_dx の値を減少させ、ball_dy の値をランダムで変えています。

なお、勝敗が決まったらリトライするかどうかを訊き、「はい」が押されたら *setting にジャンプし、「いいえ」が押されたらプログラムを終了します。

5.6 最後に

私のプログラミングの能力が拙いため、これらのような簡単なゲームしか作れませんでしたがご容赦ください。来年はこれらよりも複雑なプログラムを作れるように一生懸命勉強していきたいと思います。

参考文献

[1] 大槻有一郎:12 歳からはじめる HSP3.0 わくわくゲームプログラミング教室 Windows98/2000/Me/XP 対応; ラトルズ

[2] おにたま, 悠黒喧史, うすあじ:最新 HSP3.3 プログラミング入門 Windows98/2000/Me/XP/Vista/7 対応; 秀和システム

6 スケジュール管理アプリ

情報工学課程 2 回生 矢口　喬脩

6.1　おことわり

Lime 記事は技術的な文書になりがちであるが、学祭での配布物であるという性質上、情報工学等に造詣が深い人のみを対象とするのは難しく、ソースコードの列挙ばかりでは飽きるとおもうので、少しふざけた調子で以下を記述しています。(技術的なものを読みたい人は読み飛ばしてください。)

6.2　はじめに

スケジュール管理アプリをググると多くのアプリがストアに並んでいる。しかし、どれも僕の欲しい機能が実装されているアプリはないように思えた。ならば、後期の Java の練習も兼ねて自分でアプリを自作してみようと考えた。アプリに求める要求仕様は以下の点に尽きる。

- キャラクターがスケジュールを管理してくれる。

ちょっと何言ってるのか分らないという人も、少し待ってほしい。これには訳がある。(要求仕様が漠然としているとかは … わざとです。) 某ソシャゲでは、ホーム画面で、キャラが時間ごとにランダムに喋りかけてくれる機能を持つものがある。それに目をとめたとき、少し癒されたということがあるのではないだろうか。(個人的感想です。) 無味乾燥な文字の羅列で、進捗を煽られたところで、誰が嬉しかろう、しかし、自分の好みのキャラクターが煽ってくれたなら、少しは一考の余地が生じるのではなかろうか。

というよくわからないモチベーションで作ったアプリの詳細を以下記載していく。(執筆時点で開発中段階)

6.3　詳細

使用統合開発環境は AndroidStudio であり、本アプリはドロワーを備えている。

現在実装されている機能

- 本日のタスク

  現在の日時を Calendar クラスから取得し、その日時と七日後の日時を求め、Google カレンダーに記載された現在から七日後までのイベントを取得し、その最直近のイベントタイトルをキャラクターが述べてくれる。カレンダーからその予定を消せば、次の予定を述べるように切り替わる。

- 時間割

  大学の時間割を preference として保管し、起動時に EditText にセットするもの。時間割の書き込みはタップして入力することで実現する。まずドロワーから時間割をタップするとキャラが曜日を選択するように要求する Fragment が開き、曜日を選択するとその曜日の時間割 Fragment が開くようになっている。自分用に作ったので、時間割の授業開始時刻は京都工芸繊維大学が定めるものに準ずる。

これから実装していく機能

- バイトシフト管理

  まだ執筆段階では構想の段階だが、アルバイトのシフトが学業に影響しないように、調整したり、今月の給料を計算し表示する機能を実装しようと考えている。

6.4 私、気になります !!! (非技術的セクション)

これだけキャラが、どうのこうの言っておいて、肝心の想定キャラのことについて、触れていないじゃないかー。という人もいると想定してこのセクションを追記しています。

想定キャラはアイドルマスターシンデレラガールズの島村卯月さんです。実際、キャラのデータを使用して開発していましたし、セリフも彼女が言いそうな、言っているセリフに調整していました。

6.5 問題発生！

しかし、事態はそう簡単にいかなかった。部誌として、対外的に配布するとなると、島村さんが著作権法に抵触する恐れが生じたのである。つまり、開発中スクリーンショットに島村さんを使用することが難しくなったのである。教育的利用だとか、個人の範囲内の使用だとか、言うのは難しくないかもしれないが、部の上の方に怒られそうなので、島村さんの使用を断念した。

