目次

群とテンプレ

レジスタさん

1　とりあえず、書いてみただけ

群をご存知だろうか。集合とある閉じた演算の組で、結合律を満たし、単位元が存在し、全ての元に逆元が存在するものである。数学では群論という一大ジャンルが築かれるほど大人気である。自論だが、群論とテンプレートは相性がいい。何故なら、群論を扱えるフレームワークをテンプレートで作ってしまえば、扱う群が一般線形群であろうが置換群であろうが一緒くたのコードで扱えるからである。ふと思い立ち、私は C++ で有限群を扱えるライブラリを作り始めた。ここに、その片鱗の片鱗を記す。今はまだ、有限集合が群であるか否かの判定しかできない。無限群は少し厳しいモノがある。リファクタリングなどをしていないのはご愛想である。次回には、もう少しマシになっているであろう。

```
#include <iostream>
#include <string>
#include <vector>
#include <memory>

template <typename T> 抽象代数と言えば抽象、抽象と言えばテンプレート。//
class Group{ 名前思いつかなかったから「群」って名前にしちゃった☆//
        std::vector<std::unique_ptr<T>> Container; 集合の元を格納する。//
public:
        T (*calc)(T,T); 集合に定められた二項演算を関数ポインタとして保持する。ラムダ
            でも突っ込んでおこう。//

        T Item(int i){ 元に番号を番号で管理するが、ここでは番号から元を取得する。//
```

```
                return *(Container[i]);
        }

        int Order(){ 集合の元の数。群で言う位数ってやつ。//
                return Container.size();
        }

        bool Add(T x){ 集合に元を追加する。//
                for(int i=0;i<Order();i++){
                        if(Item(i)==x) return 1; カブってたらその旨を返す。
                            //
                }
                Container.resize(Order()+1);
                Container[Order()-1].reset(new T(x)); 元を追加。//
                return 0;
        }

        bool isGroup(){ この集合が群であるか否か。//結合則を満たすか？満たすなら
            半群。
//
                for (int i = 0; i < Order(); i++){
                        for (int j = 0; j < Order(); j++){
                                for (int k = 0; k < Order(); k++){
                                        if (calc(calc(Item(i), Item
                                            (j)), Item(k)) != calc(
                                            Item(i), calc(Item(j),
                                            Item(k)))) return false
                                            ;
                                }
                        }
                }

                std::cout << 半群です"\n";単位元が存在するか？存在すれば半群の中
                    でも、特にモノイド。

//
                bool isExId = false;
                T ID;
                for (int i = 0; i < Order(); i++){
                        bool isId = true;
                        for (int j = 0; j < Order(); j++){
                                if ((Item(j) != calc(Item(i), Item(
                                    j))) || (Item(j) != calc(Item(j
                                    ), Item(i)))) {
```

```
                                        isId = false;
                                        break;
                                }
                        }
                        if (isId == true){
                                isExId = true;
                                ID = Item(i);
                                break;
                        }
                }
                if (isExId == false) return false;
                std::cout << モノイドです"\n";
                std::cout << ID;
                std::cout << "\n";全ての元に逆元が存在するか？存在すればモノイドの中でも特に群。

//
                for (int i = 0; i < Order(); i++){
                        bool isExMi = false;
                        for (int j = 0; j < Order(); j++){
                                if ((ID == calc(Item(i), Item(j)))
                                    && (ID == calc(Item(j), Item(i)
                                    ))) {
                                        std::cout << Item(i);
                                        std::cout << の逆元は"";
                                        std::cout << Item(j);
                                        std::cout << だよっ！"\n";
                                        isExMi = true;
                                        break;
                                }
                        }
                        if (isExMi == false) {
                                std::cout << Item(i);
                                std::cout << に逆元は存在しないよっ！"\n";
                                return 0;
                        }
                }
                std::cout << 群です"\n";
                return true;
        }
};
```

2 おわりに

さて、私は今回、反省をしなければならない。まず、有限群を扱うライブラリを作るつもりが群判定のコードしか書けなかったことである。本来ならば、部分群を探したりもさせるつもりであった。しかし、何よりも反省しなければならないことは、締切当日になって書き始めて、まだ書き終わっていないにも関わらず、東方鈴奈庵の 3 巻を買いに世田谷から蒲田まで歩いて行き、さらにサイゼリアで夕飯を食べて帰宅して 11:57 分にこの TeX を書き始めたことである。ぶっちゃけソースコードを実際に動かしてるところも書きたかったが、時間も時間なので諦めることにする。この本を購入してくださった方には申し訳ないが、続きはいつか、http://griervels.blog70.fc2.com/ に記そうと思う。それか次のコミケで。ということで、次回から本気出す、と宣言して私の記事の締めくくりとさせてもらう。

Scala で作るファイル暗号器

ナナシ

1　はじめに

私が Scala を知ったのは、たまたまプログラミング系雑誌を読んでいた時に、「今、Scala が熱い！」みたいな特集が組まれているのを読んだことからでした。それから時々名前を見るようになり、少し興味がわいたことで Scala を勉強してみようかなと考えました。また、パスワード付きでファイル暗号するソフトはどんな動作をしているのか気になった事から、どうせならちょっと勉強した Scala でそれっぽいソフトを作ってみようと思ったのでこのソフトを作りました。ちなみに、Scala は JVM 上で動作しますが、Java にある嫌がらせかと思えるようなコードを書く必要がない言語です。Java をやろうとしてやたらと冗長な標準入力のコードを見て Java を投げ捨てた私にとってはありがたいですね。

2　暗号化方法

暗号には色々有りますが恐らく最も有名な暗号方式はシーザー暗号でしょう。これはアルファベットを指定文字数分シフトすることで元の文章を支離滅裂な文章に変換する方法です。この暗号方法は、シフトした文字数がわかってしまえば簡単に解読できます。しかし、実装はとてもとても楽です。なにせ指定文字数分ずらすだけですから。

シーザー暗号の解読が簡単理由は、シフトする文字数が固定的であるためだと考

えられます。このシフトする文字数をある区切り内で変動させれば、ある程度の暗号強度は確保されそうです。幸い、コンピュータの世界では文字も数字で表されます。つまり今回は、パスワードの文字列に含まれる文字 1 つ 1 つの文字コードを使ってシフトさせ暗号化する方法を実装していきます。

3 とりあえず文字列

ファイルの暗号化を行う前に、まずは入力された文字列を暗号化するテストコードを実装します。このテストコードがうまくいけば、そのままバイナリに適応させれば良いので、予備実験的な意味で進めていきます。そして出来たものがこちらです。

```
def SiftCode(str:String,pass:String,flag:Int) : String= {
    var count = -1
    val sift=if(flag==1){1}else{-1}
    val Re = for(i <- str)yield{
      count = (count+1)%pass.length()
    (i+(pass(count).toInt)*sift).toChar
    }
    return Re
  }
```

実際に文字を暗号化しているのは、SiftCode 関数です。この関数は、まず第 3 引数が 1 でプラス、それ以外がマイナスシフトとして sift 変数に代入しています。次に、for 式を使い、第 1 引数の str から 1 文字ずつ文字を取り出し、i に代入します。さらに第 2 引数の pass から 1 文字取り出し、i に加算して文字をシフトします。この動作を str の文字列が無くなるまで繰り返して、for 式の返り値として変数 Re に代入しています。

文字列とパスワードを入力し、動作を確認すると、パスワードによって暗号化後の文字列が違うことが確認できました。また逆方向にシフトさせると元の文字列に戻っているので、どうやらうまくいっているようです。では次はバイナリに対応さ

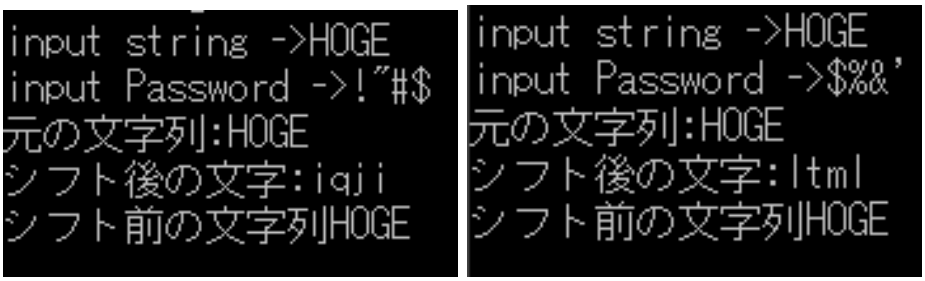


図 1　実行結果

せましょう

4　ファイルの暗号化

ファイルを暗号化するためにはまず、バイナリファイルを読み込む必要があります。残念ながら Scala の io ではバイナリを読み込むことができないので、Java.io の力を借ります。以下は SiftCode 関数をバイナリ用に書きなおした SiftBinary 関数を含む、ファイル暗号器の全体ソースです

```
import java.io.File
import java.io.FileInputStream
import java.io.FileOutputStream
object Main {
  def FiletoCaesar(InputName:String,SaveName:String,
                   PassWord:String,Flag:Int):Unit = {
val inputfile = new FileInputStream(InputName);//読み込みストリーム
val outputfile = new FileOutputStream(SaveName,true)//書き込みストリーム, 末尾追加型
val flag = if(Flag==0){1}else{-1}
var buf = new Array[Byte](10)//バッファ用意 10 バイトずつ読み込む
var count = 0
```

```
var i=0
  while( i != -1){
    val write = for(tmp <- buf if(i != -1))yield{
      (tmp.toInt+(PassWord(count).toInt)*flag).toByte
    }
      outputfile.write(write,0,i)
      i=inputfile.read(buf)
      count = (count+1)%PassWord.length()
  }
  outputfile.close()
  inputfile.close()
}
  def checkFile(FilePath:String):Option[Boolean] = {
    val file = new File(FilePath)
    if(file.exists()){Some(true)}
    else{
      println("File not found")
      None
    }
  }
def main(args:Array[String]) = {
    val FileName = readLine("input File Name ->")
    if(checkFile(FileName) != None){
    val SaveName = readLine("input Save Name ->")
    val PassWord = readLine("input PassWord ->")
    val Flag = readLine("Encryption:0 Decryption:1\n").toInt
     FiletoCaesar(FileName,SaveName,PassWord,Flag)
     println("End")
   }
```

```
  }
}
```

