平成 26 年 11 月 15 日
京都工芸繊維大学コンピュータ部

Lime 50

はじめに

こんにちは, コンピュータ部部長の出羽裕一です. 今年も無事に Lime が発行できて嬉しく思うと同時に, 製作に携わった部員, 手に取っていただいた方に厚く御礼申し上げます. この Lime もついに 50 号を迎えました. この Lime は, 部員の活動の一部を記したものです. 日々の活動の成果や興味に対する調べ物, 個人的趣味全開なものまで内容はさまざまです. 是非, 最後までお付き合いください.

平成 26 年 11 月 12 日
京都工芸繊維大学コンピュータ部部長　出羽 裕一

目 次

1 Hit&Blow とその発展ルールにおける最強戦略の導出に関する考察

情報工学課程 4 回生 永徳 泰明

1.1 はじめに

数当てゲーム Hit&Blow は MOO, Cow&Blow など様々な名で呼ばれ, 親しまれてきたゲームである. Hit&Blow の固定戦略においては, 相手がどのような戦略を使用しても勝率が 0.5 を超えることがないような戦略, すなわち最強戦略が存在することが田中 (1996)[1] によって示されている. 本稿では, 相手の戦略に応じて戦略を変化させる動的戦略において,Hit&Blow の最強戦略が存在するかどうか, および, 計算機により最強戦略を導出することが可能であるかについて論ずる.

1.2 Hit&Blow のルール

数当てゲーム Hit&Blow は MOO, Cow&Blow など様々な名で呼ばれる数当てゲームである. ルールを説明すると以下のようになる.

- 2 人のプレイヤーが互いに異なる 0〜9 までの数字からなる 3 桁の数を選択する.
- プレイヤーは先攻, 後攻に分かれ, 先攻のプレイヤーは相手の数を推理し 3 桁の数を宣言し質問する. 後攻のプレイヤーは選択した数と宣言された数を比べ, 桁と数字が一致する数字の数を Hit, 数字は含まれているが桁が一致しない数字の数を Blow として答える. 例えば選択した数が「123」, 宣言された数が「421」だった場合,2 は桁と数字が一致しており,1 は数字のみ一致するため, プレイヤーは「1Hit1Blow」と答える.
- これを交互に行い, 先に 3Hit を答えさせて数字を当てたプレイヤーの勝利となる. ただし, 同じ質問回数で双方が数字を当てた場合引き分けとするルールも存在する.

なお本稿では簡便のため,「1Hit1Blow」を「1H1B」のように表記する.

当てさせる数字の桁はルールにより異なり,4 桁のものも広く遊ばれている.

また,Hit&Blow からさらにルールを発展させたいくつかのゲームが考えだされている. フジテレビ放映の「Numer0n」という番組では,「Numer0n」と呼ばれる Hit&Blow の拡張ルールを採用したゲームが競技に用いられている. Numer0n で特徴的であるのは, 自分の手番で

宣言を行う前に使用できるアイテムが存在することである. これらは相手の数字を特定する情報を与える, あるいは自分の数字をゲーム中に変更することができ, プレーヤー間の駆け引き要素を大きくしている.

1.3　従来の研究

1.3.1　コンピュータと Hit&Blow

田中 (1996)[1] は MOO(Hit&Blow) をコンピュータにプレイさせる試みについて次のように語っている.

> ランダムな MOO 数を生成し, 人間の質問に対して bull 数, cow 数を答える MOO 出題プログラムは早くから作られていた. [2] に 1971 年当時のケンブリッジ大のシステムが紹介されている. プレイヤーの平均質問回数によって順位をつけ, ハイスコア表示をするもので, ユーザの間で大流行し, 乱数生成アルゴリズムを解析して, 次の出題を予測するものすら現れたとある. MOO 出題プログラムは, プログラミングの良い練習問題でもあり, [3] のようにプログラミングシンポジウムの課題プログラムになったこともあるし, [4] のように shell で書いた例もある.

コンピュータに Hit&Blow をプレイさせる試みは古くから行われており, またプログラミング初学者のよい課題ともなっていた. 当コンピュータ部においても, 偶然にも去年と今年で 2 人の部員が Hit&Blow をプログラミングの題材に取り上げていた.

1.3.2　従来のコンピュータプレイヤー

Hit&Blow において, 互いに異なる 0〜9 までの数字からなる数ををここでは HB 数と呼ぶことにする. 最初に候補となる HB 数は 720 個 (${}_{10}P_3$) 存在する. 質問を重ねるうちにこれらは絞り込まれていく. 最小の質問回数で HB 数の候補を絞りこむことができれば, それはよいコンピュータプレイヤーと呼べる. 候補の HB 数の集合に対して最良の質問となる HB 数をいかにして求めるかがコンピュータプレイヤーの鍵となる部分である.

コンピュータ部員が作成したコンピュータプレイヤーは, いずれもその時点の候補の HB 数の中からランダムに質問する HB 数を選択する, というものであった. こちらで同じ手法で計算するプログラムを作成して計算したところ,10 万回の試行中, 当てさせる数字が 3 桁の場合平均質問回数 5.24031 回,4 桁の場合 5.46737 回という結果を得た.

このコンピュータプレイヤーには改良の余地がある. 図 1.1 の例は計算途中のログである. 実際の解は 0123 で, 解の候補が $\{0123, 0231, 0235, 2503, 3012\}$ に絞りこまれたところからのログとなっている. この時点で 0231 または 0235 を質問すれば, 最悪でも 2 回の質問回数ですべての場合で数を当てることができる.(一つ目の質問がたとえ正解でなくとも, 候補を一通りに絞り込めるから) しかし, コンピュータプレイヤーは 3012 を質問し, 数を当てるまでに 3 回の質問を必要とした.

```
<hit 1,blow 2
0123 0231 0235 2503 3012
>call 3012
<hit 0,blow 4
0123 0231
>call 0231
<hit 1,blow 3
0123
>call 0123
<end
```

図 1.1: 候補からランダムに質問を選択するプレーヤーのログ

これはコンピュータプレイヤーが手の評価に質問の良し悪しを含めていないためである.

候補となる HB 数の集合を π,HB 数 T を選び質問した結果,4H になる数が t_0 個,3H1B になる数が t_1 個... というように分割した集合の個数を評価する関数 $f(T)$ を作成し, これを最小化するような T を質問に選ぶようなプログラムを作成することを考える. 先の例で言うと $T = 3012$ のとき,4B には $\{0123, 0231\}$ の 2 つの数が含まれている. $t_0, t_1, ...$ を用いて $f(3012) > f(0231)$ となるような関数, 例えば $f(T) = max(|t_i|)$ を用いれば,$f(3012) = 2, f(0231) = 1$ となり, プログラムは 0231 を優先して選択し質問する.

このように評価関数を用いる手法は実際に大きな成果を上げ,1987 年には工夫した $f(T)$ を用いたプログラムで 4 桁の数字に対して平均質問回数 5.22 回が達成されている.

さらに, 分割された集合に対して再帰的に探索をかけることでよりよい質問を求めることができる. 田中 (1996) は全探索を行い, 各局面での最良質問をテーブルとして保持することで, 平均質問回数 5.213 回を達成している.

これが一見最強戦略に見えるが, 実はそうではない. Hit&Blow では相手より早く数を当てなければ, 必ず負けてしまう. つまりそれは,7 手を 6 手にするよりも,4 手を 3 手にするほうが重要となることを意味する. 田中は固定戦略での最高勝率戦略も求めており, 最小質問戦略とは異なることを示した.

