ソフトウェア概論及び演習 [演習] (2006/11/17)

Ver. 1.0

栗野 俊一
kurino@math.cst.nihon-u.ac.jp
http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2006/soft/soft.html

2006 年 11 月 17 日

概 要

ソフトウェア概論及び演習[1]の Tom & jerry の解説[2]

目 次

[1] http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2006/soft/soft.html
[2] http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2006/soft/20061117/tj/tj.html

1 実習の目標

1.1 モジュール

C 言語に関らず、プログラミング言語では、複数のプログラム部品を組み合わせることによって一つのプログラムを作りあげることができる。

このように、組み合わせを前提に作成されたプログラムの部品を「モジュール」と呼ぶことにする。

C 言語では、「関数」が、「モジュール」となる。

C 言語のプログラムでは、(原則として) main 関数が必要だが、main 関数以外の関数は、モジュールとして扱われる。

1.2 実習の目標

本日の実習の目的は、モジュールの利用によって様々な利点が得られるので、それを体験しようというものである。

2 実習の内容

2.1 実習の材料

今回の実習では、前回 (2006/11/10) の演習[3]の課題 soft17_1 を題材とする。

2.2 ファイル分割

soft17_1 は、既に複数の関数からなっているので、このままでも良いのだが、後のことを考えて、次のような形で、ファイルを分割する。

- tom.c[4]:トムの関係のファイル

```
int cat_attacks(void)
int jump_distance(void)
```

- jerry.c[5]:ジェリィ関係のファイル

```
int mouse_runs_away(int cmd)
int runaway_distance(int cmd)
```

- display.c[6]:表示関係

[3] http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2006/soft/20061110/20061110.html
[4] v001/tom.c
[5] v001/jerry.c
[6] v001/display.c

```
void display_guide(void)
void display_scene(void)
void display_result(int game)
```

- game.c[7]: main

```
main()
```

元のファイルから、上記の tom.c jerry.c display.c を取り除いたもので、更に、これに次の include を追加する。

```
#include "display.c"
#include "jerry.c"
#include "tom.c"
```

game.c 以外は、該当する関数を単にそのまま、抜き出して、新しい名前を付けて保存するだけである。game.c は元の soft17_1.c から、逆に、該当する関数を取り去ったものであるが、上記にあるように、取り去った後に、#include を指定することによって、結果的に、元の soft17_1.c と同じように動作する。

- 実習
  - game.c をコンパイルし、ファイルを分割する前と同じ動きをすること確認する。
  - 実施例[8]

2.3 表示の変更

表示周りは、全て display.c に集まっているので、これを変更することにより、表示の様子を変更することができる。

2.3.1 エスケープシーケンス

エスケープシーケンス[9][10]機能とは、特別な文字列[11]を、画面上[12]に出力することにより、様々な効果をもたらす機能のことであり、状況によって使えたり、使えなかったりするのだが、Windows 上では利用可能[13]なので、これを利用して、表示上の効果を実現する。

[7] v001/game.c
[8] v001/v001.html
[9] http://ja.wikipedia.org/wiki/%E3%82%A8%E3%82%B9%E3%82%B1%E3%83%BC%E3%83%97%E3%82%B7%E3%83%BC%E3%82
[10] C 言語の printf 関数で利用するエスケープシーケンス (\t,\n など..) と同じ呼び方だが、異なるものであることに注意。ややっこしいが...。
[11] この文字列をエスケープシーケンスと呼ぶ
[12] 画面以外では意味がないことに注意。画面特有の機能と考えてよい。
[13] http://kasai86.ld.infoseek.co.jp/dos/esc.htm

2.3.2 エスケープシーケンス機能の利用

エスケープシーケンス機能は、単に、出力する文字列に特別な符号を追加するだけであるが、プログラムの読みやすさを実現するために、エスケープシーケンス機能を実現するためファイル (esc.h[14]) を作成し、それを display.c の中で利用する。

リスト 1: esc.h

```
#define crt_flush() fflush(stdout)
#define crt_out(s) fputs(s,stdout);crt_flush()
#define crt_pos(x,y) printf("\x1b[%02d;%02dH",(y)+1,(x)+1)
#define crt_clear() crt_out("\x1b[2J")

#define        CRT_BLACK        30        // 黒
#define        CRT_RED                31        // 赤
#define        CRT_BLUE        32        // 青
#define        CRT_PURPLE        33        // 紫
#define        CRT_GREEN        34        // 緑
#define        CRT_YELLOW        35        // 黄
#define        CRT_SKY_BLUE        36        // 水色
#define        CRT_WHITE        37        // 白

#define        CRT_DEFAULT        0        // デフォルト
#define CRT_REVERSE        7        // 反転

#define crt_color(color) printf("\x1b[%dm",color)
```