このコードでを見て、Java を知っている人は while 文の条件に違和感を感じると思います。java では while 文の条件を

while((i=inputfile.read(buf)) != -1) このように書くと思いますが、Scala では代入式では Unit 型を返すので条件が常に真になってしまうので、無限ループに陥ってしまいます。ブロック文を使えば解決できますが条件式が少し見づらくなるので、i を while 文の中で代入しています。

SiftBinary 関数は、SiftCode 関数より引数が 1 つ増えていますが、保存する名前を渡しているだけで、処理自体は文字列を暗号化した時と変わりません。

input.read が-1 が返すとファイルの末尾なので i が-1 になるまで while で 10 バイトずつ読み込んでいます。変数 Password から文字を 1 つ取り出し、buf に加えてシフトさせています。つまり 10 バイトずつシフトさせたバイナリは for 式の返り値として変数 write に代入されるので、write を追加書き込みしてバイナリファイルを作ります。FileOutputStream の第 2 引数を true にしておかないと上書きされてしまので注意が必要です。

checkFile 関数はファイルが存在しているかを返す関数です。入力されたファイルが無いとエラーが出るので一応ファイルの存在をチェックしています。3 つ目の入力で 0 が入力されると暗号化し、1 が入力された場合は復号化を試みます。簡単な暗号化ですが、ファイルの暗号器が完成しました

5　おわりに

今回は初めて Scala を使ってソフトを作りましたが、for や if で値を返すことができたり、まだまだ Scala の基本構文を勉強しただけにもかかわらず今まで使っていた言語とは違う所が多くあり、とても面白いです。このファイル暗号器は、今後も開発を続けて、ファイルだけでなくフォルダを圧縮暗号化などもできるように作りたいと思います。

twitter: @ 04nanashi09 2014 年 12 月 10 日

シューティングゲーム

ピーポー

1　はじめに

高校生のときに、Ｃ＋＋について少し学んでいたので、Ｃ＋＋を用いてゲームを作ってみようと思いました。また、ゲーム作成を易しくするＤＸライブラリを入れています。ゲーム自体作ったことがなかったので、プログラミング初心者でも、そこまで敷居の高くないという、シューティングゲームを作りました。

2　ゲームのポイント

ただシューティングゲームを作るだけだと、僕の技術力では、何の特徴もないサンプルプログラムとほとんど変わらないものしか作れないので、簡単なつくりでも、クリアするまで何度もプレイしたくなるようなものを作りました。

3　ルール説明

- このゲームのクリア条件は敵キャラにこちらの弾を５回当てることです。
- 相手の弾に１度でも被弾したらゲームオーバーです。
- 弾は一度に１発しか撃てません。
- リトライした回数が右上に表示されます。

4　操作方法

ゲームを立ち上げたら、スペースキーを押すとゲームが開始されます。十字キーの左右で自キャラを操作します。スペースキーで弾を撃ちます。

5　特徴のあるコード

何度も挑戦したくなるようなゲームに必要なものは程よい難易度だと考えたので、敵の弾の軌道に特に力を入れました。以下に敵の弾の軌道に使用したコードの一部を記載します。

```
goukei12 = -rikejox+px;
        goukei22 = goukei12/tyousei2;
eshotx2 -= goukei22;
```

goukei12 の中に自分の x 座標から敵の x 座標の値を引いたものを代入し、代入した値を tyousei2 という数値で割ったものを敵の弾の横の動きにしています。

6　作っての感想

前からゲームを作ってみたかったので、簡単なものでも作れてよかったです。今回は矢印キーで動かすシューティングゲームを作ったので、次回はマウスで動くいらいら棒のようなゲームを作りたいと思っています。

May log aim

カリカリ

1　はじめに

ゲーム作りが初めてということでイメージしながらプログラミングしやすい Unity を用いて作成しました。今回はあたり判定の感覚を掴んだりフラグを立ててゲームを進行させたりする練習になると思い、迷路ゲームを作りました。

2　ルール

- 矢印キーで移動、スペースキーでジャンプ。ただしジャンプ中は動けません。
- 赤い玉や白いプレートに当たったり、その時点では行けないようなところに無理やり入り込んだりするとゲームオーバーになります。
- 右上にある指示通りに行動することでゴールできるようになります。
- セーブ機能などはないのでゲームオーバーするたびに初めからやり直しとなります。

3　アルゴリズムについて

せっかく不恰好なりにも仕掛けを作ったので少々表面上だけの説明を加えさせていただきます。（ルールで述べた、敵である赤玉さんたちの説明は除く）

まずは設置障害物系です。このゲームは3 D に見えますがプレイヤーの動きは（ジャンプを除いて）2 D です。そのため要らない障害物を置いておく場所を作ることができました。そのおかげで条件を満たしたら待機場所に移動させたり担当場所に移動させて障害物となってもらうことで役割をはたしてもらうことができました。ボールなしでやるドッヂボール並につまらなかったとは思いますが職務を全うしてくれた障害物さんたちに感謝の言葉を贈りたいと思います。

他の説明を入れたいところですが、ソースコードの一部を抜き出す関係でこのページに入りきらないため改ページいたします。

次にジャンプ機能についてです。このゲーム唯一の３ D アクション部分なので意地でも実装しました。とは言ってもスペースキーを押したときに上昇、既定の高さまで上がったら下降させるというだけの簡単なプログラムになっております。実は下降の際に下がりすぎたりすることがあるのですが視点の角度のおかげで目立たないのでセーフとしています。また、自機が物体との干渉で２ D 以外の動きをしてしまったときもこのジャンプ機能の作用で対処しているように見えてくれました。鬼だけでやる鬼ごっこ並にカオスな処理を受け持ってくれたジャンプ機能さんに感謝の言葉を（ry

文章だけではアレなので、以下にジャンプのコードを載せます
～自機を制御するスクリプトから抜粋（「」内部は言葉で代用した部分です）～

```
if(「敵とのあたり判定」){
  「当たっていたらゲームオーバー判定に引っ掛ける」
}else if(flag == true || jump != 0){
 if(transform.position.z >= -3f && jump != 1){
    rigidbody.velocity = new Vector3 (0.0f,0.0f,-200.0f*Time.deltaTime);
    jump = 2;
    obj.SetActiveRecursively(true);
  }else if(transform.position.z >= -4f && transform.position.z < 0f){
    rigidbody.velocity = new Vector3 (0.0f,0.0f,200.0f*Time.deltaTime);
    jump = 1;
  }else{
    rigidbody.velocity = new Vector3 (0.0f,0.0f,0.0f);
    jump = 0;
  }
}else{
  transform.position += new Vector3
   (Input.GetAxisRaw ("Horizontal") * Time.deltaTime * 30,
     Input.GetAxisRaw ("Vertical") * Time.deltaTime * 30, 0.0f);
}
```

最後に、このゲームの最後の砦である門番さんについてです。消えてもらうときには設置系と同じく待機場所に移動してもらえれば上手くいったのですが、なんといっても自機との押し合いでのバグが多くて困りました。脇を抜かれて自機を通すし、押し合いに負けてゴールさせちゃったりと一番の問題児でした、故に作るうえで一番妥協点が多いギミックです。脇を抜くような（意地悪な）プレイヤーさんにはゲームオーバーという天罰を、押し合いに強くするためにある程度押されてしまったらスピードをあげて無理やり押し返す・・・とまあ捻りも何もない解決方法を駆使しました。世話のかかった門番さんには感謝の言葉は送りません。

4　おわりに

今回は完成品を作るということだけが目標だったため、計画も立てずに思いついたままに形にしていくスタイルをとりました。

次に作るものは事前にある程度の計画を立てたうえで作ったり、大好きなケモノをグラフィックに使ったりしたいなと考えています。

twitter: @krkr_faction　2014 年 12 月 6 日

2 つの弾幕アルゴリズム

mika

世田谷祭で作った弾幕シューティングの 2 つの自作弾幕のアルゴリズムを説明します。開発環境　visual studio 2013、C++、DX ライブラリ

――前提定義――――――――――

```
//弾幕に関する構造体
typedef struct{
//フラグ、種類、カウンタ、色
int flag, knd, cnt, col;
//ベース速度、ベーススピード
double base_angle[1], base_spd[1];
bullet_t bullet[4000];
}shot_t;
shot_t shot;

//弾に関する構造体
typedef struct{
//フラグ、種類、カウンタ、色、状態
int flag, knd, cnt, col, state;
//座標、角度、速度、スピード、
double x, y, angle,vx,vy, spd;
```

```
}bullet_t;
bullet_t bullet;
```

