ネットワークプログラミング2

自由課題

提出日　2021年7月27日 14:20

組番号 409
学籍番号　17406

氏名　金澤雄大

1　目的

ネットワークプログラミング 2 で学習した内容への理解を深めるために自由課題として製作物を作成することを目的とする.

2　製作物の説明

本章では製作物の説明として, 製作物の概要, 花札の基礎と用語, こいこいのルールの 3 つについて述べる.

2.1　製作物の概要

製作物として同期通信を利用して, 花札で遊べるゲームの 1 つである「こいこい」を作成した. こいこいは 2 人または 3 人で同じ絵柄の札を揃えて役を作るカードゲームで, ここでは 2 人での対局を想定している. また実装したこいこいのルールは任天堂のルール !文献 を参考に, 一部プログラミングがしやすいように改変した.

2.2　花札の基礎と用語

花札の基礎と用語について説明する. 花札のカードは全 48 枚の札から構成されている. 札には 1 月から 12 月までの札が各 4 枚ずつ存在する. 各月の札の絵柄は図 1〜図 12 に示す通りである. 札は月とは別に描かれている絵の種類で光札, タネ札, タン札, カス札の 4 種類に分けられる. 光札は最もレア度の高い札で 1 月の鶴が描かれている札, 3 月の桜と幕が描かれている札, 8 月の月が描かれている札, 11 月の小野道風にカエルが描かれている札, 12 月の桐に鳳凰が描かれている札の 5 枚のみである. タネ札は光札以外で動物や橋, 盃 (さかずき) が描かれている札である. タン札は短冊が描かれている札である. カス札は植物のみが描かれている札である. 間違えやすい札として 11 月のカス札がある. 11 月のカス札は黒と赤で構成されていて, 太鼓のようなものが描かれているがカス札である. 例外として 9 月のタネ札 (盃が描かれている札) はタン札とタネ札の両方としてカウントする.

図 1: 1 月 松に鶴

図 2: 2 月 梅にうぐいす

図 3: 3 月 桜に幕

図 4: 4 月 藤にほととぎす

図 5: 5 月 菖蒲 (アヤメ) に八ツ橋

図 6: 6 月 牡丹に蝶

図 7: 7 月 萩に猪

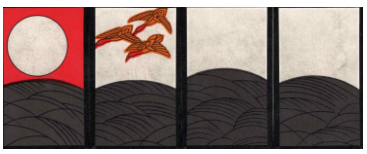

図 8: 8 月 ススキに月・雁

図 9: 9 月 菊に盃 (さかずき)

図 10: 10 月 紅葉に鹿

図 11: 11 月 小野道風にカエル, 柳にツバメ

図 12: 12 月 桐に鳳凰

2.3 こいこいのルール

こいこいは花札を使用するゲームの 1 つである. 対局は 2 人または 3 人で行うが, ここでは 2 人で対局を行うときのルールについて説明する. こいこいは 12 回親と子を変えて対局し, 12 回目の対局が終了したときに得点が高いほうが勝利するというルールである.

対局手順について説明する. 1 回目の対局では, はじめに親と子を決める. 親と子の決め方はまず, 48 枚全ての札を裏返してシャッフルする. 次に親 (先攻) と子 (後攻) を決めるために, シャッフルした札から 1 枚ずつめくる. めくった札の月が早いほうが親, 遅いほうが子である. 月が同じときは親子決めをやり直す. 親子が決まったら, 親が全ての札を裏返してシャッフルする. そして親が親, 子, 場にシャッフルした札を配る. 親と子は裏向き, 場は表向きで札を配る. 残った札は山札として裏向き重ねて置いておく. これで対局を始める準備ができた. また 2 回目以降の対局は親と子を交互に交代する.

対局は親のターンから始まり, 親のターンが終わると子, 子のターンが終わると再び親のターンに戻る形式である. ターン中の行動は図 13 のフローチャートの通りである. 図 13 の「こいこい」とはまだゲームを続けるかやめるかを選択することである. 「こいこい」を宣言すると相手のターンになり, 「あがり」を宣言すると対局が終了し, こいこいを宣言した側の勝ちとなる. こいこいを宣言した後に相手の役 (後述) ができると相手の役の得点が 2 倍になるためこいこいするかどうかは慎重に判断する必要がある.