2.3.3 表示の変更

表示の変更として、次の三つを行う。

- 準備
  - 変更の対象は display.c[15]である。
  - 取りあえず、先頭に次の行を挿入し
- 画面の消去と、説明表示開始位置の指定
  - オリジナルでは、説明表示は画面の先頭だが、最初から、画面の真中当りに表示するように変更してみる。
  - 変更の対象は、display.c の中の void display_guide(void)
- トムとジェリーの表示位置を固定化
- プログラム終了時の結果の表示への色付け
- 実習
  - esc.h を作成しなさい。

[14] v002/esc.h
[15] v001/display.c

- display.c の display_guide を変更して、説明を画面中央に表示するようにしなさい。
  実際に、game.c をコンパイルし、これが実現できていることを確認すること。
- display.c の display_scene を変更して、状況を画面下部の一定の位置にに表示するようにしなさい。
  実際に、game.c をコンパイルし、これが実現できていることを確認すること。
- display.c の display_result を変更して、結果に色が付くようにしなさい。
  実際に、game.c をコンパイルし、これが実現できていることを確認すること。
- display.c に、自分なりの変更を加えて、色々と試してみなさい。
  例えば、次のようなことが考えらえる。
  * 結果の色に、更に、反転表示を行ってみる。
  * 説明文の中の大事な部分に、色をつけてみる。
  * 状況表示で、トムやジェリーに色を付けてみる。
  * 2006/07/14 に利用したプログラム[16]に esc.h を組み込んで
- 実施例[17]

2.4 トムの振舞いの変更

2.4.1 トムの振舞い

トムの振舞いは、tom.c[18]の中で記述されている jump_distance によって定められている。
この関数の振舞い、簡単に言えば、次のような流れになっている。

- 現在の時間の秒 (bdt.tm_sec) から乱数 (8 で割った余りなので 0〜7) を 4 から引いて結果 (4〜-3 となる) を rad に収める。
- 現在の自分の位置 (tom) と、ジェリーの位置 (jerry) の差を求め、これに、rad を加えものが、ジェリーの移動位置 (d) になり、これが答となっている。

この方法なので、次のような現象がおきる。

- 最初にトムが左の方角にいたとしても、いきなり、ジェリーの近くに来る (突然ワープする)。
- トムは、ジェリーの前後の 8 の範囲に行くことになり、更に、乱数が 0 (秒数が 8 で割切れる) 場合は、常に、捕まってしまうことになる。

これは、ちょっと、トムも芸がないし、また、ジェリーの逃げ方と無関係に勝敗が決るのもつまらない[19]。
そこで、tom.c (の中の jump_distance) を変更して、もう少しゲーム風にしてみよう。

---

[16] http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2006/soft/20060714/program/index.html
[17] v002/v002.html
[18] v001/tom.c
[19] まあ、トムがジェリーの周りをウロチョロして、中々捕まらないという点では、確かに、原作に忠実といえないこともないが...。

2.4.2 トムの振舞いの設計

では、トムの振舞いをどのように考えてみればよいだろうか?トムがアニメの主人公 (?) としても、やっぱり、現実の制約を受けることは間違いない。特に、トムはジェリーに比較して体が大きいので慣性法則は、無視できないだろう。

そこで[20]、次のような方針を考える。

- トムは、現在の速度 (tom_v とする) をもっていることにする。
- トムは、現在の速度を少しだけかえる (可速度を持つ) ことができる。
- トムの移動は、変更後の速度だけ移動する。
- トムの方針は、現在の状態 (トムの位置、速度、そして、ジェリーの位置) から移動距離を求める

2.4.3 トムの振舞いの変更

上記の設計方針の為に、プログラムに次の変更を加える。

- game.c[21]に、変数 tom_v に関する処理を二行追加する。

リスト 2: 変数 tom_v の宣言 (game.c への変更 1)

```
/* (2) 状態変数の宣言 */
int      tom, jerry;            /* ネコ, ネズミの位置 */
int        tom_v;                       /* ネコの速度 */                  /* ←(新規追
加) */
```

リスト 3: 変数 tom_v の初期化 (game.c への変更 2)

```
        tom   = CAT;
        tom_v = 0;                  /* ネコの速度の初期変化 */ /* ←(新規追加) */
        jerry = MOUSE;
```