PI2=2 π、PI=π、FIELD_MAX_X=弾が動ける範囲 (X 軸方向)、ch.x(y)=自機の座標、shot.cnt は 1 フレームに 1 増加

-----弾幕 1 ----------

自機の周囲に出現する 10 個の弾と自機狙い依存の１０ way 弾が、ランダムな方向に直進する弾幕。

```
limtime = 10;
int i, k, t = shot.cnt%limtime,t2=shot.cnt;
double angle;
if (t < 10 && t2 % 30 == 0){
angle = arctan();//arctan は敵と自機の角度を出す関数

for (i = 0; i <10; i++){
bullet[k].state = 1;
bullet[k].col = 2;
bullet[k].x = ch.x + 110 * cos(i*PI2 / 10);
bullet[k].y = ch.y + 110 * sin(i*PI2 / 10);
bullet[k].knd = 1;
bullet[k].angle = arctan();
bullet[k].flag = 1;//弾のフラグ on
bullet[k].cnt = 0;
bullet[k].spd = 0;
angle = GetRand(PI2);
bullet[k].vx = cos(angle - PI / 8 * 4 + PI / 8  + PI / 16) * 3;
bullet[k].vy = sin(angle - PI / 8 * 4 + PI / 8  + PI / 16) * 3;
}}
```

```
if (t < 10 && t % 10 == 0){
angle = arctan();
for (i = 0; i < 10; i++){
bullet[k].x = .x;
bullet[k].y = .y;
bullet[k].angle = angle + PI2 / 10 * i;
bullet[k].flag = 1;
bullet[k].cnt = 0;
bullet[k].spd = 3;  }}

for (i = 0; i < 4000; i++){
if (bullet[i].flag>0){
if (bullet[i].cnt > 60){
int angle = GetRand(PI2);
bullet[i].x += bullet[i].vx;
bullet[i].y += bullet[i].vy;
bullet[i].angle = arctan(bullet[i].vy, bullet[i].vx); }}}
```

------弾幕２----------

出現座標 y は画面上部固定 x はランダムの、sin,cos 軌道を描く弾幕

```
limtime 120
int i, j, k, n, t = cnt%limtime, t2 = cnt;
if (t2 > 0 && t2 % 2 == 0){
for (n = 0; n < 1; n++){
bullet[k].flag = 1;
bullet[k].state = 0;
bullet[k].cnt = 0;
bullet[k].knd = 1;
```

```
bullet[k].col = 0;
bullet[k].angle = PI;
bullet[k].spd = 1;
bullet[k].x = 100;
bullet[k].y=10;
bullet[k].vy = 1; }}

if (t2 > 0 && t2 % 2 == 0){
for (n = 0; n < 1; n++){
bullet[k].flag = 1;
bullet[k].state = 1;
bullet[k].cnt = 0;
bullet[k].knd = 1;
bullet[k].col = 1;
bullet[k].angle = PI;
bullet[k].spd = 1;
bullet[k].x = 100;
bullet[k].y = 10;
bullet[k].vy = 1; }}

for (i = 0; i<4000; i++){
if (bullet[i].flag>0){
if (bullet[i].state == 0){
if (bullet[i].cnt == 0){
bullet[i].vx = GetRand(FIELD_MAX_X);
}
if (bullet[i].cnt<150)
bullet[i].vy += 0.036;
bullet[i].x = bullet[i].vx+100*sin(bullet[i].cnt*PI/90);
bullet[i].y+=bullet[i].vy;
```

```
}
if (bullet[i].state == 1){
if (bullet[i].cnt == 0){
bullet[i].vx = GetRand(FIELD_MAX_X);
}
if (bullet[i].cnt<150)
bullet[i].vy += 0.036;
bullet[i].x = bullet[i].vx - 100 * sin(bullet[i].cnt*PI / 90);
bullet[i].y += bullet[i].vy; }}}
```

CUI マインスイーパー

いちごまっちゃきゃんでぃ

1　はじめに

初めまして、いちごまっちゃきゃんでぃと申します。
ゲームを作るときにパソコンが壊れ、作業ができない事態に陥ってしまったためこんなゴミみたいな作品を皆様のお目を汚すこととなってしまい誠に申し訳ございません。ほとんど参考にならないどころか見ていて気持ち悪い、イライラするといった副作用が現れる可能性があるかと思いますので、それらが嫌な場合は読み飛ばしていただければと思います。

2　基本システム

初めに盤面の大きさと爆弾の数を指定させる。その後はゲームを開始する。爆弾の生成は初めの選択をした時とし、選択には開くのか、旗を立てるのかとその位置を指定させる。盤面は 2 次元配列を用いて管理し、3 つの配列を使っています。一つ目は爆弾の有無と選択の有無、二つ目は周りにいくつの爆弾があるかの管理（爆弾の数は生成時に取得)、三つ目は旗の管理となっています。入力には getchar() を用いて ASCII コードを解析することで想定外の入力に対して対策をしました。

次に選んだ増野周りに爆弾がなかったとき周りを開くためのソースコードを示します。

```
// 0 の時まわりを開くための関数
void open(Gamesystem* gs)
{
// 指定されたマスが 0 の時実行 (但しそのマスは盤面外ではない)
if (gs->target2[gs->i][gs->f] == 0 && gs->i != 0 && gs->f != 0
&& gs->target[gs->i][gs->f] != 1 )
{
gs->i--;
gs->f--;
defopen(gs);
gs->f++;
defopen(gs);
gs->f++;
defopen(gs);
gs->f--;
gs->f--;
gs->i++;
defopen(gs);
gs->f++;
gs->f++;
defopen(gs);
gs->f--;
gs->f--;
gs->i++;
defopen(gs);
gs->f++;
defopen(gs);
gs->f++;
defopen(gs);
gs->i--;
```

```
gs->f--;
}
}

// open 関数の補助
void defopen(Gamesystem* gs)
{
// そのマスの周りに爆弾がなく、かつそのマスが爆弾でないとき、
// さらにそのますが開かれていないとき、そのますに旗が立っていないときにの
み実行
if (gs->target2[gs->i][gs->f] == 0 && gs->target[gs->i][gs->f] == 0&&
gs->target3[gs->i - 1][gs->f - 1] == 0 && gs->target3[gs->i - 1][gs->f] == 0
&&gs->target3[gs->i - 1][gs->f + 1]==0 && gs->target3[gs->i][gs->f-1] == 0
&& gs->target3[gs->i][gs->f] == 0 && gs->target3[gs->i][gs->f + 1] == 0
&& gs->target3[gs->i+1][gs->f-1] == 0 && gs->target3[gs-> i + 1][gs->f] == 0
&& gs->target3[gs->i + 1][gs->f + 1] == 0)
{
gs->target[gs->i][gs->f] = 2;
gs->count++;
open(gs);
}
if (gs->target2[gs->i][gs->f] != 0 && gs->target[gs->i][gs->f] == 0
&& gs->target3[gs->i][gs->f] == 0)
{
gs->target[gs->i][gs->f] = 2;
gs->count++;
}
}
```

このようになっており、再起関数を用いて自分の上下左右に対して同じような処理を施し、周りを開いていきます。
盤面の周りは盤面生成時に一括して開いた状態に添えているのでこの関数の処理の対象外となっています。

3　おわりに

まだまだ力量が足りずあまり良いものが作れないとは思いますが、次回までにはもう少しまともなプログラムを皆様にお示しできればと思います。次回はパソコン壊れないといいなぁ等と思っているのですが、これがフラグってやつなんでしょうか？壊れたら今度こそなにも作れないで自滅してしまうような気がするので壊すわけにはいかないのですが、原因不明の画面が付かないなどの症状に関しては対策のしようもなくただただ焦るのみとなっております。twitter: @Ayano_Nagamori

2014 年 11 月 30 日

もじすくろーる

ssk

1　はじめに

Visual Studio 、DX ライブラリを使って 2D スクロールアクションをつくった。「もじすくろーる」というタイトルにしたのは、そのまま、文字がスクロールしているからである。

2　工夫した点

制作にあたって自分で工夫したと思えることは 2 つある。1 つ目は、スクロールだ。スクロールと言われて思い浮かべるのは、プレイヤーの移動に合わせて、背景が動く様子だろう。背景全体を「マップ」、現在表示しているマップを「画面」と呼ぶことにすると、スクロールとはマップが固定されていてその上を画面が移動しているように考えられる。今回自分が作ったスクロールは、画面を固定し、その下をマップが動いてスクロールするようにしている。

```
int scsp=1; //スクロールする速さ
int map_add; //プレイヤーが動いたときにスクロールするためのもの
int map_x; //マップ全体の x 座標
int screen_x; //画面上の x 座標
map_add-=scsp;
DrawGraph(map_j*cs+map_add,map_i*cs, ●●●.img,TRUE);
```

map_add=0 のとき、マップは x 座標が 0 のところから描画される。map_add がマイナスされていくことにより、マップの x 座標もマイナス方向にずれていく。こうしてマップを動かしていき、スクロールとなる。上記の map_i、map_j は、field という二次元配列のマップを描画するための変数。繰り返し文で使う、 i ,j と同じ働き。下記のように使う。cs は、#define で定義した定数。cs=32 である。ちなみに chip_size の略。●●●には、それぞれ番号に対応したマップチップが入る。

```
for(map_i=0;map_i<15;map_i++){
        for(map_j=0;map_j<100;map_j++){
                if(field[map_i][map_j]==1) Drawgraph〜;
                if(field[map_i][map_j]==2) Drawgraph〜;
                if(field[map_i][map_j]==3) Drawgraph〜;
                if(field[map_i][map_j]==4) Drawgraph〜;
                if(field[map_i][map_j]==5) Drawgraph〜;
                if(field[map_i][map_j]==6) Drawgraph〜;
        }
}
```

