プログラミング言語実験・C 言語　第 3 回課題レポート

I 類 メディア情報学
氏名：LEORA DAVID
学籍番号：2210745

2024 年 05 月 05 日

課題 5（コンピュータ大貧民プログラムの実行状況のスクリーンショット）

大貧民サーバを起動し、大貧民標準クライアント（tndhm_devkit_c-20180826.tar.gz に同梱されている方）を 5 台起動する。サーバの実行画面（クライアント名が default と表示されている対戦画面）とグラフの画面（棒グラフか線グラフ）の計 2 画面のスクリーンショットを撮った。

課題 5 の実行結果

まず、ターミナルを起動して、「コンピュータ大貧民の実行」ページの通りの手順でサーバを起動した。各フォルダから configure を行い、make を行った。実行するには、以下のコマンドを実行する。ポート番号は 52745 とした。

```
$ ./tndhms -p 52745
```

また、標準クライアントを起動するには、同じく「コンピュータ大貧民の実行」ページの通りの手順でクライアントを起動した。実行するには、以下のコマンドを実行する。

```
$ ./client -p 52745 &
```

実行したサーバの実行画面と線グラフの画面のスクリーンショットを以下の図 1 と図 2 に示す。

図 1　サーバの実行画面

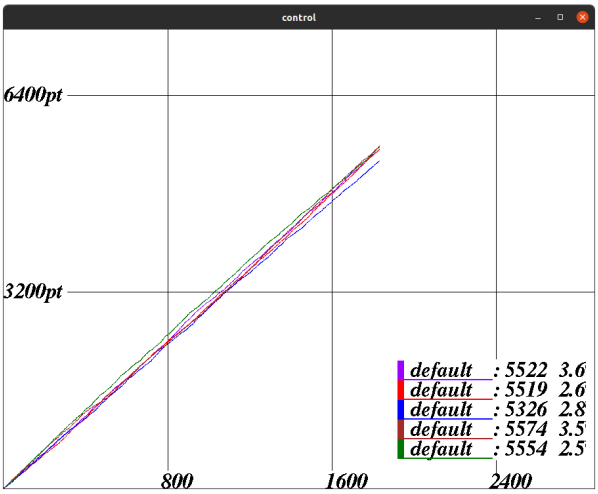

図 2　線グラフの画面

課題６（ペア出し機能の実装）

コンピュータ大貧民教育用クライアント（tndhmc-0.03.tar.gz）のディレクトリ src にある、select_cards.c などを改変し、ペア出し機能を実現した。この課題では、場にカードがない状況で、かつ提出するカードにジョーカーを含まない場合について実装した。実装が完了したら、大貧民サーバを立ち上げゲームを実行し、ペア出しが行われている様子がわかるスクリーンショットを取得した。（サーバの実行画面中のクライアントプログラムが Normal と表示されているかを確認した）。

実装した「ペア出し機能」のソースコードは以下のソースコード１の通りである。具体的に、daihinmin.c の make_info_table 関数と search_low_pair 関数を追加した。make_info_table 関数は、自分の手札（my_cards 配列）にあるカードの数を集計し、その情報を info_table 配列に格納する。search_low_pair 関数は、手札の中から一番弱いペアを探し、dst_cards 配列に格納する。dst_cards 配列は、提出するカードを格納する配列である。

まとめとして、配列は主に以下の目的で使用される。

- my_cards 配列：自分の手札のカードを格納する配列である。各要素は特定のカードが手札に存在するかどうかを示す。
- info_table 配列：自分の手札のカードの数を集計する配列である。また、info_table 配列は、make_info_table 関数で更新される。
- dst_cards 配列：提出するカードを格納する配列である。また、dst_cards 配列は、search_low_pair 関数で更新される。
- select_cards 配列：提出するカードを格納する配列である。

また、以下のソースコードによって、search_low_pair 関数で、info_table 配列をチェックして最大枚数また

は一番弱いペア（２枚の場合）を dst_cards に格納する。これにより、最大枚数のカードか一番弱いの２枚カードの提出が可能になる。これらの関数と配列の使用により、ペア出し機能が実現されている。