しかしこれもすべての戦略の中で最強であるとは言えない. これらの戦略は相手の手を全く考慮していないからである. 田中は相手の手の進みを考慮して戦略を変化させる動的戦略の可能性についても触れており,

> MOO の固定戦略の最小質問戦略と最強戦略を求めた. ただし, 最強戦略といっても固定戦略の中で最強なのであって, 相手の手の進み具合を見て手を変更する動的な戦略ならば, 最強戦略に対して勝ち越すことも可能であろう.

> ただし, こちらが正解までの質問を n 回と判断したかを相手が察知してしまうと, 自分の戦略で n 回で正解になる集合に解が含まれると判断して, より早く

> 数を当ててしまうかもしれない. 更に, それを見越して戦略を変えることも考えられる.

と述べている.

ただし, 通常のルールではこれらの駆け引きの要素は比較的小さいことが予想される. しかしながら,Hit&Blow の拡張ルールを採用した Numer0n では, 駆け引き要素が大きい.

本稿では,Hit&Blow およびその拡張ルールにおいて, 最強戦略である動的戦略をコンピュータ上で求めることが可能かどうかを検証する.

1.4　ゲーム理論における Hit&Blow の位置づけ

二人零和ゲームとは, ゲームを行うプレーヤーが二人でかつ, ゲーム終了時に双方が獲得する利得の合計が零となるゲームである. 勝利を 1, 敗北を-1 とすると,Hit&Blow は二人零和ゲームとみなせる.

有限ゲームとは, そのゲームにおける各プレーヤーの可能な手の組み合わせの総数が有限であるゲームのことである. Hit&Blow は有限ゲームではない. なぜなら一度質問した数をもう一度聞くことに対する制限が無いため, ルール上は無限に手が続くことも考えられるからである. ここで一度質問した数を二度と質問してはならない, という拡張ルールを導入すると, 有限性が保証される. これはゲームの通常のプレイに当たってはほぼ考慮する必要のない拡張である.

ナッシュ均衡は, ゲーム理論における非協力ゲームの解の一つである. プレーヤーの数と各プレーヤーの戦略の数が有限のゲームでは, 少なくとも 1 つのナッシュ均衡が存在することが知られている.[3] ナッシュ均衡とは, 片方がどう戦略を変更しようとも, より高い利得を得られないような均衡状態のことである. このナッシュ均衡に基づいた戦略を取る限り, 二人零和ゲームであれば相手プレイヤーはどのように戦略を変更しようとも, 同様にナッシュ均衡に基づいた戦略を取る以上に勝率を上げることはできない. これを最強戦略と呼ぶならば,Hit&Blow において, すべての戦略の中で最強となる戦略は存在するといえる.

Hit&Blow の拡張ルール Numer0n では複数回のゲームにわたって一回のみ使用できるアイテムが存在し, ゲームが一回で閉じていないため, 上の議論をそのまま適用することはできない. ただしアイテム自体は有限性を壊すものではないため, 例えば双方があるアイテムが使える状態でのナッシュ均衡戦略は存在する.

1.5　ナッシュ均衡戦略を利用した Hit&Blow 最強戦略の可能性

一般にナッシュ均衡は, あらゆる状態での行動の組み合わせ (純戦略) と, その行動を取った時得られる利得を組み合わせた行列 (利得行列) を用いた計画問題を解くことで求められる.

状態数が多くなるゲームの場合, 計算量が膨大になりこの解を求めることは困難になる. Hit&Blow の場合,3 桁でも状態数は計算不可能なほど多くなる. 現実的に 10 ターンゲームが続くことは考えづらいため,10 ターン以上かかったゲームを強制的に引き分けにするルー

ルを導入したとしても, 状態数 (情報集合の数) は大きく見積もって $720 * (720 * 9 * 720)^{10}$ 程度あり, 現実的に計算は不可能である.(CounterFactual Regret Minimization という手法を用いることで,10^{12} 程度の状態数ならば近似的に計算可能であるとされている.[4])

状態数が膨大で理想的な最適解を導けない場合, ゲームの状態を集約・抽象化することで, 近似的なナッシュ均衡戦略を導くことは可能となる. 例えば, 現在のターンの自分, 相手の候補数のみから戦略を決定するというようにすれば, 大幅に状態数を減らすことが可能となる. しかし, 状態数を減らすことはすなわちゲームにおいて考慮しない要素を増やすことでもあり, 戦略の厳密さが失われることを意味する.

どういった抽象化が最適か, 固定戦略と比べどれほど優れているのか, あるいは逆に劣ってしまうのかを把握するには, 実験による検証の必要があると考えている.

1.6 おわりに

本稿ではナッシュ均衡戦略を用いた Hit&Blow およびその拡張ルールにおいての最強戦略の計算可能性について考察を行った. これらの理論に基づきコンピュータプレイヤーを実装し, どのような挙動を示すかの観察を行うことが望ましかったが, 時間的余裕のなさにより考察のみにとどまった. これらは今後の課題となるだろう.

元々私は麻雀のコンピュータプレイヤーを作成する過程でゲーム理論を応用できないかと考え, 先行研究を漁っていたのだが,Hit&Blow にもこれらを適用できるのではとのひらめきでこれを書いた.(本当はコンピュータプレイヤーを作成して既存のコンピュータプレイヤーとの比較等をやる予定だったが, メインが既存研究の紹介になってしまった…)

計算可能性を考えると, 我々が日常プレイするほとんどのトイゲームではナッシュ均衡戦略を適用しようとするとモデルの抽象化が必須となる. しかし抽象化は戦略の厳密さ・正確さとのトレードオフである. これらの抽象化は個々のゲームに応じて様々なものが考えだされているが, 定石とされるようなものはなく, これからの研究が期待される.

参考文献

[1] 田中 哲朗: 数当てゲーム MOO の最小質問戦略と最強戦略, 第 3 回ゲームプログラミングシンポジウム (1996).

[2] 古居 敬大: 相手の抽象化による多人数ポーカーの戦略の決定, ゲームプログラミングワークショップ 2012 論文集 (2012).

[3] Wikipedia: ナッシュ均衡, http://ja.wikipedia.org/wiki/

[4] Martin Zinkevich, Michael Bowling,Michael Johanson,Carmelo Piccione: Regret Minimization in Games with Incomplete, http://martin.zinkevich.org/publications/regretpoker.pdf (2007).

2 メールとタスク管理アプリ

情報工学課程 3 回生 出羽 裕一

2.1 機能概要

日頃メールの確認と, タスクリストの整理を繰り返しているうちにこれらの作業を同時に行えるようなアプリケーションが欲しくなり, 実際に作成したのでこれについての報告をする. 目標とした機能は以下の通り.

- メールの送受信ができる
- 一斉送信ができる
- タスクリストの作成削除ができる

使用した言語は Java と呼ばれるものであり, 使いやすさを考えて GUI アプリケーションにしている.

2.2 プログラム概要

作成したクラスの名前と機能を以下に示す.

1. MailFrame:アプリケーションのメインフレームの設定とメインメソッドの実行
2. MainPanel:メイン画面のパネルの設定
3. TaskPanel:タスク管理画面のパネル設定とタスク管理処理
4. MailSend:メール送信画面のパネル設定と送信処理
5. MailRecv:メール受信画面のパネル設定と受信処理
6. AppUtil:ファイル操作と終了処理

またタスクの保存, メールアドレスの記録のために 2 つのテキストファイルを使用している. AppUtil クラスを除くクラスの詳細を示す.