- tom.c[22]の jump_distance を全面書換え。

[20]などと、色々ともっともらしいことを述べているようだが、当然、これは、単に、次の設計方針を正当化するためだけの「故事付け」でしかない。

[21]v002/game.c

[22]v002/tom.c

2.4.4 再設計

上記のトムの設計だと、最初、巣穴の前に待ち構えていたトムが急激にジェリーに近付いてくる様子が実現され、最初の要望は改善されたのだが、問題は、一旦すれちがってしまうと、もう、トムがジェリーに追い付くことができない。

方向転換が難しいという点が仇になったわけだ。

これでは、ゲームとして余りにも面白くないので、方向転換が急激にできるようにしよう[23]。

すなわち、次のような変更を tom.c[24]の中にある jump_distance に加えることにする。

リスト 4: tom.c の変更前

```
} else if ( d > 0 ) {                /* すれちがった */
  tom_v = tom_v + 1;                /* 右方向に加速 */
} else {
```

リスト 5: tom.c の変更後

```
} else if ( d > 0 ) {                /* すれちがった */
  if ( tom_v < 0 ) {                /* すれちがったので急転換を許す */
    tom_v = 2;
  } else {
    tom_v = tom_v + 1;          /* 右方向に加速 */
  }
} else {
```

また、すれちがった後に、追い掛ける余裕を与えるために、ゲーム板を少し大きくしてみよう。この為に、game.c[25]に手を加えよう。

リスト 6: game.c の変更前

```
#define  LENGTH      54          /* 通路の長さ */
#define  HOLE        50          /* 穴の位置 */
```

リスト 7: game.c の変更後

```
#define  LENGTH      64          /* 通路の長さ */
#define  HOLE        60          /* 穴の位置 */
```

これで、多少ゲームとしてバランスが取れるようになった。

- 実習
  - 実際に、tom.c 書換えてみなさい。
  - 自分なりに tom.c をつくってみなさい。
  - 実施例 (一回目)[26]
  - 実施例 (ニ回目)[27]

[23]「慣性の法則」はどうなったんだとの、お叱りの声も聞こえそうだが、なんせ、トムはアニメの主人公なのだから... (と、さっきと全く違う主張を...)

[24]v003/tom.c

[25]v003/game.c

[26]v003/v003.html

[27]v004/v004.html

2.5 ジェリーの挙動

ジェリーは人間が操作することになっているが、これを計算機に任せることを考えてみよう。

この為には、jerry.c[リスト 8] の runaway_distance を書き換えれば良い。この関数は、引数として、人間から入力をおこなっているが、これは無視して、計算機の考えた結果を返せばよい。

ここでは、簡単に、トムの挙動を無視して、兎に角、穴に入ることを考えることしよう[28]。

リスト 8: jerry.c の変更後の runaway_distance

```
int runaway_distance(int cmd)
{

  switch ( HOLE - jerry ) {        /* もし、移動可能な場所に HOLE があれば */
  case -1:
  case 1:
  case 3:
  case 5:
    return HOLE - jerry;        /* HOLE に飛び込む */
    break;
  default:                        /* そうでなければ、ひたすら右へ.. */
    return 5;
  }
}
```

- 実習
  - 実際に、jerry.c 書換えてみなさい。
  - 自分なりに jerry.c をつくってみなさい。
  - 今度は、tom.c 側を人間に操作できるようにするには、どうすれば良いか?
  - 友達と、tom.c / jerry.c を交換して、戦えさせてみよう。
  - 実施例[29]

3 提出

自分なりに作った、jerry.c を提出してください。

4 Link

- トムとジェリー
  - トムとジェリー (Wikipedia)[30]
  - Tom and Jerry Online[31]
- エスケープシーケンス
  - エスケープシーケンス (Wikipedia)[32]

[28] 当然、このような形では、トムにあっというまに食べられてしまう..
[29] v005/v005.html
[30] http://ja.wikipedia.org/wiki/%E3%83%88%E3%83%A0%E3%81%A8%E3%82%B8%E3%82%A7%E3%83%AA%E3%83%BC
[31] http://www.tomandjerryonline.com/
[32] http://ja.wikipedia.org/wiki/%E3%82%A8%E3%82%B9%E3%82%B1%E3%83%BC%E3%83%97%E3%82%B7%E3%83%BC%E3%82

- VT-100 エスケープシーケンスについて[33]
- Short Program （モグラたたき）[34] LSI-C 用なので、そのままでは利用できない。
- 2006/07/14 に利用したプログラム[35]

[33]http://hp.vector.co.jp/authors/VA016670/escape_code.html
[34]http://www.geocities.co.jp/SiliconValley-Bay/7437/c/short.htm
[35]http://edu-gw2.math.cst.nihon-u.ac.jp/~kurino/2006/soft/20060714/program/index.html