工夫した点の 2 つ目は、ジャンプである（工夫というよりも、他にいい方法が思いつかなかったので、これを使わざるを得なかった)。物理的に考えると、初速度や重力加速度が必要になってくると思うが、自分は物理が苦手なのでその方法は思いつかなかった。そこで使ったのは、sin 波だ。通常の y=sinx のグラフを考えると、0〜 π の間はジャンプの軌跡のように見える。これを利用して、きれいな曲線を描

くようなジャンプを作った。

```
//sin([波の幅]*(pi/2/[角度]*count)*[跳ぶ高さ]
jsp=(float)sin(3*(pi/2/120*count))*200;
```

jps は、jump_speed の略。pi は π（円周率）。(pi/2/120*count) の count を大きくしていくことにより、sin の値が徐々に大きくなり、count=120 で sin の値は最大となる。この時がジャンプにおける最高点になる。ここからさらに count を大きくしていくと、sin の値は 0 に近づく。0 に近づくということは、地面に落ちていくということである。

以上で自分が工夫したと思う点についての解説を終える。

3 おわりに

今回作ったゲームは、初めてということもあり、完成後 1 か月しか経っていない今でも、改善できるところが多く見られる。これを作った考え方と経験を活かして、次につなげたいと思う。

twitter: @

2014 年 12 月 10 日

CUI マインスイーパー

horaizon

1　はじめに

CUI マインスイーパーを製作したので工夫した点などについて書く。
使用言語:C
開発環境:Visual Studio 2013

2　ゲーム概要

マスには以下の構造体の二次元配列を使用した

```
typedef struct{
                int bomb;       //地雷の有無 (0/1)
                int open;       //タイルの開閉 (0/1)
                int flag;        //旗の有無 (0/1)
                int cnt;         //周囲にある地雷の数 (1~8)
                int place;      //タイルの位置 (1~9)
                int flag_cnt;   //タイルの周囲にある旗の数 (1~8)
} tile;
```

ゲームの流れは以下のようになる

1. プレイヤーに盤面の横幅と縦幅、地雷の数を入力してもらう
2. マスのステータスの初期化
3. プレイヤーに初回に開けるマスを指定してもらう
4. 盤面に爆弾を配置する
5. すべてのマスについて周囲にある地雷の数を数える
6. 終了条件の判定
7. 盤面の描画
8. プレイヤーの指示を受ける (マスを開ける、旗を立てる、etc)
9. 指示に応じてマスのステータスを調整
10. すべてのマスに対して周囲にある旗の数をカウントする
11. 終了条件の判定
12. ゲームが終了したなら結果を表示、そうでないなら 7 へ戻る

3　各工程の説明

次に、工夫した工程について解説する

4. 盤面に爆弾を配置する

この工程では 2 つの関数を作成し、使用した。

```
void initialize(void){
        srand(time(NULL));
}
```

```
void get_rand(int x,int y,int* tmp_x,int* tmp_y){
        *tmp_x=rand()%x+1;
        *tmp_y=rand()%y+1;
}
```

initialize 関数では get_rand 関数で使用する乱数の初期化を行わせた。
get_rand 関数では盤面の横幅、縦幅、爆弾を設置するマスの座標 (bomb_x,bomb_y) のアドレスを引数とした。
あるランダムな値を横幅または縦幅の最大値で割った余りに 1 を足すことで盤面内に収まるランダムな座標を取得した。

処理全体では以下のようになる。

```
initialize();
bomb_count=0;
while(bomb_count!=bomb){
        get_rand(x,y,&bomb_x,&bomb_y);
        if(tile[bomb_x][bomb_y].bomb==0
                  &&tile[bomb_x][bomb_y].open==0){
              tile[bomb_x][bomb_y].bomb=1;
              bomb_count++;
        }
 }
```

変数 bomb にはプレイヤーが入力した地雷の個数が格納されている。bomb_count は最初は 0 であり、地雷があるマスに配置されるごとに +1 されてゆく。この 2 つの変数の値が等しくなるまで地雷を配置し続けるようにした。一度地雷が配置されたマスは tile[i][j].bomb の値が 1 となるため次回からは if 文で弾かれる。また、初回にプレイヤーが開けたマスは tile[i][j].open の値が 1 となっており、こちらも if 文で弾かれるようにした。これにより最初に開けたマスが地雷だったという事態を避けられる。

5. 周囲にある地雷の個数を調べる

処理が長いのでどのように考えたかを記す。
横座標を i, 縦座標を j, 盤面の横幅の最大値を x, 縦幅の最大値を y とした場合ある特定のマスの位置は大きく 9 種類に分けられる。

i==1&&j==1:左上（tile[i][j].place=8 と定義）
i==x&&j==1:右上（tile[i][j].place=2 と定義）
i==1&&j==y:左下（tile[i][j].place=6 と定義）
i==x&&j==y:右下（tile[i][j].place=4 と定義）
j!=1&&j!=y&&i==1:左辺に接している（tile[i][j].place=3 と定義）
j!=1&&j!=y&&i==x:右辺に接している（tile[i][j].place=7 と定義）
i!=1&&i!=x&&j==1:上辺に接している（tile[i][j].place=1 と定義）
i!=1&&i!=x&&j==y:下辺に接している（tile[i][j].place=5 と定義）
上記のいずれでもないもの:周囲を 8 マスに囲まれいるマス（tile[i][j].place=9 と定義）

この分類によって得た tile[i][j].place の値を switch 文で場合分けした後にそれぞれマスに応じて爆弾があるかを調べるマスの数と座標を指定し、調べた。例えば、tile[i][j].place=1 であるならこのマスは上辺に接していて、周りを 5 マスに囲まれているから自分の右（i+1,j）、右下（i+1,j+1）、下（i,j+1）、左下（i-1,j+1）、左（i-1,j）に地雷があるかを調べればよく、tile[i][j].place=9 であるならばこのマスは周囲を８マスに囲まれているので周囲８マスに地雷があるかを調べる、といった具合である。地雷が見つかった場合はその度に自分の tile[i][j].cnt の値を +1 してゆく。これにより盤面が描画された時にそれぞれのマスが表示するべき数字が定まる。もし、tile[i][j] 自身が地雷である場合には tile[i][j].cnt の値は 9 となるようにした。tile[i][j].cnt が 0 である場合にはそのマスの周囲に地雷がないことを意味する。

8. プレイヤーの指示を受ける（マスを開ける、旗を立てる、降参する、etc）
scan ではプレイヤーに座標を指定してもらい（scan_x,scan_y）、tile[scan_x][scan_y] をスキャンする。スキャンは GUI マインスイーパーにおける左右同時クリックに相当する機能である。この処理は tile[scan_x][scan_y].cnt=tile[scan_x][scan_y].flag_cnt が成り立つときのみ実行される。すなわち、あるマスの周りにある地雷の数と周り

に立てられている旗の数は一致しなければならない。この処理が実行されると tile[scan_x][scan_y].place の値に応じて、tile[scan_x][scan_y] の周りにある、旗が立てられていない全てのマスの open を 1 にする。

10. すべてのマスに対してそのマスの周囲にある旗の数をカウントする

この処理ではまずすべてのマスの flag_cnt を 0 にする。その後で、工程 5. で行った処理に似た処理を行う。あるマスについて tile[i][j].flag=1、すなわち旗が立てられていた場合に tile[i][j].place を元に switch 文で分類してマスの位置に応じてそのマスの周囲のマスの flag_cnt を +1 してゆく。これをすべてのマスについて行うことで、すべてのマスについて特定のマスの周りに何個の旗が立てられているかを把握することができる。

補足. 空白マス (cnt=0) の処理

一般的なマインスイーパーでは空白のマスを開いた場合に数字のあるマスまで広がり続ける。このマインスイーパーでもこの挙動を実装したのでその仕組みを解説する。このマインスイーパーではマスが開かれた場合、そのマスの open の値は 1 となる。あるマスの open が 1 であり、かつ cnt が 0 である場合に、place の値に応じて周囲のマスの open の値を 1 とするようにした。この処理を全てのマスに対して左上から右下、右下から左上に向かって絨毯爆撃のように 10 往復ほどさせた。

4 おわりに

駆け足で仕上げたゲームだが製作中はとても楽しかった。通学中も電車の中で常にアルゴリズムを考えていた記憶がある。何も見ずに独力で作り上げることを目標としていたが結果的に友人からアドバイスをもらった（絨毯爆撃の部分）。次はこのマインスイーパーを GUI 化しようと思う。