2.2.1 MailFrame クラス

このクラスは今回のアプリケーションのメイン部分であり, パネル遷移メソッドの定義も行っている.
まずはコンストラクタ部分を示す.

```
public MailFrame(){
  this.add(mp);mp.setVisible(true);
  this.add(tp);tp.setVisible(false);
  this.add(ms);ms.setVisible(false);
  this.add(mr);mr.setVisible(false);
  this.setBounds(100, 100, 800, 800);
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e){
      tp.write_task_tofile(tp.model, util);
      util.EndFunc();
      System.exit(0);
       }
    });
}
```

setVisiable メソッドで最初に表示するパネルを指定してそのサイズの設定を行っている.addWindowListener ではウィンドウの閉じるボタンを押されたときのイベント処理を定義している.new WindowAdapter 以下の書き方は Anonymous Classes と呼ばれるものであり, この内部で宣言され使用されるものである.
次にパネル遷移メソッドを示す.

```
public void Change_Panel(JPanel panel, String p_name) {
  String name = panel.getName();
  if(name==PanelName[0])
      mp = (MainPanel)panel; mp.setVisible(false);
//以下同様の処理を他のパネルで行う
  if(p_name==PanelName[0])
      mp.setVisible(true);
//以下同様の処理を他のパネルで行う
}
```

このメソッドでは, 現在のパネルを非表示にして他のパネルを表示することでパネル間の移動を実現している.
以上の定義によってできたフレームにパネルを重ねて表示させ, アプリケーションを起動している.

図 2.1: 遷移前のメイン画像

図 2.2: 遷移後のタスク画面

2.2.2 MainPanel

実際の GUI パーツはこのパネルに貼り付けられており, 機能定義も行われる. 例としてタスク管理画面へ移動するボタンの定義部分を示す.

```
//タスク遷移ボタンの設定
  task_btn = new JButton("タスク画面へ移動");
  task_btn.setBounds(40, 500,200, 50);
  task_btn.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
      change_from_this(mf.PanelName[1]);
      }
    });
  this.add(task_btn);
```

ここではボタンの名前, サイズ, 位置, 機能の定義を行っている. 機能定義の部分では Anonymous Classes を用いて, 画面の移動を行っている. ここで使用される change_from_this メソッドは,Change_Panel メソッドの第 1 引数を this で固定したメソッドである.

2.2.3 TaskPanel

タスク処理の概要は以下の手順で行われる.

1. task.txt に保存されているタスクをリストに書き込む
2. タスクリストの要素をスクロールパネルに表示させる

3. タスクの完了と登録をユーザーから受け付ける
4. 現在スクロールパネルに登録されているタスクを temp ファイルに保存する
5. task.txt を temp ファイルで上書きする

手順 4 のファイルへの書き込みのタイミングは, メイン画面にある終了ボタンが押下されたときとウィンドウが閉じられたときである.

2.2.4 MailSend

メールの送信処理は以下の手順で行われる.

1. プロパティクラスでサーバ接続に必要な値を設定する
2. プロパティクラスの値を用いてセッションの確立する
3. セッションオブジェクトを用いてトランスポートオブジェクトを取得する
4. メッセージを作成して指定された相手にメールを送信する
5. トランスポートオブジェクトをもちいて SMTP で接続を行う
6. 終了時に接続をクローズする

次にメール処理を行っている部分のソースコードを示す. 接続先は Google が提供している Gmail のアカウントとしている.

```
//プロパティ設定
  mailprop = new Properties();
  mailprop.setProperty("mail.smtp.host", "smtp.gmail.com");
  mailprop.setProperty("mail.transport.protocol", "smtps");
  mailprop.setProperty("mail.smtp.port", "587");...
//セッション確立
  mail_sess = Session.getInstance(mailprop);
  //トランスポートオブジェクト取得
  trans = (SMTPTransport)mail_sess.getTransport("smtps");
//本文と副題の設定
  mail_mesg.setText(mail_body.getText());
  mail_mesg.setSubject(mail_sbct.getText());
//実際に接続する
  trans.connect("smtp.gmail.com","OO","XX");
//メッセージ送信
  trans.sendMessage(mail_mesg, mail_mesg.getAllRecipients());
```

この例では簡単のため, 一部のプロパティ設定と各メソッドで投げられるエラーのための try 文を省略し, 接続時のメールアカウントの情報を伏せ字にしている. 送信するアドレスはテキストファイル内に保存されるアドレスから選択し, 複数人に対する送信も可能である.
このクラスではメール処理以外に, 上記の操作のための GUI パーツを実装している.

2.2.5 MailRecv

メール受信処理は以下の手順で行われる.

1. プロパティクラスにサーバ接続に必要な値を設定する
2. プロパティクラスの値を用いてセッションの確立を行う
3. 受信ボックスに接続する
4. ボックスからメッセージを取得する
5. 終了時に受信ボックスとの接続をクローズする

次にメール処理を行っている部分のソースコードを示す. 条件は送信処理の場合と同様であり, プロパティ設定の部分は省略する.

```
//セッション確立
  mail_sess = Session.getInstance(mail_prop,new Authenticator() {
      protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication("OO","XX");
        }
      });
//Store オブジェクトを取得, これで受信ボックスにアクセスできる
  mail_store = mail_sess.getStore("imap");
  mail_store.connect();
//受信ボックスに接続
  mail_fldr = mail_store.getFolder("INBOX");
//mail_num 番めのメールを取得
  mail_mesg = mail_fldr.getMessage(mail_num);
//Object 型でメッセージを取得
  mail_obt = mail_mesg.getContent();
//メール本文を文字列型にしてテキストフィールドに表示
  mail_body.setText(mail_obt.toString());
//送信者の取得と表示
  from = mail_mesg.getFrom();
  mail_from.setText(((InternetAddress)from[0]).getAddress());
//副題の取得と表示
```

```
mail_sbjt.setText(mail_mesg.getSubject());
```

この例でも簡単のため try 文の省略とアカウント情報の部分を伏せ字にしている.getMessage メソッドの引数である mail_num の値を操作することで最新のメールから過去のメールも受信できる.

2.3 最後に

Java でアプリケーションを作成した経験が少ないため, いろいろと大変なことはありましたが, たくさんの協力を得ながら完成させることができました. しかし, メールの送受信で添付ファイルが取り扱えなかったり, タスク管理の選択肢が「登録」と「完了」の 2 通りだったり, まだまだ改善する余地はあります. また無計画に進めたため, オブジェクト指向言語らしいソースコードからも遠く感じます.
次はこの経験を生かして, さらに実用的なものを作成できればと思います.

参考文献

[1] 複数の JPanel で画面遷移 2 nowloading.blog.jp/archives/36741210.html

[2] JavaMail API documentation https://javamail.java.net/nonav/docs/api/overview-summary.html

[3] [Java] JavaMail で Gmail からのメール送信 https://hirooka.pro/?p=5058

[4] avaMail を使って Gmail に IMAP で接続するデモコード http://d.hatena.ne.jp/jbking/20080608/p1

3 テトリスのGUI化

情報工学課程 2 回生 前田 竜輝

3.1 はじめに

今回は去年作成したテトリスの GUI 版の制作を行いました. 開発環境は前回と同じ Microsoft の Visual Studio Express2012, 開発言語は C++で DX ライブラリを用いました. コーディングは主に「プログラミング入門サイト〜bituse〜」(URL は参考文献に掲載) を参考にしました. 基本的には前回作成したテトリスを元に作成したので変更または追加した部分に絞って書いていきますので, その他の部分は去年の Lime をご参照ください.