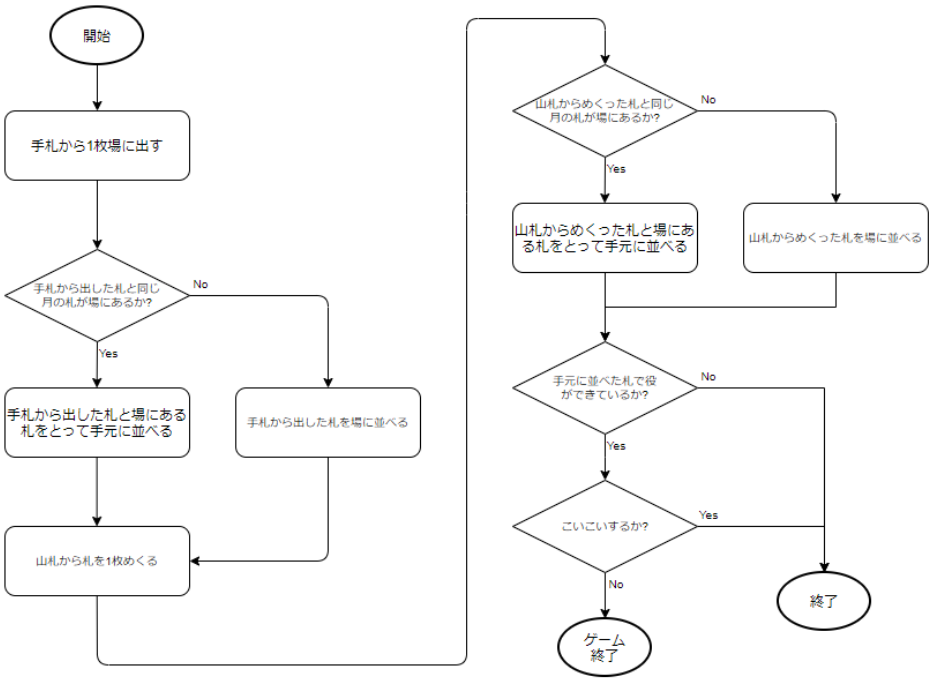

図 13: ターン中の行動手順

役について説明する. 役とは手元に特定の札が揃ったときに自分の得点になり, こいこいするかどうかを決めるためのものである. 今回は次に示す 11 個の役を実装した. ここに示す以外の役 (例:手四, くっつき) もあるがここではそれらの役の判定は行わないものとする.

- 五光 (10 点) : 光札 5 枚が全て揃う.
- 四光 (8 点) : 11 月の光札を除く光札 4 枚が揃う.
- 雨四光 (7 点) : 11 月の光札とそれ以外の光札 3 枚が揃う.
- 三光 (5 点) : 11 月の光札を除く光札 3 枚が揃う.
- 花見で一杯 (5 点) : 3 月の光札, 9 月のタネ札が揃う.
- 月見で一杯 (5 点) : 8 月の光札, 9 月のタネ札が揃う.
- 猪鹿蝶 (5 点) : 6 月, 7 月, 10 月のタネ札 3 枚が揃う.
- 赤短 (5 点) : 1 月, 2 月, 3 月のタン札 3 枚が揃う.
- 青短 (5 点) : 6 月, 9 月, 10 月のタン札 3 枚が揃う.
- タネ : タネ札 5 枚で 1 点, 1 枚増えるごとに 1 点追加.
- タン : タン札 5 枚で 1 点, 1 枚増えるごとに 1 点追加.
- カス : カス札 10 枚で 1 点, 1 枚増えるごとに 1 点追加.

3　実行環境とビルド方法

本章では, 実行環境, ファイル構成, ビルド方法の 3 つについて述べる.

3.1　実行環境

表 1 に実行環境を示す. OpenGL と GLpng は 4 年次にプログラミング演習で使用したものを用いる.