2014 年 12 月 10 日

Transer(試作品)

every.sh

1　はじめに

みなさん、こんにちは。every.sh と申します。私は Unity でゲームを開発中です。今回は、私のコードスタイルを紹介したいと思います。

2　State と ScalarInfo と VectorInfo

まず、ゲームを開発するにあたり、私は ScalarInfo と VectorInfo という 2 つのクラスを作りました。しかし、これで開発しようとすると非常にコードなどが乱雑になったので新たに State というクラスを加え ScalarInfo、VectorInfo、State の三つのクラスをまず作ることにしました。以下にそれぞれについての説明をのせます。ちなみに言語は C#です。

2.1　State クラス

概要

Scalar クラスの「値」となるクラスです。

機能

■文字列との互換性　　比較を簡単にするため文字列に暗黙的にキャストされます。

2.2 Scalar クラス

概要

複数の State クラスと VectorInfo クラスを持ち、特定の「値」さすクラスです。

機能

■「値」の制限　　このクラスがさす「値」は保持する State クラスによって制限されます。

■VectorInfo を用いた「値」の変更　　このクラスがさす「値」は VectorInfo によって変更されます。それ以外では変更できません。

2.3 VectorInfo クラス

概要

このクラスは Scalar の「値」を変更するためのクラスです。

機能

■Scalar クラスの値の変更　　Scalar クラスの「値」を変更します。

■エフェクト　　Scalar クラスの「値」変更時に特定のメゾットを実行します。この機能と Scalar クラスの値の変更する機能は以下のようにして実装しました。

```
//VectorInfo 側
public bool todo()
{
        bool ans;
        this.Acsepter = true;
        this.Scalar.chenge();
        ans = this.effecter();
```

```
        this.Acsepter = false;
        return ans;
}

//ScalarInfo 側

public void chenge()
{
        foreach (VectorInfo vec  in this.Vectors)
        {
            if (vec.acsepter && vec.now == this.current)
            {
                this.current = vec.next;
                break;
            }
        }
}
```

注）effector は特定のデリゲイト。Scalar は対象となる Scalar クラス。courrent は ScalarInfo のさす「値」。

3　とのように使うのか

これらのクラスをどのように使うかというと、シーンを読み込んだり、プレイヤーの場所などに使えます。具体的には VectorInfo クラスを継承したクラスに以下のようなメゾットをたすことによって、シーン変更とうを実現します。

```
        public bool mv()
        {
            GameObject player;
            if (!load)
```

```
        {
            GameObject _camera;
            player = GameObject.FindGameObjectWithTag("Player");
            _camera = GameObject.FindGameObjectWithTag("MainCamera");
            GameObject.DontDestroyOnLoad(player);
            GameObject.DontDestroyOnLoad(_camera);
            Application.LoadLevel(this.scene);
            load = true;
        }
        if (load)
        {
            GameObject[] MvPoint;
            player = GameObject.FindGameObjectWithTag("Player");
            MvPoint = GameObject.FindGameObjectsWithTag("MovePoint");
            foreach (GameObject mp in MvPoint)
            {
                if (mp.name == this.inPoint)
                {
                    load = false;
                    mp.SendMessage("InRoom", player);
                }
            }

        }
        return load;
    }
```

注)LoadLevel メゾット直後、ゲームオブジェクトを探すと前のシーンのままなので load というフラグを使って見つかるまで探さしている。また、このメゾット、effecter に代入するほか、単体で UpDate メゾットでも実行させる。

4　おわりに

ここまで、付き合いいただきありがとうございました。これらを使って私はゲームを作っていくつもりです。

twitter: @ every_956

2014 年 11 月 30 日

LINQ のすゝめ

りぶ

1　はじめに

突然ですが、問題です。C#で int 型コレクションのうち、奇数である値のみの合計はどのようにして求めるでしょうか。ただしコレクションは IEnumerable を実装しているものとします。コレクションは以下の通りとします。

```
int[] collection = new[] { 1, 5, 4, 82, 35, 9, 38, 5 };
```

普通に求めるなら、恐らく以下のコードの通りになると思います。

```
int sum = 0;
foreach (var x in collection) {
    if (x % 2 == 1)
        sum += x;
}
```

C#はフリーフォーマットの言語なので「行」という概念は無いのですが、あえてこの例で言えば、この問題を解くのに変数 sum の宣言を含めて 5 行もかかっています。この例はまだ単純ですが、複雑なコレクションの処理をしているとパッと見て何をやっているのかはすぐには分かりません。

LINQ は、この複雑化してしまう問題を解決できるものであると僕は思っています。LINQ を使えば、C#や VB 上でのコレクション処理をとてもスマートに書くことができます (LINQ と聞いて from とか orderby とかのキーワードが思い浮か

んだ方はその先入観を捨ててください)。

なお、この記事で使用する言語は C# 3.0 以上、.NET Framework のバージョンは LINQ が使える 3.5 以上となっておりますのでご了承ください。

2 LINQ とは

LINQ とは、Language Integrated Query の略称で、データベースの問い合わせ処理をプログラミング言語に組み込むというものです。データベースのソースとして、`IEnumerable` を実装するコレクションを使うものを LINQ to Objects、SQL サーバーを使うものを LINQ to SQL、XML 文書を使うものを LINQ to XML と言ったりします。今回は LINQ to Objects に焦点を当てて紹介していきます。

LINQ to Objects は、言ってしまえば「コレクション処理でよく使うものが拡張メソッド郡になったライブラリ」です。拡張メソッドは `IEnumerable` の拡張メソッドとして定義されているので、それを実装している `List<T>` や `int[]`(配列) などからメソッドを呼び出すことで使用することができます。

3 LINQ を実際に使ってみる

では、これから実際に LINQ の拡張メソッドを使ってみましょう。`collection` として以下の配列 (配列は `IEnumerable` が実装されている) を使用することにします。

```
int[] collection = new[] { 1, 5, 4, 82, 35, 9, 38, 5 };
```

3.1 Where

コレクションをフィルタリングして `IEnumerable` を返します。引数にフィルタリングの条件を示すメソッドを渡します (基本的にラムダ式を使用します)。

```
// collection の要素のうち、偶数であるものだけを取得
var hoge = collection.Where(x => x % 2 == 0);
// hoge is IEnumerable<int>
// hoge: { 4, 82, 38 }
```

コレクションのフィルタリング処理が 1 行で書けてしまいました。とてもすっきりしています。しかし、これで驚くのはまだ早いです。Where メソッドは IEnumerable を返しているので、ここからさらに LINQ の拡張メソッドを呼び出すことができます。

```
// collection の要素のうち、偶数で 10 より大きいものだけを取得
var hoge = collection.Where(x => x % 2 == 0)
                     .Where(x => x > 10);
// hoge is IEnumerable<int>
// hoge: { 82, 38 }
```

このように、メソッド呼び出しに続けてメソッド呼び出しができます (メソッドチェーン)。このような書き方ができるので、コレクションの処理の流れが一目でわかります (パイプライン)。

3.2 Select

コレクションを変換して IEnumerable を返します。引数にコレクションの要素を変換するメソッドを渡します。

```
// collection の要素を、string のコレクションに変換
var hoge = collection.Select(x => x.ToString());
// hoge is IEnumerable<string>
// hoge: { "1", "5", "4", "82", "35", "9", "38", "5" }
```

Select メソッドは型の変換だけでなく、コレクション全体に算術演算を施すといった応用もできます。

```
// collection の要素を、すべて 2 乗する
var hoge = collection.Select(x => x * x);
// hoge is IEnumerable<int>
// hoge: { 1, 25, 16, 6724, 1225, 81, 1444, 25 }
```

Select メソッドも IEnumerable を返すので、さらに LINQ の拡張メソッドを呼び出すことができます (実例は省略します)。

3.3 Aggregate

コレクションを集計処理して、1 つの集計結果を返します。このメソッドは最初は少々とっつきにくいかも知れませんが、使いこなせるようになるととても強力です。

Aggregate メソッドはオーバーロードがいくつかあるのですが、ここでは一番よく使うであろうものを紹介します。

```
// collection の要素の合計を求める
var hoge = collection.Aggregate(0, (sum, x) => sum + x);
// hoge is int
// hoge: 179
```

「なんでこれで合計が求まるの？？」と思う人が大半だと思います。この Aggregate メソッドの実装を見てみましょう。

```
U Aggregate<T, U>(this IEnumerable<T> collection,
                  U seed,
                  Func<U, T, U> func) {
    U value = seed;
    foreach (T item in collection) {
        value = func(value, item);
    }
    return value;
}
```

これに、先ほどの合計を求める例のものを当てはめてみましょう。

```
int value = 0; // (U is int, seed = 0)
foreach (int item in collection) { // (T is int)
    value = value + item; // (func = (sum, x) => sum + x)
}
return value;
```

これはまさにコレクションの合計を求める処理になっています。Aggregate メ

ソッドを使えば、このような集計処理をある程度簡単に書くことができます。

Aggregate メソッドのもうひとつのオーバーロードとして、seed を与えないものがあります。この場合、コレクションの最初の要素が seed として使われ、その次の要素から集計処理が行われるようになります。これを使うと「文字列をカンマ区切りで列挙する」といったことなどができます。

```
// collection の要素を文字列化してカンマ区切りで出力
var hoge = collection.Select(x => x.ToString())
                     .Aggregate((str, x) => str + ", " + x);
// hoge is string
// hoge: "1, 5, 4, 82, 35, 9, 38, 5"
```