3.2 ウィンドウの表示

ウィンドウを表示する関数は次のとおりです.

```
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow )
{
    ChangeWindowMode(TRUE);
    SetMainWindowText( "Tetris" );
    SetGraphMode(FIELD_WIDTH*25+PIECE_WIDTH*25+10,FIELD_HEIGHT*25,16);
    if( DxLib_Init() == -1 )        // ＤＸライブラリ初期化処理
    {
      return -1 ;          // エラーが起きたら直ちに終了
    }
   //裏画面に描画
    SetDrawScreen(DX_SCREEN_BACK);

    //CONTROL クラスの動的確保
    CONTROL *control = new CONTROL;

    while(ScreenFlip()==0 && ProcessMessage()==0 && ClearDrawScreen()==0
     && GetHitKeyStateAll(key)==0){
```

```
//ゲームオーバーで true を返す
        if(control->All()){
            break;
        }
    }
    DxLib_End() ;                   // ＤＸライブラリ使用の終了処理

    //確保した領域の解放
    delete control;

    return 0 ;                  // ソフトの終了
}
```

ChangeWindowMode 関数は, 画面をウィンドウとして出すように設定するための関数. 続いて SetMainWindowText 関数でウィンドウのタイトルを設定し,SetGraphMode 関数でウィンドウのサイズを設定しています. また,SetDrawScreen 関数で描画先を裏画面にして ScreenFlip 関数（表示する画面と裏画面を入れ替える関数）を 1 回描画するごとに呼び出す事で, 前回問題となっていた画面のちらつきが起こらないようにしています. キー入力に関しては GetHitKeyStateAll 関数で各キーの押下状態を取得することで, 複数キーの同時入力にも対応できるようにしています.

3.3 各クラスの説明

今回は動かすブロックに関する関数や変数を piece クラス, ブロックを積んでいくフィールドに関する関数や変数を field クラスに分け, その 2 つのクラスを control クラスでまとめて呼び出すようにして,main 関数では control クラスのみを呼び出すようにしています. 各クラスではそれぞれ All 関数が定義されており,All 関数でそれぞれのクラス内の関数をまとめて呼び出し,main 関数で control クラスの All 関数を while 文で呼び出すことで各クラスの関数が毎回呼び出されるようにしています. ここからはその各クラスに分けて説明してゆきます.

3.3.1 piece クラス

piece クラスの定義は次の通り.

```
class PIECE{

public:
    int piece[PIECE_WIDTH][PIECE_HEIGHT];
```

```
    int next1[PIECE_WIDTH][PIECE_HEIGHT];
    int next2[PIECE_WIDTH][PIECE_HEIGHT];
    int next3[PIECE_WIDTH][PIECE_HEIGHT];
    int hold[PIECE_WIDTH][PIECE_HEIGHT];
    //x 座標,y 座標
    int x,y;
    //落下予測地点の x 座標,y 座標
    int shadex,shadey;
private:
    bool firstflag;

private:
    void Draw();
    void Destroy();
    void CreatePiece();
    void NextPiece();

public:
    PIECE();
    ~PIECE();
    void All(bool);
    void HoldPiece();
    int GetPieceTop();
    int GetPieceBottom();
    int GetShadeBotttom();
    int GetPieceLeft();
    int GetPieceRight();
};
```

ブロックは前回と同様 4 × 4 の 2 次元配列で定義しています.HoldPiece 関数は control クラスから直接呼び出すようにしています.GetPiece・・・関数は左右と下の壁の当たり判定の際に呼び出されます.GetShadeBottom 関数は落下予測地点の表示の際に呼び出されます.piece クラスの All 関数は次の通り.

```
void PIECE::All(bool dropflag){
    if(dropflag){
        NextPiece();
        CreatePiece();
    }
    Draw();
}
```

NextPiece 関数と CreatePiece 関数は dropflag が true になった時（ブロックが落ちきった時）のみ呼び出されるようになっており Draw 関数は毎回呼び出されるようになっています.NextPiece 関数と CreatePiece 関数はほぼ前回と同じなので説明を省きます.Draw 関数は次の通り.

```
void PIECE::Draw(){
    SetDrawBlendMode(DX_BLENDMODE_ALPHA,100);
    for(int i=0;i<PIECE_HEIGHT;i++){
        for(int j=0;j<PIECE_WIDTH;j++){
            if(piece[j][i])
            DrawGraph((shadex+j)*width,(shadey+i)*height,
                gh[piece[j][i]],FALSE);
        }
    }
    SetDrawBlendMode(DX_BLENDMODE_NOBLEND,0);
    for(int i=0;i<PIECE_HEIGHT;i++){
        for(int j=0;j<PIECE_WIDTH;j++){
            if(piece[j][i])
            DrawGraph((x+j)*width,(y+i)*height,gh[piece[j][i]],FALSE);
        }
    }
    DrawString(FIELD_WIDTH*25+10,0,"NEXTPIECE",GetColor(255,255,255));

    for(int i=0;i<PIECE_HEIGHT;i++){
        for(int j=0;j<PIECE_WIDTH;j++){
            DrawGraph((FIELD_WIDTH+j)*width+10,i*height+15,
                gh[next1[j][i]],FALSE);
        }
    }
    for(int i=0;i<PIECE_HEIGHT;i++){
        for(int j=0;j<PIECE_WIDTH;j++){
            DrawGraph((FIELD_WIDTH+j)*width+10,
                (PIECE_HEIGHT+i)*height+25,gh[next2[j][i]],FALSE);
        }
    }
    for(int i=0;i<PIECE_HEIGHT;i++){
        for(int j=0;j<PIECE_WIDTH;j++){
            DrawGraph((FIELD_WIDTH+j)*width+10,
                (PIECE_HEIGHT*2+i)*height+35,gh[next3[j][i]],FALSE);
        }
```

```
    }
    DrawString(FIELD_WIDTH*25+10,PIECE_HEIGHT*3*height+45,
        "HOLD",GetColor(255,255,255));
    for(int i=0;i<PIECE_HEIGHT;i++){
        for(int j=0;j<PIECE_WIDTH;j++){
            DrawGraph((FIELD_WIDTH+j)*width+10,
                (PIECE_HEIGHT*3+i)*height+60,gh[hold[j][i]],FALSE);
        }
    }

}
```

まず SetBlendMode 関数は加増を描画する際の透過度を設定する関数で, 第 2 引数に 0〜255 の間で数字を設定し, 数字が大きいほど透過度が高くなります. まず, 画像を透過させて描画できるように設定して落下予測地点の描画を行い, 透過度を 0 に設定しなおして, 残りのブロックや次に出てくるブロックの描画をしています.

3.3.2 field クラス

field クラスの定義は次の通り.

```
class FIELD{
private:
    int field[FIELD_WIDTH][FIELD_HEIGHT];

private:
    //画像の横, 縦幅
    int width,height;

    //ゲームーオーバーフラグ
    bool endflag;


private:
    void Draw();
    int DeleteLine();
    void ShiftLine(int);
    void Destroy();

public:
    FIELD();
```

```
    ~FIELD();
    int GetField(int,int);
    void WriteField(int,int,int);
    void All(bool);
};
```

フィールドも前回と同様 10 × 20 の 2 次元配列で定義しています.GetField 関数,WriteField 関数はフィールドの 2 次元配列を control クラスで読み書きする際に呼び出す関数です. 特に難しいことはしていませんが, 一応ソースコードを示しておきます.

```
int FIELD::GetField(int i,int j){
    return field[i][j];
}
void FIELD::WriteField(int i,int j,int a){
    field[i][j]=a;
}
```

field クラスの All 関数は次の通り.

```
void FIELD::All(bool dropflag)
{
    if(dropflag){
        ShiftLine(DeleteLine());
    }
    Draw();
}
```

こちらも piece クラスと同様に DeleteLine 関数と ShiftLine 関数は dropflag が true になった時（ブロックが落ちきった時）のみ呼び出されるようになっており Draw 関数は毎回呼び出されるようになっています.DeleteLine 関数と ShiftLine 関数もほぼ前回と同じなので説明を省きます.Draw 関数は次の通り.

```
void FIELD::Draw(){
    GetGraphSize(gh[0],&width,&height);
    for(int i=0;i<FIELD_HEIGHT;i++){
        for(int j=0;j<FIELD_WIDTH;j++){
            DrawGraph(j*width,i*height,gh[field[j][i]],FALSE);
        }
    }
}
```

こちらは右上から順番にフィールドの 2 次元配列を描写していってるだけなので非常に簡素です.