表 1: 実行環境

CPU	Intel(R) Core(TM) i7-6500U 2.50GHz
メモリ	16.0GB DDR4
OS	Microsoft Windows 10 Home
gcc	version 9.3.0
make	version 4.3

3.2　ファイル構成

リスト 1 に製作したゲームのファイル構成を示す. 本ゲームは Google Drive!リンクでダウンロードできるようになっている. mylib ディレクトリは授業中に作成したソケット通信用のライブラリである. 花札を実現するプログラムは main ディレクトリに入っている. さらに main ディレクトリ内に card ディレクトリがあり札の画像ファイルを格納している.

リスト 1: ファイル構成

```
5J_hanahuda
├── main
│   └── card
├── mylib
├── readme_img
└── readme.md
```

3.3　ビルド方法

ビルド方法について説明する. Google Drive から j17406.tar.gz をダウンロードして解凍した状態とする. まず, mylib ディレクトリで make コマンドを実行してソケット通信用のプログラムをコンパイルする. 次に main ディレクトリで make コマンドを実行して花札を実行するプログラムをコンパイルする. コンパイルができたら, s.exe と c.exe を実行すると対局が始まる.

4　プログラムの説明

本章では server.c, client.c, hanahuda.c の 3 つのプログラムの説明について述べる.

4.1 server.c

サーバー端末で動作する server.c について説明する. リスト 2 に server.c のプログラムを示す. リスト 2 の 10 行目から 28 行目は OpenGL の処理である. 12 行目から 14 行目ではウィンドウの生成を行っている. 16 行目から 21 行目で動的な画面の描画, ウィンドウサイズの変更, キーボード, マウス処理のためにコールバック関数登録をしている. 23 行目から 28 行目ではアルファチャネルの有効化と背景色の変更の処理を行っている. 29 行目から 31 行目では, サーバーのセットアップの処理を行っている. セットアップが失敗した場合は実行を終了する. 34 行目では対局の初期化を行っている. game_init 関数の処理内容は 4.3 章の hanahuda.c で説明する. 最後に 37 行目で OpenGL のメインループに入る処理を行う. サンプルの五目並べのプログラムでは while 文を用いたメインループでゲームの進行を行ったが, この処理を OpenGL に置き換えることでグラフィカルなゲームを実現している.

リスト 2: server.c

```
1  #include<stdio.h>
2  #include "hanahuda.h"
3  #include<GL/glut.h>
4  #include <GL/glpng.h>
5
6  int main(int argc,char **argv){
7      int soc;
8      // 初期化処理
9      // 引数処理
10     glutInit(&argc,argv);
11     // 初期Windowサイズ設定
12     glutInitWindowSize(WINDOW_W,WINDOW_H);
13     // 新規Window作成
14     glutCreateWindow("hanahuda_host");
15     // 関数登録
16     glutDisplayFunc(Display);
17     glutReshapeFunc(Reshape);
18     glutKeyboardFunc(Keyboard);
19     glutMouseFunc(Mouse);
20     glutPassiveMotionFunc(PassiveMotion);
21     glutTimerFunc(500,Timer,0);
22     //  テクスチャのアルファチャネルを有効にする設定
23     glEnable(GL_BLEND);
24     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
25     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
26     // display初期化
27     glutInitDisplayMode(GLUT_RGBA);
28     glClearColor(0.28,0.48,0.32,1.0);
29     if( (soc=setup_server(PORT))==-1){
30         exit(1);
31     }
32
33     // ゲーム初期化
34     game_init(soc,0);
35
36     // glutのメインループ
37     glutMainLoop();
38     return 0;
39 }
```

4.2 client.c

クライアント端末で動作する client.c について説明する. client.c のプログラムをリスト 3 に示す. 基本的な処理の流れは server.c と同じである. server.c との違いは 31 行目から 38 行目である. 30 行目および 31 行目では接続するホスト (サーバー) 名を入力させる処理を行っている. ホスト名は「localhost」,「192.168.10.4」のように入力する. 33 行目で実行している chop_newline 関数は mylib ディレクトリで定義されている関数

で入力した文字列の末尾の改行文字をナル文字に置換する機能を持つ. 36 行目から 38 行目では入力されたホストの IP アドレスをもとにクライアントのセットアップの処理を行っている.