4 おわりに

LINQ の世界、いかがだったでしょうか。使いこなせればとても強力な武器になります。「はじめに」の問題を LINQ を使って解いてみましょう。

```
int sum = collection.Where(x => x % 2 == 1)
                    .Aggregate((s, x) => s + x);
```

紙面の都合上 2 行になっていますが、実際は 1 行に収まるほどの量です。とてもすっきりしています。LINQ を使うことで.NET プログラミングが更に楽しくなることでしょう。

(ここで紹介した LINQ の拡張メソッドたちはほんの一部です。他のメソッドも含めて詳しく知りたい方は MSDN などのドキュメントや紹介記事を検索してみるといいかもしれません。)

github: gssequence
twitter: g_s_sequence
2014 年 12 月 10 日

ブロック崩し

ウバタマ

1　はじめに

　プログラミング初心者ですが、何かゲームを作ってみたいなと思ってブロック崩しを作りました。作るにあたってウィンドウ生成とかが簡単にできるという DX ライブラリを使いました。

2　開発環境

C++
DX ライブラリ
Visual Studio 2013

3 あたり判定

作るにあたって一番苦労したボールとバーのあたり判定について書こうと思う。

```
//上から
if (y_bar - y_ball < (height_ball + height_bar) / 2
        && abs(x_ball - x_bar) < width_bar / 2 && dy_ball > 0
                && y_bar - y_ball > 0)
{
   dy_ball *= -1;
}
//右から
if (x_ball - x_bar < (width_bar + width_ball) / 2
        && abs(y_ball - y_bar) < height_bar / 2 && dx_ball < 0
                && x_ball - x_bar > 0)
{
   dx_ball *= -1;
}
//左から
if (x_bar - x_ball < (width_bar + width_ball) / 2
        && abs(y_ball - y_bar) < height_bar / 2 && dx_ball > 0
                && x_bar - x_ball > 0)
{
   dx_ball *= -1;
}
```

変数は図1のように定義している。`y_bar - y_ball < (height_ball + height_bar) / 2` という条件は、バーとボールが y 座標だけで見てボールとバーが接触しているかどうかを判定している。この条件を満たしていると接触していることになる。`abs(x_ball - x_bar) < width_bar / 2` は、x 座標だけで見てバーの横幅の範囲内にボールの中心があるかどうかを判定している。`dy_ball > 0` は移動量の正

負、つまりどういう方向に動いているかを判定している。これを満たすときは上からバーに向かって動いているということになる。この条件がないと、バーの中で反射し続けるということが起きてしまう。`y_bar - y_ball > 0` は、バーの下側で反射が起きないようにしている。

左右はこれと同じことを x 座標 y 座標でひっくり返してるだけです。

この条件を満たすと、該当する移動量の正負を反転させて反射させる。

4　作ってみた感想

苦労せずに作れたのでよかった。次は弾幕シューティングゲームとかを作ってみたい。

2014 年 12 月 10 日

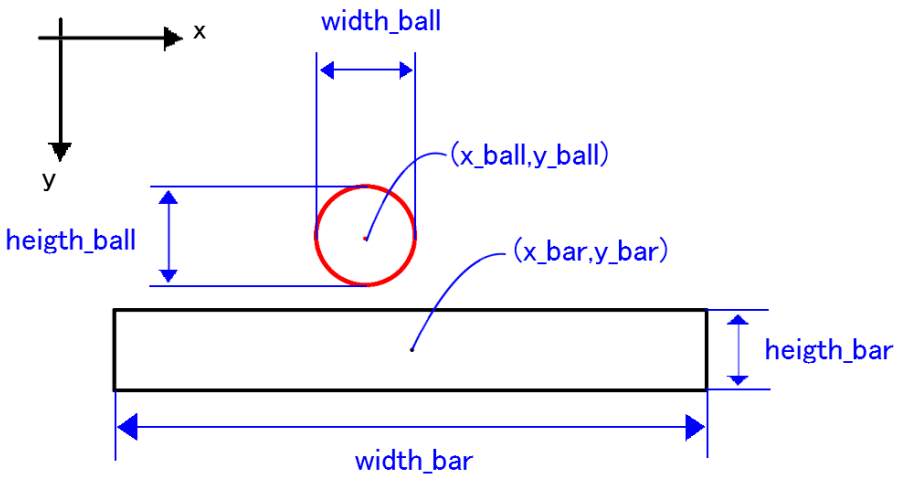

x
y
width_ball
(x_ball,y_ball)
heigth_ball
(x_bar,y_bar)
heigth_bar
width_bar

図 1　変数定義

Block Man

たち

1　はじめに

初めまして。今回初めて C++ を使用したゲーム作りに挑戦しました。自分はまだ上手できれいなプログラミングができず、できる人から見れば稚拙で無駄が多く、ゴミを散らかしたままの部屋のようなプログラミングになってしまったと思います。具体的に言えば、今回のプログラミングに構造体は登場しませんでした。ですので、今回は技術的な面ではなくアイデアや、知識不足ながらに工夫した部分について書いていきたいと思います。

2　開発環境

・Visual Studio 2013
・C++
・DX ライブラリ

3　能力変化

能力をブロックから受け取るというシステムがありますが、これはプレイヤーが特定の地面に接地しているときに Block 変数にそれぞれ対応した値が入れられ、その状態で X キーを押すと CBlock 変数に Block 変数と同じ値が入れられるという仕組みです。なのであるブロックに接地した後、ジャンプして空中にいる状態でも Block 変数はそのままで能力変化が可能な仕組みになっています。ジャンプ状態は Wjump 変数で判断しているので接地状態じゃないと能力変更ができないようにすることもできたのですが、そうすると操作性が悪くなるのでこのままにしました。

CBlock 変数の値によって、炎ブロックに触れてもゲームオーバーにならないだとか、二段ジャンプできるだとか、重さが増えたりだとかが決まっています。第六感の能力も、CBlock=6 でブロックの色が白から黒になったり黒から白になったり、といった具合で能力のように見せています。

4　マップチェンジ

マップチェンジも単純な構造です。

```
//6 to 11
if (Cmap == 6 && (int)(PlY - CHAR_SIZE * 0.5F) < 0){

PlX = (float)((int)(PlX - CHAR_SIZE * 0.5F));
PlY = 464.0;
Mtimer = 0;
Cmap = 11;
SBlock = Block;
SCBlock = CBlock;
SPlX = PlX;
SPlY = PlY;
Cont = 611;
}
//11 to 6
if (Cmap == 11 && (int)(PlY - CHAR_SIZE * 0.5F) >
480 && (int)(PlY - CHAR_SIZE * 0.5F) <= 640){

PlX = (float)((int)(PlX - CHAR_SIZE * 0.5F));
PlY = 16.0;
Mtimer = 0;
Cmap = 6;
SBlock = Block;
SCBlock = CBlock;
SPlX = PlX;
SPlY = PlY;
Cont = 116;
}
```

マップの番号はあらかじめこちらで決めています。PlX と PlY はプレイヤーの

X 座標と Y 座標、Mtimer は原因不明のバグ防止の応急処置、Cmap はマップ変更の変数 (後述します)、S が頭についている 4 つはオートセーブのための現状記憶用の変数、Cont はコンティニューする場所の変数 (後述します) です。このゲームはマップを生成するのに MapData(DXli ｂ) と DrawBox 関数（DXli ｂ）を使用しています。Cmap が変わると、DrawBox をすべての座標に適応し、プレイヤーの位置を変えることでマップが移動したように見せています。また、どのマップにいるか、どの範囲の X・Y 座標にいるかでマップ移動を出る場合と入る場合でそれぞれ一つ一つ作りました。ゴールブロックも同じ仕組みです。上の例だとマップ 6 からマップ 11 への移動と、マップ 11 からマップ 6 への移動を表しています。これで 1 セットです。今だったら構造体でも使えば少しは楽に作れるのか…とか考えるのですが結局この面倒くささは消えそうにありませんね。ちなみにマップ生成はこんな感じでやっています。

```
void Map1_Update(){
switch (Cmap){        //シーンによって処理を分岐
case 0:
MapData[0][0] = 1; MapData[0][1] = 0; MapData[0][2] = 0;
MapData[0][3] = 0; MapData[0][4] = 0; MapData[0][5] = 0;
MapData[0][6] = 0; MapData[0][7] = 0; MapData[0][8] = 0;
MapData[0][9] = 0; MapData[0][10] = 0; MapData[0][11] = 0;
MapData[0][12] = 0; MapData[0][13] = 0; MapData[0][14] = 0;
MapData[0][15] = 0; MapData[0][16] = 0; MapData[0][17] = 0;
MapData[0][18] = 0; MapData[0][19] = 0;

MapData[1][0] = 1; MapData[1][1] = 0; MapData[1][2] = 0;
        (省略)
```

これが MapData[14][19] まで続き、それで 1 セット (マップ一つ分) です。マップ数は全部で 16 なのでとんでもないことになってます。正直見せるのも恥ずかしいくらい原始的です…。現在は Excel を使っての効率化を勉強中です。

Cont はとても単純で、同じマップでもどこから入ったかで復活地点が区別されるようにしています。Cont の値によって、switch 文で呼び出す値が変わります。