3.3.3 control クラス

control クラスの定義は次の通り.

```
class CONTROL{
private:
    //フィールドクラスのポインタ
    FIELD *field;
    //ピースクラスのポインタ
    PIECE *piece;
    //ピースの座標
    int x,y;
    //ブロックの画像の幅と高さ
    int width,height;
    int leftcount,rightcount,upcount,downcount,spacecount,hcount;
    //ゲームーオーバーフラグ
    bool endflag;
    bool dropflag;
    DWORD sleep;
    DWORD droptime;
private:
    void MoveRight();
    void MoveLeft();
    bool MoveDown();
    void TurnPiece();
    void PieceToField();
    void PieceShade();
public:
    CONTROL();
    ~CONTROL();
    bool All();
};
```

MoveRight 関数〜PieceShade 関数までは前回とほぼ同様.control クラスの All 関数は次のとおり.

```
bool CONTROL::All(){
if(dropflag) droptime=GetTickCount();
    dropflag=false;
    if(key[KEY_INPUT_LEFT] == 1){
        if((++leftcount)>=LRCOUNT){
```

```
            //左に移動.
            MoveLeft();
            leftcount=0;
        }
    }
    else leftcount=0;
    if(key[KEY_INPUT_RIGHT] == 1){
        if((++rightcount)>=LRCOUNT){
            //右に移動.
            MoveRight();
            rightcount=0;
        }
    }
    else rightcount=0;
    if(key[KEY_INPUT_DOWN] == 1){
        if((++downcount)>=COUNT){
            //下に移動.
            if(!MoveDown()){
                PieceToField();
                dropflag=true;
            }
            downcount=0;
        }
    }
    else downcount=0;
    if(key[KEY_INPUT_UP] == 1){
        if((++upcount)>=COUNT){
            //ハードドロップ
            while(MoveDown());
            PieceToField();
            dropflag=true;
            upcount=0;
        }
    }
    else upcount=0;
    if(key[KEY_INPUT_SPACE] == 1){
        if((++spacecount)>=COUNT){
            //回転
            TurnPiece();
```

```
            spacecount=0;
        }
    }
    else spacecount=0;
    if(key[KEY_INPUT_H] == 1){
        if((++hcount)>=COUNT){
            piece->HoldPiece();
            hcount=0;
        }
    }
    else hcount=0;
    if(key[KEY_INPUT_ESCAPE] == 1){
        endflag=true;
    }
    if(GetTickCount()-droptime>dropspan){
        if(!MoveDown()){
            PieceToField();
            dropflag=true;
        }
        droptime=GetTickCount();
    }
    PieceShade();
    skip:
    field->All(dropflag);
    piece->All(dropflag);
    for(int i=0;i<PIECE_HEIGHT;i++){
        for(int j=0;j<PIECE_WIDTH;j++){
                if(piece->piece[j][i]&&field->GetField(piece->x+j,piece->y+i))
                    endflag=true;
        }
    }
    return endflag;
}
```

前述のとおりキー入力に関しては GetHitKeyStateAll 関数を用いています.GetHitKeyStateAll 関数は全キーの押下状態を確認し, キーが押下状態にある場合それに応じた配列の要素に 1 を代入します.All 関数内ではその配列を確認し, キー入力に応じた関数を呼び出しています. 例えば, 右矢印キーが押下状態にある時,rightcount に 1 が足されます. そして,rightcount が COUNT（7 と定義している）以上になると MoveRight 関数が呼び出されます. 押下状態でなくなると rihgtcount は初期化（0 になる）されます. このような形にして

いるのは,count なしで関数を呼び出すようにすると, 少しの入力で何度も MoveRight 関数が呼び出されてしまい, 正確な操作ができなくなってしまうためです. ゲームオーバーの判定は前回と同様にフィールドに新しいブロックが出現した際にフィールド上にあるブロックと重なってしまった場合となっています. 条件に当てはまると endflag が true になり,endflag が main 関数に返され,while 文から抜け出します.

3.4 おわりに

半分程度は前回の流用であまり書くことがありませんでしたが, 一応の完成には至りました.C++のコーティングは初めてなので class ファイルなどの使い方がこれでよいのかはいまいちわかりませんが, 満足の行く出来になったと思います. 最初は WinAPI で作るか Unity などで 3D 化しようかとも思いましたが, 勉強が間に合いませんでした. しかし, さすがに 3 年連続テトリスというわけにも行かないので, 来年はなにか違うものを開発したいと思います.

参考文献

[1] プログラミング入門サイト〜bituse〜 http://bituse.info/

[2] C++クラス http://www.asahi-net.or.jp/~wv7y-kmr/memo/old/cpp_cls.html

4 Hit&Blow

情報工学課程 1 回生 家原 瞭

4.1 はじめに

なにか今自分にできる技術で作れる面白いものはないかなと思い, これを作りました. Lime の中身を書くときに参考にしようと思い過去の Lime を見た時に, 去年のものにそっくりなものがあるというのに気付きすごい焦っていますが, ソースの中身は一から自分で考えたものなのでたまたま重なったということで許して欲しいです. せめて昨年や一昨年のものくらいは確認しておくべきだったと反省してます.

4.2 ゲーム説明

よくある Hit&Blow です. 3 か 4 桁の隠された数字を当てるゲームです. すべての桁の数は違っていて, 0 から始まることも有ります. 適当な予想を入力するとその予想がどれだけ隠された数字に近いかを教えてくれます.

- 予想と桁も数字も同じ時は Hit
- 予想と数字は同じだけど桁が違う時は Blow
- Hit と Blow の数が表示される

例えば, 隠された数字が２４６３で, 予想が３４５６なら, 2 桁目の 4 が Hit で 3 と 6 が Blow なので, 1 Hit 2 Blow になります.

4.3 モード説明

4.3.1 normal mode

相手の隠された数字を当てるモードです. まず桁数を選び, 次に難易度を選びます. 難易度によって予想の回数が限定されます.（桁数 easy:×４　normal:×３　hard:×２　very hard:×１）hard 以上は運が必要になってきます. 規定の回数内に隠された数字を当てることができれば, クリアになります.

4.3.2 watch cpu mode

cpu がこちらの数字を当ててくるのを鑑賞するだけのモードです. おもしろさはあんまりないですが,cpu が頑張るところを眺めてみてください. 使い方は, 桁数選択の前に自分のなかで数字を決めて, メモする. あとは cpu が勝手に数字を言ってくれるので, それに合わせて Hit と Blow の数字を入力していくと cpu がこちらの数字を当ててくれます.

4.3.3 endress mode

normal mode のクリアの規定回数がないモードです. 当てるまでチャレンジできるので, 練習用に作りました.

4.3.4 vs cpu mode

cpu と対戦できるモードです. 相手は内部で隠された数字を選ぶので, こちらは最初の予想を入れる前に数字を決めてメモしてください. こちらの予想と, 相手の予想が交互にでるので, 予想する数字と HitBlow の数を交互に入力してください. cpu が当てるよりさきに cpu の数字を当てると勝ちになります.

4.4 ソースコード

いろいろ書いてソースコードは長いので, 一部分だけ紹介したいと思います.