リスト 3: client.c

```
1  #include<stdio.h>
2  #include "hanahuda.h"
3  #include<GL/glut.h>
4  #include <GL/glpng.h>
5
6  int main(int argc,char **argv){
7      int soc;
8      char hostname[64];
9      // 初期化処理
10     // 引数処理
11     glutInit(&argc,argv);
12     // 初期Windowサイズ設定
13     glutInitWindowSize(WINDOW_W,WINDOW_H);
14     // 新規Window作成
15     glutCreateWindow("hanahuda_client");
16     // 関数登録
17     glutDisplayFunc(Display);
18     glutReshapeFunc(Reshape);
19     glutKeyboardFunc(Keyboard);
20     glutMouseFunc(Mouse);
21     glutPassiveMotionFunc(PassiveMotion);
22     glutTimerFunc(500,Timer,0);
23     //  テクスチャのアルファチャネルを有効にする設定
24     glEnable(GL_BLEND);
25     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
26     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
27     // display初期化
28     glutInitDisplayMode(GLUT_RGBA);
29     glClearColor(0.28,0.48,0.32,1.0);
30     //サーバーのホスト名の入力
31     printf("Input server's hostname: ");
32     fgets(hostname,HOSTNAME_LENGTH,stdin);
33     chop_newline(hostname,HOSTNAME_LENGTH);
34
35     //接続完了まで
36     if((soc = setup_client(hostname,PORT)) == -1){
37         exit(1);
38     }
39
40     // ゲーム初期化
41     game_init(soc,1);
42
43     // glutのメインループ
44     glutMainLoop();
45
46     return 0;
47 }
```

4.3 hanahuda.c

本節では hanahuda.c における構造体および関数の説明について述べる.

4.3.1 札の処理

札の扱いと処理について説明する. 札はリスト 4 に示す card 構造体の配列で定義されている. card 構造体は月, 画像ファイルの番号, 札の種類の 3 つをメンバとして持っている.

リスト 4: card 構造体

```
// 札用の構造体
struct cardstruct{
    int month; // 1-12
    int num; // imgの番号
    int rank; //1-4, 4:光札,3:タネ札,2:短冊札,1:カス札
};
typedef struct cardstruct card;
// 札用の構造体の配列を定義
static card cards[CARD_NUM];
```

次に山札や場札から出すときの処理について説明する. 山札と手札は札を出す, 場札は札の出し入れの処理を行う必要がある. また獲得した札を保持する配列が必要である. これらの処理を行うために, java における setter と getter にあたる関数を用意した. リスト 5 に山札, 場札, 手札, 獲得した札の配列に札を出入力する関数を示す. 関数名が pop で始まる関数は札を管理する配列から札を取り出す関数, push で始まるものは札を入力する関数である.

リスト 5: 札の入出力をする関数

```
// 山札から札を出す処理
// 札のインデックスを返す
int popDeck(void){
    deck_num++;
    return deck[deck_num-1];
}

// 場に札を入れる処理
void pushPlace(int c){
    place[place_num++]=c;
}

// 場から札を出す処理
int popPlace(int index){
    int i;
    int tmp=place[index];
    for(i=index+1;i<place_num;i++){
        place[i-1]=place[i];
    }
    place[i-1]=-1;
    place_num--;
    return tmp;
}

// 手札から札を出す処理
int popHandCard(int index){
    int i;
    int tmp;
    if(role==0){ // 親のとき
        tmp = mycard[index];
        for(i=index+1;i<mycard_num;i++){
            mycard[i-1]=mycard[i];
        }
        mycard[mycard_num-1]=-1;
        mycard_num--;
    }else{ // 子のとき
        tmp = peercard[index];
        for(i=index+1;i<peercard_num;i++){
            peercard[i-1]=peercard[i];
        }
        peercard[peercard_num-1]=-1;
        peercard_num--;
    }
    return tmp;
}

// 持ち札に札を入れる処理
```

```
48 void pushgetCard(int c){
49     if(role==0){
50         mygetcard[mygetcard_num]=c;
51         mygetcard_num++;
52     }else{
53         peergetcard[peergetcard_num]=c;
54         peergetcard_num++;
55     }
56 }
```