5 移動する床

お仕置き部屋で登場した移動する床ですが、これは時間経過によって DrawBox であたり判定のある白いブロック (バニラブロックではない) を表示させたり消したりしているだけです。なので慣性等は考慮されていません。まあ考慮したらそれはそれでゲームの難易度があがりそうですが。

```
#define Timer72          (20)
(中略)
if (GetChipParam(afX, afY) == 112){
time7 = 1;
}
(中略)
if (time7 == 1){
Timer77++;
}
(中略)
if (Timer77 > Timer72 * 57 && Timer77 <= Timer72*62){
MapData[4][10] = 1;
}
if (Timer77 > Timer72 * 58 && Timer77 <= Timer72*63){
MapData[3][10] = 1;
}
if (Timer77 > Timer72 * 59 && Timer77 <= Timer72*64){
MapData[2][10] = 1;
}
```

かなり省略してますが、大方こんな感じです。大まかに説明すると、112 番のブロックに触れた瞬間にタイマー 77 が作動し、1 秒間につき 60 ずつ数字が増えていきます。見ればわかる通り、一つのブロックにつき Timer77 が 100(上の例で言うと (Timer72 = 20) × (62 − 57) = 100 です) 増える間だけ存在し、100 を超えると消滅します。それを繰り返してあたかも床が移動しているように見せています。消滅する床も仕組みは同じで、タイマーが一定の値に達したら 0 に戻るようにしてループさせています。

6　その他

最後に細かなことについてお話します。自分は、どれだけゲームを面白くできるか、楽しみ方に多様性を見いだせるかを意識して『特殊能力』という要素を取り入れ、そしてマップ設計をしました。また、ただゴールという結果がすべてになってしまうと面白くないと思い、スコア制も取り入れました。まあ結局は一番得点の高いゴールにどれだけコンティニューせずにどれだけ速くたどり着けるか、でスコアは決まってしまうんですけどね。例えば、特定のランクのゴールを目指すという制約をつけてスコアを競ったり、なんて遊び方もあると思います。とにかく遊んでいただく方に満足してもらえるようにできる限りの要素を詰め込みました。最終ゴールよりも到達しにくいゴールも中にはいくつかあるので、いろいろ試してもらいたいですね。そしてここだけの話ですが、実はある能力の状態でゴールすると得点が変わるゴールブロックが一つだけあります。また、総合得点がある値ぴったりだとクリアランクがスペシャルになるという要素も実はあったりします。まあそれだけなんですけど。作者の遊び心です。

7　終わりに

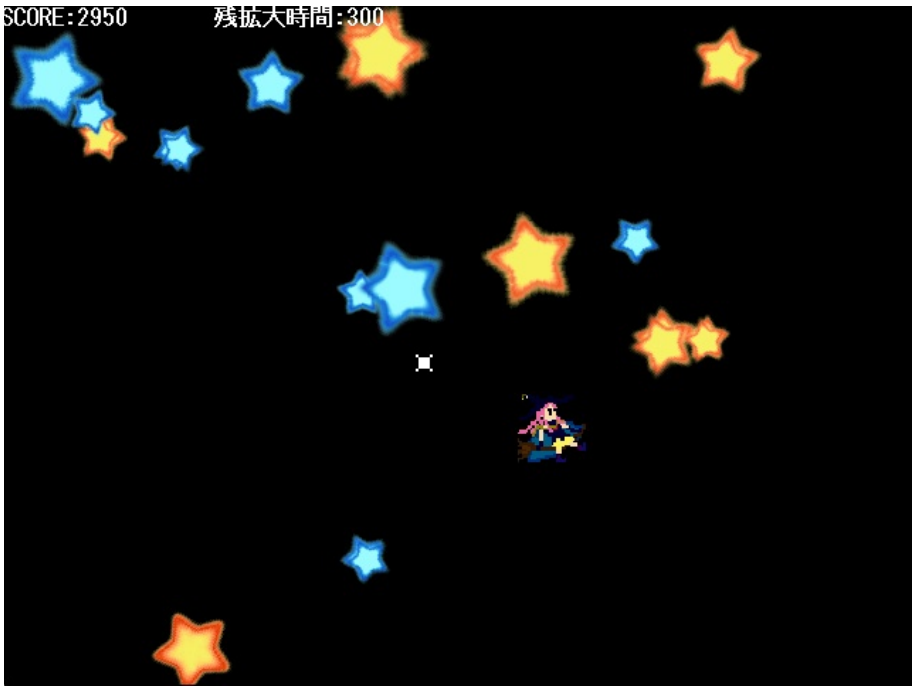
いかがだったでしょうか。自分で工夫した点といえばこれくらいですね。いろいろな遊び方があると思うので、何でもかんでもしちゃってください。とにかく楽しんでもらえれば幸いです。次回もゲームを作ろうと思いますので、今より少しは成長した技術をお見せできると思います。それでは、ありがとうございました。

twitter: @Tachiazul

2014 年 11 月 30 日

魔女の星集め

Shogun



1　ゲーム説明

空から降ってくる星をあつめてスコアを競うゲームです。

操作説明

矢印キー:移動

SHIFT:低速移動

A:SCORE500 を犠牲に弾の拡大時間を 150 延長する

X:押している間、弾を拡大する

Z:押している間、弾の速さを速くする

2　あたり判定について

このゲームをつくるにあたって一番苦労したのはあたり判定です。
作ってしまえば簡単だったことなんですが本当に高校 2 年生レベルの数学の知識をただ文字におこしただけでした。

簡単に説明すると物体の領域にほかの物体の領域がぶつかることがあたるということです。
例えば半径 1 の円があるとします。その円の中の領域を数式にすると

$$x^2 + y^2 <= 1 \tag{1}$$

になります。
これが今回の僕のプログラミングのあたり判定のすべての基本になっています。
今回、実際につかった僕のプログラミングのあたり判定を見てみましょう。

```
(playerx - (enemyx[1] - enemyxspeed[1] * enemyclock[1] / 3))*
(playerx - (enemyx[1] - enemyxspeed[1] * enemyclock[1] / 3)) +
 (playery - (enemyy[1] + enemyyspeed[1] * enemyclock[1] / 3))*
(playery - (enemyy[1] + enemyyspeed[1] * enemyclock[1] / 3))
<= (12 + 16)*(12 + 16)
```

プレイヤーの中心点 (playerx,playery), 半径 12
エネミーの中心点 ((enemyx[1]-enemyxspeed[1]*enemyclock[1]/3)

, (enemyy[1]+enemyyspeed[1]*enemyclock[1]/3)), 半径 16
２つの中心点間の距離が２つの半径の合計より短ければあたっているということになります。
このようにあたり判定はとても簡単なことだとわかります。
円のあたり判定が理解できれば点や線や楕円や長方形などのあたり判定も理解できてきます。

3　終わりに

今回、初めてのゲーム作成とあって、あまりたいしたものを作れなかったので、今回の経験をいかして次のものをより楽しめるようなものに改善していければと考えています。
最後に、ここまで読んでいただきありがとうございました。

ブロック歩き

がわ

1　はじめに

自分はまだプログラミングの熟練度がなく、C++ を使用して DX ライブラリを使ってもめちゃくちゃ、またはネット上で参考にしたものと同じようなアルゴリズムになってしまうので、本文章では本作品の操作説明、各処理で生じた問題の一部とその対処法について述べたいとおもいます。

2　作品概要

2D アクションゲームにブロック崩しの要素を加えたようなゲーム

3　開発環境

開発環境・・・Visual Studio Express 2013 for Desktop

開発言語・・・C++

ライブラリ・・・DX ライブラリ

4 操作説明

移動： 右・・・D 左・・・A ジャンプ・・・W or スペース
緑ブロックカーソルの移動・・・マウス
緑ブロック出し・・・左クリック
緑ブロックを消す・・・Q
メニュー画面に戻る・・・Esc

5 制作中に生じた問題点

目立った問題点はプレイキャラクター（以下 キャラ) と他ブロックとのあたり判定に関することが大半を占めていました。ここではその中の一部の問題点とその対処法について述べたいと思います。

・キャラの重力による問題点
以下は重力によるキャラの y 座標移動の処理です

```
        ・・省略・・

 if (jumpflag){

g = 0.0;

}
else{
y += (int)g;
g += 0.2;
}

        ・・省略・・
```

jumpflag=キャラが足場（ブロック）に着いているかどうかのフラグの bool 型の変数で、着いていれば true 着いていなければ false となるようにしています。
g=重力加速度のような double 型の変数です。これの詳しい説明は以下でし ます。
y=ここではキャラの画像の中心 y 座標の変数です。

他クラスなどで足場とキャラの画像が重なったとき、キャラが足場にぶつかって通れないようなあたり判定の処理をしています。キャラが足場に着いていないとき、y に int でキャストされた g を足していき、g の数値も大きくしていくことによりだんだん落ちるスピードが大きくなるような動きが可能になりました。足場に着いているとき同様な処理をすると g の値がとても大きくなってしまい、あたり判定の範囲を超えてキャラが足場をすり抜けてしまう（キャラの画像が足場の画像を飛び越える）ので、キャラが足場についているときつまり、jumpflag が true のときは g の値を 0 にしています。今述べたようにここでの問題点は”g の値が大きくなりすぎてあたり判定を超えてしまう”ことで、キャラが高いところから落ちるときにその問題が起きました。そこでその対策がいくつかあげられます。