4.4.1 隠された数を決める

```
void ranset(int* num,int figure){
        int tmp,count=0,flug,i;
        while(count<figure){
                flug=0;
                tmp=rand()%10;
                for(i=0;i<count;i++){
                        if(tmp==num[i]){
                                flug=1;
                        }
                }
                if(flug==0){
                num[count]=tmp;
                count++;
                }
```

```
        }
}
```

桁数 figure と int 型配列 num のアドレスを受け取り, 指定桁数分の重複のないランダムな数列を用意する関数です. 重複を抜く方法は, 重複したらやり直しを繰り返してるだけです.

4.4.2 hit と blow を数える

```
void hitblow(int* hit,int* blow,const int* num1,const int* num2,int figure){
        int i,j;
        *hit=0,*blow=0;
        for(i=0;i<figure;i++){
                for(j=0;j<figure;j++){
                        if(num1[i]==num2[j]){
                                if(i==j){
                                        (*hit)++;
                                }else{
                                        (*blow)++;
                                }
                        }
                }
        }
}
```

なんかいろいろ受け取ってややこしいですが, 要約すると,num1 と num2 に入っている数列を桁数 figure 分比べて hit と blow の指すアドレスにそれぞれの値を入れてるだけです.

4.4.3 cpu の内部構造

```
int main(void){
        int mode=0,num1[4],num2[4],num3[4],figure,i,j,dif,max_cnt,cnt;
        int hit,blow,flug,cpu_cnt,tmp_hit,tmp_blow,roop_cnt;
        char tmp[5];
        typedef struct{
                int dat[4];
                int dat_hit;
                int dat_blow;
        } data;
        data list[100];
        srand((unsigned)time(NULL)); //乱数のシードを与える
```

```
        printf("桁数を入力してください (3 or 4):");
        scanf("%d",&figure);
        cpu_cnt=0;roop_cnt=0;
        while(1){
                do{
                        flug=0;
                        ranset(num3,figure);
                        for(i=0;i<cpu_cnt;i++){
                              hitblow(&tmp_hit,&tmp_blow,list[i].dat,num3,figure);
                              if((tmp_hit!=list[i].dat_hit)||(tmp_blow!=list[i].dat_blow)){
                                      flug=1;
                               }
                       }
                        roop_cnt++;
                        if(roop_cnt>=100000000){
                               printf("条件を満たす数字を思いつきません. 終了し
ます\n");
                               return -1;
                       }
                }while(flug==1);
                printf("あなたの数字は");
                for(i=0;i<figure;i++){
                        list[cpu_cnt].dat[i]=num3[i];
                        printf("%d",num3[i]);
                }
                printf("ですか？ \n");
                printf("Hit:");
                scanf("%d",&(list[cpu_cnt].dat_hit));
                printf("Blow:");
                scanf("%d",&(list[cpu_cnt].dat_blow));
                if(list[cpu_cnt].dat_hit==figure){
                        printf("正解ですね, おつかれさまでした\n");
                        break;
                }else{
                        cpu_cnt++;
                }
        }
        return 0;
}
```

watch cpu mode のプログラムと変数リストの抜粋です. コンピュータの処理能力を活かしたゴリ押しです. いままでに与えられた条件を構造体 data 型の配列 list に保管して, そのすべての情報を満たすものを出力するという思想に基づいてます. ４桁であろうと, たかだか１万に満たない個数なので１億回のランダム試行の間に残り１通りだろうと正解を導き出すことができます. double 型の 0.9999 を 1 千万回ループさせるプログラムで試行した結果, 残り１個の時に正解を導き出せない可能性は 0.000000 という結果が出たので, ほぼ 0 とみなしていいと思います. 実際に動かしてみても存在する場合はほぼノータイムで答えを出してくるので, 問題ないと判断しました. 使ってない変数は他のモードで使っているので気にしないでください. また,cpu の精度としては基本条件を満たすものの中から乱数で数字を取り出していることになるので, 最善手には程遠いです. しかし, 実際に対戦するとそこそこの精度で答えを導いてくれるため対戦という使い方では十分だと判断しました.

5 HaskellでWebスクレイピング

情報工学課程 1 回生 川﨑 真

5.1 はじめに

Haskell は強い静的型システムと遅延評価が特徴的なプログラミング言語である. Haskell にはモナドを代表とする高等な数学理論を背景とする仕様が多い. これらの仕様を根底から理解するには, 高度な数学的スキルが必要である. しかし, 単純に道具として使うだけであれば, 難しい理論的背景を気にする必要は無い.

この記事では, カジュアルな Haskell の使い方として,Web スクレイピングプログラムを制作したので, それについて解説する.

5.2 目的

今回は,Twilog から指定アカウントの指定月のツイートをダウンロードし, タブ区切りテキストに変換するプログラムを制作した.

5.3 手法

5.3.1 概略

今回は,HandsomeSoup というライブラリを使用する. このライブラリを用いると URL と CSS のセレクタから, マッチした部分の HTML の構文木が得られる. これを加工すればツイートの本文や投稿日時などを取得できるので, これを整形すれば良い.

5.3.2 HTML の構文木の取得

TwilogScraper.hs

```
getXmlTree :: String -> String -> IO XmlTrees
getXmlTree url c = runX $ fromUrl url //> css c}

craw :: String -> Int -> Int -> Int -> IO [Tweet]
craw ui y m c = do
  r <- craw' ui y m c
  return $ sortTweet r
    where
  craw' ui y m c = do
    let url = buildUrl ui y m c
    next <- getXmlTree url ".nav-next a"
    if null next
      then getPage url y
      else do
        res <- getPage url y
        threadDelay 2000000
        aft <- craw' ui y m (c+1)
        return $ res ++ aft

getPage :: String -> Int -> IO [Tweet]
getPage url y =do
  writeLog $ "Downloading \"" ++ url ++ "\" ..."
  res <- getXmlTree url ".tl-tweet"
  writeLog $ "Downloaded \"" ++ url ++ "\"."
  case res of
    [] -> do
      writeLog $ "No Tweet found : \"" ++ url ++ "\""
      return []
    x  -> do
      let convd = mapMaybe (readTweet y) x
      return convd
```

使用ライブラリである HandsomeSoup は,HTML(XML) の構文解析と構文木の操作を行うライブラリである．今回は Cookie の処理等の細かな制御は不要なので,URL から標準の HTTP ライブラリを使用した HTML のダウンロードと構文解析を続けて行う fromUrl 関数を使う．

getXmlTree 関数では,fromUrl 関数で URL から構文木を取得する手続きを構築している. css 関数は, 文字列で与えられた CSS のセレクタにマッチする部分をリストにして返す関数である. そして,runX 関数で fromUrl 関数の返り値を起動し,//> 演算子で css 関数に流している.

craw 関数では, ページ内の「次へ」のリンクの有無で次のページの存在を確認している. そして,getPage 関数でツイート部を取得している. サーバへの負荷を軽減するために,2 秒の間隔を開けて取得するようにした.

5.3.3 HTML の構文木の解析

TwilogScraper.hs

```
readTweet :: Int -> XmlTree -> Maybe Tweet
readTweet y xtr = do
  let
    htr = convertXmlTree xtr
  d <- readDay htr y
  t <- readTime htr
  (h, i) <- parseUser htr
  c <- readContent htr
  return $ Tweet (DateTime d t) h i c
```

readTweet 関数では, ツイート部の HTML の構文木から本文・投稿日時・ユーザー名などを取得している. ここでは, 解析の失敗に対応するため,Maybe モナドを使用している. Maybe 型は失敗を表せる型であり,Maybe モナドとして使う事によって, 日時などのパースに失敗した場合値を返さない処理が可能である.