4.3.2 ゲームの初期化処理

ゲームの初期化処理について説明する. ゲームの初期化はリスト 6 の 3 行目から 15 行目に示す game_init 関数を実行することで行われる. game_init 関数の内部では, まず 6 行目で親と子の区別をするための変数 role に引数から受け取った値を代入する処理を行っている. 親のときリスト 2 の 34 行目に示したように, role を 0, 子のときリスト 3 の 41 行目で示したように role を 1 に設定している.

8 行目では配列の初期化を行う関数である array_init 関数を実行している. array_init 関数の定義は 44 行目から 46 行目である. 9 行目では readImg 関数を実行している. readImg 関数の定義は 18 行目から 41 行目である. readImg 関数では card ディレクトリに置いたある札の画像を取得し cardimg 配列に代入する処理を行っている. カード名を sprintf 関数で指定してから指定されたパスの画像を取得する pngBind 関数を実行することで 48 枚の札の取得を実現している.

11 行目から 14 行目では山札をシャッフルして場と手札に札を配る処理を行っている. 山札のシャッフルは 83 行目から 100 行目に定義した shuffle 関数で行っている. 花札の札には重複がないため乱数をランダムに代入して山札を生成することができない. そこで 0 から 47 までの数字が 1 つずつ入った配列を生成し, ランダムに 2 つを選択して交換する処理を何度か繰り返すことで重複のないランダムなリストを生成している. 交換回数は 10000 回としている. 山札の設定が終わったから 104 行目から 127 行目に示す arrangeCard 関数を用いて場と 2 人の手札にそれぞれ 8 まいずつ札を配る処理を行っている. このとき, 場に 3 枚以上同じ月の札がでたら山札のシャッフルと札の分配をやり直すかどうかの判定を行い, やり直しの必要があるときは 1 を返すようにしている. game_init 関数関数では返り値が 0 になるまでループさせることで場に取れない札がでることを防止している. なお本来の対局では場に 3 枚同じ札が出た時は残りの 1 枚を場に出したときに 4 枚すべて獲得できる, または場に出ている 3 枚から 1 枚をランダムに山札に戻して別の 1 枚をだし, 山札をシャッフルしなおすというルールになっている. 場に 4 枚同じ札がでたときのルールは公式なものはないが場と山札をシャッフルしなおすのが一般的であると考える.