6.6 ソースコードを書かせてくれ …

機能未実装は残っているのに、キャラの問題が生じた。(上記参照) そこで、誰か絵を描いてくれそうな人を探そうとしたが、ついに発見するに至らなかった。少しでもアプリを

形にするために、ソースコードよりも作画を優先せざるを得なくなってしまったのである。たとえ機能が実装されていても、キャラがなければ、最初に述べたこのアプリの意味がなくなってしまう。(グラフィッカー志望部員求む！) ペンタブとか、持っていないので、もちろん、手書きである。それをスキャンして、コンピュータ上でベタ塗りによって色をつけてそれらしく仕上げた。

6.7 君の名前は ····

せっかく描いたので、名前をつけよう、とか考えた。ソフトウェア開発で最も犠牲になるもの、それは UI だと授業で聞いた気がする。僕のアプリもなかなかにひどい UI な気がする。開発がひと段落したら、その UI も調整しよう、と思っていた矢先であったので、UI からとって、「結依」と命名した。人と機器を結ぶもの、そして、人を機器に依らせるものという意味を込めた。そして、キャラの名前は「松ヶ崎　結依」に決めた。あとで知り合いが数時間程度で作画してくれたイラストの方が綺麗で、三日ぐらいかかって、このざまの己の画力に、なにかこみ上げるものがあった。(なぜプログラムでは熱く語らないのかとか、つっこまないで ··) アプリに実装したら、なんかＣＶ欲しくなってきた ····

図 6.1: 「松ヶ崎　結依」

6.8 記述コード

コードと言っても二種類あり、Java のソースと、アプリのデザインを決める XML のソースがある。まだ開発段階ということもあり、コードは一部のみ記載する。

<MainActivity.java>アプリを全体を大まかに制御するコード。

```
package org.kitcc.taka.privatesecretary;


import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
import android.view.View;
import android.support.design.widget.NavigationView;
import android.support.v4.view.GravityCompat;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.app.ActionBarDrawerToggle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity
        implements NavigationView.OnNavigationItemSelectedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action"
```

```
 , Snackbar.LENGTH_LONG).setAction("Action", null).show();
        }
    });

    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    ActionBarDrawerToggle toggle = new ActionBarDrawerToggle(
            this, drawer, toolbar,
         R.string.navigation_drawer_open,R.string.navigation_drawer_close);
    drawer.setDrawerListener(toggle);
    toggle.syncState();

    NavigationView navigationView=(NavigationView) findViewById(R.id.nav_view);
    navigationView.setNavigationItemSelectedListener(this);
}

@Override
public void onBackPressed() {
    DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
    if (drawer.isDrawerOpen(GravityCompat.START)) {
        drawer.closeDrawer(GravityCompat.START);
    } else {
        super.onBackPressed();
    }
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();
```

```
        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @SuppressWarnings("StatementWithEmptyBody")
    @Override
    public boolean onNavigationItemSelected(MenuItem item) {
        // Handle navigation view item clicks here.
        FragmentManager fragmentManager= getSupportFragmentManager();

        int id = item.getItemId();

        if (id == R.id.nav_today_task) {
                fragmentManager.beginTransaction()
                .replace(R.id.container,Fragment.instantiate(
               MainActivity.this,"org.kitcc.taka.privatesecretary
.TaskManageFragment"))
                    .commit();
        } else if (id == R.id.nav_timetable) {
                fragmentManager.beginTransaction()
                  .replace(R.id.container,Fragment.instantiate(
               MainActivity.this,"org.kitcc.taka.privatesecretary
.SelectDateFragment"))
                        .commit();
        } else if (id == R.id.nav_shiftmanage) {


        }

        DrawerLayout drawer = (DrawerLayout) findViewById(R.id.drawer_layout);
        drawer.closeDrawer(GravityCompat.START);
        return true;
    }
}
```