TwilogScraper.hs

```
convertXmlTree :: XmlTree -> HtmlElement
convertXmlTree (NTree (XTag name attrs) cs) =
  let
    name' = cutEachSide $ show name
    attrs' = map convertsAttr attrs
    cs' = map convertXmlTree cs
  in
    HtmlTag name' attrs' cs'

convertXmlTree (NTree (XText val) []) = HtmlText val



convertsAttr :: XmlTree -> Attr
convertsAttr (NTree (XAttr name) [NTree (XText  val) [] ]) =
  Attr (cutEachSide $ show name) val
```

convertXmlTree 関数ではパターンマッチと再帰を使い, 取得された HTML のツリーを扱いやすい形に変換している. convertAttr 関数は属性のデータの変換をする関数である.

5.4 結果

実行例

```
# ./Main sample 14 10
Downloading "http://twilog.org/sample/month-1410/allasc" ...
Downloaded "http://twilog.org/sample/month-1410/allasc".
Downloading "http://twilog.org/sample/month-1410/allasc-2" ...
Downloaded "http://twilog.org/sample/month-1410/allasc-2".
```

出力例

```
14/10/01 00:00:16    サンプルな@sample    眠い
14/10/01 00:00:17    Hage@foo    寝ろ
14/10/01 00:00:17    サンプルな@sample    アッハイ
```

ダウンロード状況とエラーは標準エラー出力に出力される. 取得結果は日時・ユーザー名と ID・本文がそれぞれタブ区切りで出力される.

結果は単純なテキストなので,cut や grep などを使って処理できる.

5.5 未実装の機能

このプログラムでは早く完成する事を目指したため, 冗長な部分や未実装な部分が存在している. 以下に, これから実装が必要であると思われるものを挙げる.

5.5.1 エラー処理

現在, 通信のエラー処理, パースのエラー処理等は行っていない. 信頼性のためには, エラーを適切に扱うように修正する必要があるだろう.

5.5.2 より簡潔な処理

HTML のツリーを解析するには, 今回したように木構造を再帰的に扱う以外に,Arrow を使った方法もある. これを用いれば,HTML のツリーから内容を得る部分がより簡潔に書けるだろう.

5.5.3 コマンドラインオプション

現在は, プログラムの振る舞いを変えられるようなオプションは存在していない. 例えば日ごとにファイルを分け, フォルダ分けして保存したり, 全ツイートを一括でダウンロードする機能などが考えられるだろう.

5.5.4 マルチプラットフォームへの対応

基本的に, このプログラムで問題になり得るのは, 改行コード・文字コードの問題である. 文字コードの変換処理を加えれば,Windows でも文字化けすることなく動作するようになるはずであるが,Windows へのライブラリのインストールに手間がかかるので取りやめた.

参考文献

[1] Haskell で HandsomeSoup を使って web スクレイピング `http://qiita.com/AyachiGin/items/029e9c90866cd76a2df7`

[2] すごい Haskell 楽しく学ぼう (オーム社, 2012 年)

[3] Real World Haskell (オライリー・ジャパン , 2009)

6 タイマー

情報工学課程 1 回生 田中 鷹太郎

6.1 はじめに

9/21 に行われた部内のハッカソンでタイマーのプログラムを作りました. もともとは Android アプリを作成しようと考えていたのですが, 環境構築がうまくいかず,Windows 上で動くタイマーを作成しました.

6.2 プログラム内容

exe ファイルを起動すると, 計りたい時間, 分, 秒の入力を求められて, タイマーが始まります.

タイマーが終わると選択画面が出てきて. キャンセル, 再実行, 続行を選択. キャンセルはプログラムの終了. 再実行は新規のタイマーをはじめるため, 時間入力を行います. 続行は再度同じタイマーを開始します.

6.3 ソースとその説明

6.3.1 ソースコード

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

int main(void) {

int ma = 0, sa = 0, ha = 0, mb = 0, sb = 0, hb = 0;
int on_button = IDTRYAGAIN;
int loop;

do{
  switch(on_button){
   case IDTRYAGAIN:
    puts("タイマーの時間をセットしてください");
    printf("時間：");
    scanf("%d", &ha);
    printf("分：");
    scanf("%d", &ma);
    printf("秒：");
```

```
    scanf("%d", &sa);

    int pastSec = ha * 3600 + ma * 60 + sa;

    hb = pastSec / 3600;
    mb = (pastSec - hb * 3600) / 60;
    sb = pastSec - hb * 3600 - mb * 60;


   case IDCONTINUE:
    printf("タイマー開始\n");

    pastSec = hb * 3600 + mb * 60 + sb;

    time_t now = 0, past = 0;

    while (pastSec >= 0) {
        now = time(NULL);
        if (now == past)
            continue;
        past = now;

        ha = pastSec / 3600;
        ma = (pastSec - ha * 3600) / 60;
        sa = pastSec - ha * 3600 - ma * 60;
        pastSec--;
        printf("%d 時間%02d 分%02d 秒のタイマーは残り%d 時間%02d 分%02d 秒です.\n", hb, mb, sb, ha, ma,
                sa);

    }
    printf("――――――――――――――――――終了――――――――――――――――――\n");
    {
        CHAR chStr[128];

        wsprintf(chStr, "%d 時間%d 分%d 秒のタイマーは終了しました.", hb, mb, sb);
        on_button = MessageBox(
        NULL, chStr, TEXT("終了"), MB_CANCELTRYCONTINUE | MB_ICONINFORMATION);
    }
   }
    if (on_button == IDCONTINUE) {
        MessageBox(
        NULL, TEXT("同じタイマーを開始します"), TEXT("同じタイマー"), MB_OK);
    loop = 1;
    }
    else if (on_button == IDTRYAGAIN) {
        MessageBox(
        NULL, TEXT("新たな時間を設定します"), TEXT("新たなタイマー"), MB_OK);
    loop = 1;
    }

    else if (on_button == IDCANCEL) {
        MessageBox(
        NULL, TEXT("タイマーを終了します"), TEXT("終了"), MB_OK);
    loop = 0;
    }
}while(loop);