ソースコード 1　daihinmin.c

```
1  #include<stdio.h>
2  #include<string.h>
3  #include<strings.h>
4  #include"common.h"
5
6  void search_low_card(int out_cards[8][15],int my_cards[8][15],int use_joker_flag){
7    /*低い方から探して最初に見つけたカードを一枚
8      ,にのせる。out_cardsがのとき
9      use_joker_flag1カードが見つからなければ,,を一枚にのせる。jokerout_cards
10   */
11   int i,j,find_flag=0;
12
13   clear_table(out_cards); テーブルをクリア//
14   for(j=1;j<14&&find_flag==0;j++){ 低い方からさがし//
15     for(i=0;i<4&&find_flag==0;i++){
16       if(my_cards[i][j]==1){ カードを見つけたら//
17         find_flag=1; フラグを立て//
18         out_cards[i][j]=my_cards[i][j]; //にのせ out_cardsループを抜ける。,
19       }
20     }
21   }
22   if(find_flag==0&&use_joker_flag==1){ 見つからなかったとき//
23     out_cards[0][14]=2; ジョーカーをのせる//
24   }
25 }
26
27 void make_info_table(int info_table[8][15], int my_cards[8][15])
28 {
29   int i;
30
31   clear_table(info_table);
32   for(i=1;i<=13;i++){
33     info_table[4][i]=my_cards[0][i]+my_cards[1][i]+my_cards[2][i]+my_cards[3][i];
34   }
35 }
36
37 int search_low_pair(int dst_cards[8][15], int info_table[8][15], int my_cards[8][15]){
38   int i, j, max_i = 0, max_val = 0;
39   clear_table(dst_cards);
40   for(i=1; i<=13; i++){
41     if(info_table[4][i] > max_val){
42       max_val = info_table[4][i];
```

```
43          max_i = i;
44        }
45      }
46      if(max_val >= 2){
47        for(j=0; j<=3; j++) dst_cards[j][max_i] = my_cards[j][max_i];
48        return 1;
49      }
50      else return 0;
51    }
```

ソースコード１の考察（最大枚数 OR 一番弱いペアの実装）

ソースコード１より、make_info_table 関数は、自分の手札にあるカードの数を集計し、その情報を info_table 配列に格納する。具体的に、自分の手札（my_cards 配列）にあるカードの数の情報を info_table 配列の４行目（info_table[4][i]）に格納している。これにより、make_info_table 関数は、search_low_pair 関数で使用される。dst_cards 配列をまずクリアし、for 文で info_table 配列の４行目（info_table[4][i]）を左から右に順番に探し、**最大枚数のカード**を探す。最後に、見つかった最大枚数のカードを dst_cards 配列に格納する。
もし最大枚数が２の場合、search_low_pair 関数は、自分の手札の中から一番弱いペアを探し、dst_cards 配列に格納する。また、値が２以上のカードが最後まで見つからない場合、０を返す（０を返すということは、ペアが見つからなかったということである)。

次に、select_cards.c のプログラムの select_cards_free の関数に加えました。書いたプログラムをソースコード２に示す。

ソースコード 2　select_cards.c

```
1   #include<stdio.h>
2   #include"common.h"
3   #include"daihinmin.h"
4   #include"select_cards.h"
5
6   void select_change_cards(int out_cards[8][15],int my_cards[8][15],int num_of_change){
7     /*
8      * カード交換時のアルゴリズム
9      * 大富豪あるいは富豪が、大貧民あるいは貧民にカードを渡す時のカードを
10     * カードテーブルと交換枚数に応じて、my_cardsnum_of_change
11     * 低いほうから選びカードテーブルにのせる out_cards
12    */
13    int count=0;
14    int one_card[8][15];
15
16    clear_table(out_cards);
17    while(count<num_of_change){
18      search_low_card(one_card,my_cards,0);
19      diff_cards(my_cards,one_card);
```

```
20        or_cards(out_cards,one_card);
21        count++;
22      }
23    }
24
25    void select_submit_cards(int out_cards[8][15],int my_cards[8][15], state *field_status)
          {
26      int select_cards[8][15];
27
28      clear_table(select_cards);
29
30      if(field_status->is_rev==0){
31        if(field_status->is_no_card==1){ 場にカードが無いとき//
32          select_cards_free(select_cards, my_cards, field_status); 通常時の提出用//
33        }else{
34          select_cards_restrict(select_cards,my_cards, field_status); 通常時の提出用//
35        }
36      }else{
37        if(field_status->is_no_card==1){ 場にカードが無いとき//
38          select_cards_free_rev(select_cards, my_cards, field_status); 革命時の提出用//
39        }else{
40          select_cards_restrict_rev(select_cards, my_cards, field_status); 革命時の提出用//
41        }
42      }
43
44      copy_table(out_cards, select_cards);
45    }
46
47    void select_cards_free(int select_cards[8][15], int my_cards[8][15], state *
          field_status){
48      int info_table[8][15];
49
50      make_info_table(info_table, my_cards);
51      if(count_cards(select_cards)==0)
52        search_low_pair(select_cards, info_table, my_cards); // 手持ちの一番弱いペアを提出
              する
53      if(count_cards(select_cards)==0)
54        search_low_card(select_cards,my_cards,0); // 手持ちの一番弱いカードを単騎で提出する
55    }
56
57    void select_cards_restrict(int select_cards[8][15], int my_cards[8][15], state *
          field_status){
58      int tmp_cards[8][15];
59
60      copy_table(tmp_cards, my_cards);
61
```

```
62      if(field_status->is_sequence==1){ // 場が階段のとき
63        if(field_status->is_lock==1){ // 場が縛られている
64
65        }else{ // 場が縛られていない
66
67        }
68
69      }else if(field_status->quantity > 1){ // 場がペアのとき
70        if(field_status->is_lock==1){ // 場が縛られている
71
72        }else{ // 場が縛られていない
73
74        }
75      }else{ // 場が単騎のとき
76        if(field_status->is_lock==1){ // 場が縛られている
77          remove_suit(tmp_cards, field_status->suit, 1);
78          remove_low_card(tmp_cards, field_status->order, 0);
79          search_low_card(select_cards,tmp_cards,1);
80        }else{ // 場が縛られていない
81          remove_low_card(tmp_cards, field_status->order, 0);
82          search_low_card(select_cards,tmp_cards,1);
83        }
84      }
85    }
86
87    void select_cards_free_rev(int select_cards[8][15], int my_cards[8][15], state *
          field_status){
88    }
89
90    void select_cards_restrict_rev(int select_cards[8][15], int my_cards[8][15], state *
          field_status){
91    }
92
93    void operate_my_cards(int my_cards[8][15], state *field_status){
94    }
95
96    void operate_field_cards(int ba_cards[8][15], state *field_status){
97    }
```