<TaskManageFragment.java>本日のタスク機能を実装。

```
package org.kitcc.taka.privatesecretary;

import android.content.Context;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import java.util.Calendar;


public class TaskManageFragment extends Fragment {
    public static TaskManageFragment newInstance(Context context) {
        TaskManageFragment fragment = new TaskManageFragment();
        return fragment;
    }
    String task;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // イベント取得用 URL をセット
        Uri eventUri = Uri.parse("content://com.android.calendar/events");

// 開始日付を取得
        Calendar cal = Calendar.getInstance(); // 現在日時を取得
        cal.set(Calendar.HOUR_OF_DAY, 0);// 時を 0 でクリア
        cal.set(Calendar.MINUTE, 0); // 分を 0 でクリア
        cal.set(Calendar.SECOND, 0); // 秒を 0 でクリア
        cal.set(Calendar.MILLISECOND, 0); // ミリ秒を 0 でクリア
        Long startMillis = cal.getTimeInMillis(); // Long 値へ変換
```

```
// 終了日付を取得
        cal.add(Calendar.DATE, 7); // 7日後を計算
        Long stopMillis = cal.getTimeInMillis(); // Long値へ変換
// 開始日付と終了日付をパラメータ配列にセット
        String[] params = new String[]{"" + startMillis, "" + stopMillis};

      /*  Cursor eventCursor = getActivity().getContentResolver().query(
                eventUri,
                new String[]{"dtstart","title","allDay"},
                "allDay = 0 and dtstart >= ? and dtstart < ?",
                params,
                "dtstart asc");
*/
        Cursor eventCursor = getActivity().managedQuery(
                eventUri,
                new String[]{"dtstart", "title", "allDay"},
                "allDay = 0 and dtstart >= ? and dtstart < ?",
                params,
                "dtstart asc");
        //    eventCursor.moveToFirst(); // 最初のイベントに移動
        if (eventCursor.moveToFirst()) {
            task = eventCursor.getString(1);
            for (int i = 0; i < eventCursor.getCount(); i++) {
           // ログに予定タイトルを出力
                 Log.d("event", eventCursor.getString(1));

                eventCursor.moveToNext(); // 次のイベントに移動
            }
        }
        else{
            task=null;
        }
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {

        View v=inflater.inflate(R.layout.fragment_task_manage, container, false);
```

```
        TextView textView=(TextView) v.findViewById(R.id.yui_comment);
        if(task!= null) {
            textView.setText("お疲れ様です。プロデューサーさん。てへっ。\n" + "
直近のお仕事は「" + task + "」ですね !! \n"+"今日もがんばるぞい、です〜！");
        }
        else{
            textView.setText("いつも、お疲れ様です。\n" +"お仕事はありませんよ
〜\n 休みもちゃんと取らないとダメですよ〜");
        }
        // Inflate the layout for this fragment

        return v;
    }
}
```

6.9 スクリーンショット

画面は開発中のものです。
以下の画像は時間割機能のものです。
やるべき課題があるときは結依ちゃんが煽ってくれます。
やるべき課題が何もない時は結依ちゃんが労ってくれます。

図 6.2: 曜日選択画面から各曜日設定へ

図 6.3: 課題ありモード (本日のタスク機能)

図 6.4: 課題なしモード (本日のタスク機能)

6.10 そして、これから …

学祭でも展示すると思いますが、これからまた作り込んでいく予定です。来年の発表にぜひご期待ください。最後まで読んで頂きましたこと末筆にはなりますが、御礼申し上げます。

6.11 参考文献

http://qiita.com/chocomelon/items/c94b4c8d12c11dfbe898
http://blog.livedoor.jp/itahidamito/archives/51616012.html

本アプリは
2017年秋
完成予定!!

編集後記

編集担当の矢口です. 今回で 53 冊目の Lime になります. 松ヶ崎祭自体が開催から危ぶまれるようなこともあり、制作から編集までなかなかハードな日程になっていましたが、今年も無事発行することができました. 今年は hackathon もあり、展示のみで店舗運営はなしとのことであり、展示運営に集中しているので、例年よりも完成度が高いのではないかと、わくわくしながら各記事を読みながら編集作業をさせていただきました. 記事の内容は楽しんでいただけたでしょうか。展示物の方も実際に楽しんでいただけると幸いです。

最後にこの冊子を手にとって、最後まで目を通していただき、ありがとうございました.

平成 28 年 11 月 20 日
編集担当　矢口　喬脩

Lime Vol.54

平成 28 年 11 月 24 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/