1. 足場の画像を大きくする

2. g の値の変化を小さくする

まず
(a) について、これについてはゲーム中の一番下の灰色足場がそうです。ほかの足場を大きくしてしまうと画面が狭く感じてしまい、なるべく広い画面でキャラを動かしていきたかったので画面下外まで表示させても問題ない灰色の足場においてこの対策を講じました。

(b) について、これは単純に落ちるスピードを落としてキャラと足場のあたり判定を超えさせないようにするためです。これについてはゲームの難易度や見た目にも影響を与えてしまうので、自分なりに調節をかさねていき上の処理 (g+=0.2 の部分) のようになりました。

6 おわりに

初めてプログラミングを使ってのゲーム制作してみてプログラミングの勉強をするにはとても貢献したとおもいます。また、先輩方などから”キャラの落ちるスピードが速い”とご指摘をいただき, やはり、一人でゲームなどを制作していると自分で気づかない場所で改善点があるので、第三者の指摘やそれをもらえる環境はとても大切だと感じました。

BMS パーサをつくろう

h1dia

1　はじめに

Be-Music Source File(以下 BMS) をご存じでしょうか。簡単な命令が記述されたテキストファイルで、BMS と各楽器の音ごとに切り分けられた大量の音楽ファイル、および画像、映像データを読み込むと某 7 鍵風な音楽ゲームが出来るすごい代物です。今回は、C++ でこの BMS のパーサを作成したいと思います。

BMS は 1997 年にやねうらお氏が提唱したファイルフォーマットです。ですが、当時の BMS の機能が乏しかったために各 BMS プレイヤー開発者によって機能を拡張されています。そのため画一的な仕様を提示することは困難を極めますが、今日では LunaticRave2 という非常に強い人気を誇る BMS プレイヤーが存在しているために、このプレイヤーに合わせた BMS が大多数を占めています。

よって、今回製作するパーサの仕様はこの LunaticRave2 に合わせていきたいと思います。なお、BMS の仕様については hitkey 氏がまとめて下さっていますので、以下のページをご参照ください。

BMS command memo (JP) - http://hitkey.nekokan.dyndns.info/cmdsJP.htm

2　読み込み方法

BMS は一部の命令を除いて不連続に記述された命令から成り立っていますが、音楽を再生する以上実行する命令には順序が発生します。そのため、1 行ずつ読み

込むのではなく全ての行を読み込んだ後に適当な順序で命令を実行していく必要があります。最近では GUI エディタを使用している BMS が多いので命令がお行儀よくソートされた状態で並んでいますが、明確な仕様ではありません。

とりあえずコマンドとして有効な行を全部読み込みます:

```
std::ifstream ifs(bmspath);  //bmspath = hoge.bms
std::string tempstring;
std::vector<std::string> temp_array;

while (getline(ifs, tempstring)){
   if (tempstring.find_first_of("#") == 0){ //コマンド判定
      tempstring.erase(0, 1); //#を消す
      temp_array.push_back(tempstring);
   }}
```

次に、不連続な BMS の命令で唯一連続的な (そして問題児な)#RANDOM 文、#SWITCH 文、#IF 文を読み込み、temp_array から切り取ります。が、これらの命令の使用率は低いため、簡単にするためこれらの命令は考えないこととします。正確な BMS パーサを作るならば、これらの命令も解析する必要があります。

その後に音楽ファイル、画像ファイル、チャンネル文 (曲の譜面) の定義及び命令を各配列に振り分けていきます。振り分けたら命令文の解析をしましょう。音楽ファイル、画像ファイルは以下のように定義されます:

```
#WAVnn hoge0.wav
#BMPnn fuga0.png //mpg 等が指定されていることもある
ex)#WAV15 lead01.wav
```

nn には 01〜ZZ までの 36 進数 (！)2 桁の数が定義されます。この数は BMS 内でそのファイルのパスの ID として紐付けられて用いられるので大切に取っておきましょう。半角スペースをデリミタとして、ファイルパスが与えられます。用いられる拡張子及びそのフォーマットに明確な仕様はなく、パーサによる判断は難しいので素直に保持することにします。

これらを管理するために、次のような構造体を考えます:

```
typedef struct{
      std::string path;
      unsigned short int id;
}DATA;
```

命令文を解析していく過程で path にファイルパスを、id に先の 36 進数を 10 進数に変換して代入しましょう。

他にもヘッダ部には多種多様な定義が存在しますが、ほぼ同様の方法で読み込む事が出来るかと思われます。ほとんどがメジャーではない定義ですので、ここでは割愛することにします。

次はデータ部本体の読み込みを行います。データ部は以下のように定義されます:

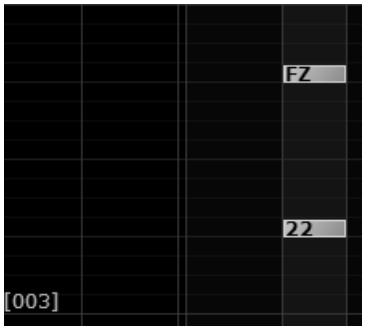
```
#xxxCH:0011223344XXZZ...
ex)#00311:002200FZ
```

xxx には小節数、CH にはチャンネル (後述) が与えられ、コロンをデリミタとしてその後に 36 進数 2 桁を 1 組としたファイルパスの ID が与えられます。なお、ID00 では何もしません。

チャンネルはこの命令の動作を指定する番号で、36 進数 2 桁で与えられます。例としてはノーツが降ってくる時間や、BGM、画像の表示位置などとさまざまです。パーサは一部の例外を除いて 10 進数として保持するだけで構いません。

コロンの後の ID は先の WAV 又は BMP の ID に対応しています。ID の長さを 2 で割ったものがその命令の解像度を指し、ID の場所は解像度中の位置 (0〜長さ-1) を指します。図では以下のようになります:

3 小節目に 2 つの ID が存在します。
22 は 2 つ目、FZ は 4 つ目の ID です。ID の長さ（解像度）は 4 です。
よって、22 は小節中の $\frac{1}{4}$ の位置に、
FZ は小節中の $\frac{3}{4}$ の位置に存在します。



これを読み込む構造体を考えます:

```
struct MAIN{
      unsigned short int measure; // 小節を示します。
      unsigned int step; // 解像度中の位置を示します。 ex.{1}/4
      unsigned int resolution; // 解像度を示します。ex.1/{4}
      unsigned short int id; // ID を示します。
      double num;

      bool operator<(const MAIN& next) const{
      return measure == next.measure
      ? (double)step / resolution <
          (double)next.step / next.resolution
      : measure < next.measure;
      }
};
```

構造体中のオペレータは std::sort を使用するために存在します。 num はチャンネル 02 の命令では ID ではなく小数が与えられるので、それを保持するために存在します。データを保持するために次のような構造体の配列を宣言するとよいでしょう。

```
std::vector<MAIN> main_data_array[575]; //(FZ)_36 = 575
```

チャンネル 02 では小数を num に保持し、それ以外では ID00 以外の命令を読み込み、その情報を保持します。

先に述べたように、これらは楽譜とほぼ同義にもかかわらず命令が不連続に宣言されているので、読み込んだ順にこれらのデータを使おうとすると大変なことになります。小節数を最優先に、その次に step を resolution で割った値を用いて std::sort 等で昇順にソートするとこれらのデータを扱いやすくなります。

BMS の簡単な解析方法は以上になります。

3　使用例

例えば、main_data_array[channel].size() からは読み込む ID の数が得られます。チャンネル 11〜19 は降ってくるノートを表すので、このチャンネルで読み込む ID の数と降ってくるノートの数は同じです。よって、これらの数を足すことでその BMS の合計ノート数を数えることができます。

DirectX 等と組み合わせれば、BMS プレイヤーや BMS エディターを作ることもできます。

4　おわりに

今回はとても簡単なパーサを考えましたが、基本的な考え方は同じですので機能を拡充することで十分使える物になるかと思います。今回の方法を用いて作られたパーサのソースコードについては github の h1dia/Bmsparser にて公開していますので、よろしければご覧ください。

駆け足となってしまい、詳細なアルゴリズムについては触れることが出来ませんでしたが、これを読んで下さった皆様が少しでも BMS の開発に興味を持っていただければ幸いです。

twitter: h1dia
github: h1dia
2014 年 12 月 7 日

初ゲーム作り

ichiro

1　はじめに

今回の取り組みのきっかけは自分でもゲームを作れるようになりたいと思い、その勉強としてこのゲームを作りました。ほとんど何も知らない状態でしたので本などを参考にして作成しました。開発環境は人気があり使っている人が多い Unity を使用しました。

2　ルール

- 目印にくっつくように物体を飛ばす。
- 次にすでに目印にくっついている物体をめがけて新たに物体を飛ばしてくっつけていく。
- なるべくたくさんくっつけることができるように考えながら物体を飛ばしていく。
- 一定数物体が落下すると、ゲームオーバーとなる。

3　おわりに

今回のゲーム製作を通じて、どのようにしてゲームは作られているのか、ゲーム作りに必要なことは何なのかということを学べて良かったと思っています。プログラミングや Unity についてもっとしっかり勉強していろいろなゲームを作れるようにしていきたいです。お読みいただきありがとうございました。

twitter: @

2014 年 12 月 10 日