ソースコード２の考察

ソースコード 2 より、select_cards_free 関数は、場にカードがない状況で、かつ提出するカードにジョーカーを含まない場合について実装した。具体的に、場にカードがない場合（field_status->is_no_card==1）に、select_cards_free 関数を呼び出す。select_cards_free 関数は、make_info_table 関数を呼び出し、info_table 配列に自分の手札のカードの数を集計する。そして、他のプログラム（common.c）から count_cards 関数を呼

び出し、select_cards 配列のカードの数が 0 の場合（選択するカードがない場合）に、search_low_pair 関数を呼び出す。search_low_pair 関数は、自分の手札の中から**一番弱いペアではなく、最大枚数**を探し、dst_cards 配列に格納する。
もし、最大枚数が 2 の場合、search_low_pair 関数は、自分の手札の中から一番弱いペアを探し、dst_cards 配列に格納する。また、search_low_pair 関数でペアが見つからない場合、search_low_card 関数を呼び出す。

search_low_card 関数は、自分の手札の中から一番弱い単騎カードを探し、dst_cards 配列に格納する。最後に、dst_cards 配列は、提出するカードを格納する配列である。

課題６の実行結果

実装した「ペア出し機能」を確認するために、大貧民サーバを立ち上げ、ゲームを実行した。サーバの実行画面中のクライアントプログラムが Normal と表示されているかを確認した。ペア出しが行われている様子がわかるスクリーンショットを以下の図３に示す。

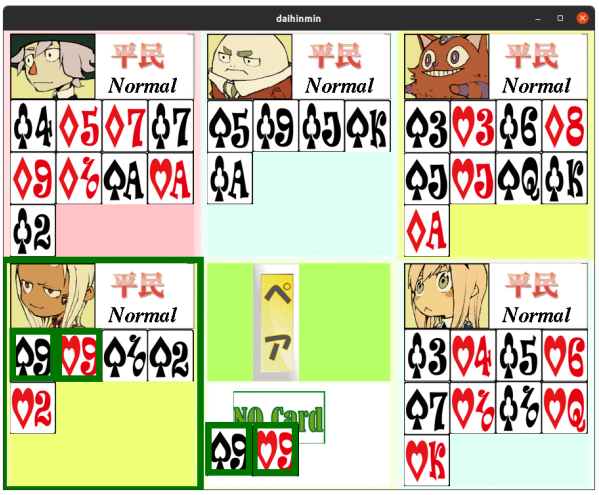

図３　ペア出し機能の実行画面（カードが２枚ずつ提出されている）

また、３枚が提出される場合のスクリーンショットを以下の図４に示す。

図４　ペア出し機能の実行画面（カードが３枚ずつ提出されている）

参考文献

[1] 第３回　コンピュータ大貧民（大貧民の実行とペア出し機能の実装），
URL : https://www.ied.inf.uec.ac.jp/text/laboratory/C/third_week/index03.html