    return 0;
}
```

6.3.2 経過時間を計る部分

入力した時間を一度秒数に直し変数 pastSec に代入します. ループ内で pastSec を 1 秒ごとに 1 ずつ引いていき,pastSec が 0 以下になるとループを終了するようにしました.

> 1 秒を計るには time 関数を用い現在の時間を now に代入,past と等しいのなら continue 文を用いてループ内の処理をスキップし, 等しくないのならば past に now を代入し pastSec を 1 引きループを続けます.[1]

時間を計測している間はタイマー終了までの残り時間を 1 秒ごとに出力し続けます.
また入力した時間が 90 分 100 秒などでも 0〜59 分,0〜59 秒の間に直し 1 時間 31 分 40 秒と表示するようにしました. ループが終わると, メッセージボックスが出現し, タイマーの終了を知らせてくれます.

6.3.3 メッセージボックスとタイマーの繰り返し

メッセージボックス内で, 押すボタンによって次の処理が変わります. do-while 文と switch 文でタイマーの時間の設定の所か, タイマー開始の所か, ループせずプログラムの終了かに分かれます.

メッセージボックスからタイマーの中止 (キャンセルボタン), 同じタイマーの再実行 (再実行ボタン), 新規のタイマーの実行 (続行ボタン) を選べて, それぞれのボタンを押すと一度確認ボタンが表示され, プログラムが再開します.

> windows.h というヘッダファイルを用い,Messagebox 関数でメッセージボックスを表示するようにしました [2, 3]

6.4 終りに

当初はウィンドウ上のみで動くタイマーを作ろうと考えていたのですが, うまくいかずコマンドプロンプトに入力して動くようにしました. 今後このプログラムをウインドウのみで動くようにし, より簡易的なタイマーになるように改良していきたいと思います.

参考文献

[1] 参考 url `http://www.orchid.co.jp/computer/cschool/clock1.html`

[2] 参考文献 猫でもわかる Windows プログラミング 第 4 版

[3] 参考 url `http://wisdom.sakura.ne.jp/system/winapi/win32/index.html`
Win32 API 入門/標準 Windows API/メッセージボックス

7 限定ジャンケン

情報工学課程 1 回生 山下 浩志

7.1 機能概要

限られた回数 (3 回) グー・チョキ・パーを使って星の奪い合いをする 1 対 1 のジャンケンです. プレイヤーと CPU は最初, 星を 5 つをもっています. ジャンケンに勝つと相手から星が 1 つもらえます. (引き分けの場合は変動なし) それを繰り返して,CPU の星を 0 にするか, あるいはグー・チョキ・パーを使い切った時に星の数が多いプレイヤーの勝利となります. 使用した言語は C 言語で,CUI プログラムになっています.

7.2 ソースコード

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(){
    int f;
    srand(time(NULL)); //乱数生成
    do{ //ループ (2)
    int Playerstars = 5; //プレイヤーの星数
    int Enemystars = 5; //コンピューターの星数
    int Playerhand = 0; //プレイヤーが出した手 (1:グー 2:チョキ 3:パー)
    int Enemyhand = 0; //コンピューターが出した手
    int gp = 3,cp = 3,pp = 3,ge = 3,ce = 3,pe = 3;
    //各々の手の使用回数 ex) gp = guu player ge = guu enemy
        do{ //ループ (1)
        if(gp == -1 || cp == -1 || pp == -1){
            printf("\n 反則負け\n\n"); //グー・チョキ・パーの回数制限
            break;
    }

        printf("★限定ジャンケン★\n\n ルール説明：ジャンケンに勝ったプレイヤーが星を
        相手から 1つもらう .\n\n 引き分けの場合は変動なし .\n\n"
        "それを繰り返して,相手の星数を 0にすれば勝利 .\n\n また,お互いの出せる手を
        使い切った状態で星の多いプレイヤーの勝利 .\n\n"
        "あなたの残り星数%2d\n\n 手の使用可能な回数 グー：%d チョキ:%d パー：%d\n\n"
        "相手の残り星数%2d\n\n 手の使用可能な回数 グー：%d チョキ:%d パー：%d\n\n"
```

```
    "※ 1,2,3以外の数値を入力すると反則負けとなります.\n\n
    また,使用回数のない手を出そうとした場合も反則負けとなります.\n\n"
    "数字以外を入力するとバグります.避けて下さい.\n\n"
    "あなたの手を入力して下さい.1:グー 2:チョキ
    3:パー=>",Playerstars,gp,cp,pp,Enemystars,ge,ce,pe);
    scanf("%d",&Playerhand);

    if(Playerhand != 1 && Playerhand != 2 && Playerhand != 3){
        printf("\n 反則負け\n\n");//1,2,3以外の値が入力された場合に対応する
        ため(アルファベット,記号は対応できなかったorz)
        break;
}
    //コンピューターが出す手の条件分岐
    if(ge > 0 && ce > 0 && pe > 0){
    Enemyhand = rand()%3 + 1;
}
    if(ge == 0 && ce > 0 && pe > 0){
    Enemyhand = rand()%2 + 2;
}
    if(ge > 0 && ce == 0 && pe > 0){
    Enemyhand = rand()%2;
}
    if(ge > 0 && ce > 0 && pe == 0){
    Enemyhand = rand()%2 + 1;
}
    if(ge == 0 && ce == 0 && pe > 0){
    Enemyhand = 3;
}
    if(ge == 0 && ce > 0 && pe == 0){
    Enemyhand = 2;
}
    if(ge > 0 && ce == 0 && pe == 0){
    Enemyhand = 1;
}
    if(Playerhand == 1){gp = gp - 1;}
    if(Playerhand == 2){cp = cp - 1;}
    if(Playerhand == 3){pp = pp - 1;}
    if(Enemyhand == 1){ge = ge - 1;}
    if(Enemyhand == 2){ce = ce - 1;}
    if(Enemyhand == 3){pe = pe - 1;}
    if(Enemyhand == 0){pe = pe - 1;}
    //ジャンケンの勝ち負けの判定
    if(gp != -1 && cp != -1 && pp != -1){ //左の条件式は,残り回数のない手を
    使用した時に勝負の判定が表示されるのを防ぐため
        if(Playerhand == 1 && Enemyhand == 2 || Playerhand == 2 &&
        Enemyhand == 3 || Playerhand == 3 && Enemyhand == 1){
        printf("\n 結果：勝ち\n\n");
        Playerstars = Playerstars + 1;
        Enemystars = Enemystars - 1;
        }
        if(Playerhand == 2 && Enemyhand == 1 || Playerhand == 3 &&
```

```
            Enemyhand == 2 || Playerhand == 1 && Enemyhand == 3){
            printf("\n 結果：負け\n\n");
            Enemystars =Enemystars + 1;
            Playerstars =Playerstars - 1;
            }
            if(Playerhand == Enemyhand){
            printf("\n 結果：引き分け\n\n");
            }
        } // //ループ (1)
            }while(Playerstars != 0 && Enemystars !=0 && ge+ce+pe != 0 &&
            gp+cp+pp != 0 && (Playerhand == 1 || Playerhand == 2 ||
            Playerhand == 3) );
                //1ゲームの勝ち負けの判定
        if(gp != -1 && cp != -1 && pp != -1){ //左の条件式は,残り回数のない手を
        使用した時に勝負の判定が表示されるのを防ぐため
        if(Playerstars>Enemystars){ //結果表示の分岐
        if(Playerhand ==1 || Playerhand == 2 || Playerhand ==3){
            printf("\n あなたの星数%2d\n\n 相手の星数%2d\n\n あなたは
            勝ちました.",Playerstars,Enemystars);
            }
        if(Playerstars<Enemystars){
            printf("\n あなたの星数%2d\n\n 相手の星数%2d\n\n あなたは
            負けました.",Playerstars,Enemystars);
            }
        if(Playerstars==Enemystars){
            printf("\n あなたの星数%2d\n\n 相手の星数%2d\n\n
            引き分けました.",Playerstars,Enemystars);
            }
                        }
                            }
    printf("もう一回やりますか？ \n\n 続けたい場合は 9 を入力して下さい.\n\n
    終了したい場合はそれ以外を入力して下さい.");
    scanf("%d",&f);
    }while(f == 9); //ループ (2)
    return 0;
}
```

7.3 最後に

このプログラムの制作に至った理由としては, 普通のジャンケンをする CUI プログラムだったらありふれているので, 限定ジャンケンを作ろうと思いました. 正直,DX ライブラリを用いて GUI で制作しようと考えていましたが, 挫折しました. こうしてまた, ゲームを作る機会があればその時は,GUI で動くものにしたいです.

編集後記

編集担当の木本です.Lime 50 号はいかがだったでしょうか. 今年は合宿を行わなかったためか記事が少なめですが, 様々な種類の記事があるため読んでいて飽きてしまうといったことはないでしょう. 少なくとも私としては自分の知らないことに関する記事が多かったため興味深い 1 冊となっております. 今回も先輩・後輩の力を合わせて無事 Lime を発行できて一安心といったところで, 次号の Lime にご期待くださいませ.

平成 26 年 11 月 12 日
編集担当　木本 大介

Lime Vol.50

平成 26 年 11 月 15 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/