リスト 6: ゲームの初期化処理

```
1  // 対局初期化
2  // role 0:親,1:子
3  void game_init(int soc,int argrole){
4      int tmp=1;
5      hanahuda_soc = soc;
6      role=argrole;
7
8      array_init();
9      readImg(); // 画像読み込み
10     card_init(); // 札情報をcardstruct構造体に格納
11     while(tmp==1){
12         shuffle(); // 山札シャッフル
13         tmp=arrangeCard(); // 場, 手札に札を配る
14     }
15 }
16
17 // 画像読み込み
18 void readImg(void){
19     char fname[100];
```

```
20      int i,j;
21      int cnt=0;
22      // 全札読み込み
23      for(i=0;i<MONTH;i++){
24          for(j=0;j<MONTH_CARD;j++){
25              sprintf(fname,".\\card\\%s-%d.png",month[i],j+1);
26              //printf("%s\n",fname);
27              cardimg[cnt] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
28                  &cardinfo[cnt], GL_CLAMP, GL_NEAREST, GL_NEAREST);
29              cnt++;
30          }
31      }
32      // 裏返しの黒い札(back)を読み込み
33      sprintf(fname,".\\card\\back.png");
34      cardimg[cnt] = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
35          &cardinfo[cnt], GL_CLAMP, GL_NEAREST, GL_NEAREST);
36
37      // こいこい時のポップアップ用の画像を読み込む処理
38      sprintf(fname,".\\koikoi.png");
39      koikoiimg = pngBind(fname, PNG_NOMIPMAP, PNG_ALPHA,
40          &koikoiinfo, GL_CLAMP, GL_NEAREST, GL_NEAREST);
41  }
42
43  // 配列の初期化
44  void array_init(void){
45      int i;
46      // 配列の初期化
47      for(i=0;i<PLACE_MAX;i++){
48          place[i]=-1;
49      }
50      for(i=0;i<INIT_PLACE;i++){
51          mycard[i]=-1;
52          peercard[i]=-1;
53      }
54      for(i=0;i<CARD_NUM/2;i++){
55          mygetcard[i]=-1;
56          peergetcard[i]=-1;
57      }
58      for(i=0;i<PLACE_MAX;i++){
59          place[i]=-1;
60      }
61      for(i=0;i<YAKU_NUM;i++){
62          myyaku[i]=0;
63          peeryaku[i]=0;
64      }
65  }
66
67  // 札の情報をカード構造体に格納
68  void card_init(void){
69      int i,j;
70      int cnt=0;
71      // すべての札について
72      for(i=1;i<=MONTH;i++){ // 12回(月の回数)ループ
73          for(j=0;j<MONTH_CARD;j++){ // 4回ループ
74              cards[cnt].month=i; // 月
75              cards[cnt].num=j+1; // 札番号
76              cards[cnt].rank=cardrank[i-1][j]; //  点数
77              cnt++;
78          }
79      }
80  }
81
82  // 札をシャッフル
83  void shuffle(void){
84
85      int i,rnd1,rnd2,tmp;
86      srand((unsigned) time(NULL)); // 乱数初期化
87      // 全ての札を山札に格納
88      for(i=0;i<CARD_NUM;i++){
89          deck[i]=i;
```

```
90      }
91
92      for(i=0;i<SHUFFLE_TIME;i++){
93          rnd1 = rand()%CARD_NUM; // ランダムに1枚指定
94          rnd2 = rand()%CARD_NUM; // ランダムに1枚指定
95          // 指定した札を交換
96          tmp = deck[rnd1];
97          deck[rnd1] = deck[rnd2];
98          deck[rnd2]=tmp;
99      }
100 }
101
102 // 場と手札に札を配る処理
103 // 場札に同じ月の札が3枚以上あるとき1,それ以外のとき0を返す.
104 int  arrangeCard(void){
105     int i,j,tmp;
106     // 場,親,子に8枚札を出す
107     for(i=0;i<INIT_PLACE;i++){
108         pushPlace(popDeck());
109         mycard[i] = popDeck();
110         peercard[i] = popDeck();
111     }
112         // 場の状態を確認
113     for(i=1;i<=MONTH;i++){
114         tmp=0;
115         for(j=0;j<place_num;j++){
116             if(cards[place[j]].month==i){
117                 tmp++;
118             }
119         }
120         if(tmp>=3){ // 場に3枚以上同じ月の札があるとき
121             tmp=1;
122             return tmp;
123         }
124     }
125     tmp=0;
126     return tmp;
127 }
```

4.3.3 画面描画の処理

画面描画処理について説明する. 画面描画処理が行われるタイミングはウィンドウサイズが変更されたとき, タイマーによってウィンドウの再描画関数が実行されたときの 2 つである. ウィンドウサイズが変更されるとリスト 7 に示す Reshape 関数がコールバック関数として実行される. Reshape 関数が実行されると, ウィンドウの再生成が行われるが, このゲームでウィンドウサイズが自由に変えられると困るため

リスト 7: Reshape 関数

```
1  void Reshape(int w,int h){
2      glViewport(0,0,w,h);
3      glMatrixMode(GL_MODELVIEW);
4      glLoadIdentity();
5      gluOrtho2D(0,w,0,h);
6      glScaled(1,-1,1);
7      glTranslated(0,-h,0);
8
9      //windowサイズ固定
10     glutReshapeWindow(WINDOW_W,WINDOW_H);
11 }
```

4.3.4 マウス, キーボードの入力処理

4.3.5 役の処理

4.3.6 ゲームの進行処理

4.4 hanahuda.h

リスト 8 に hanahuda.h のコードを示す.

リスト 8: hanahuda.h

参考文献

[1] 国立高専機構長野高専,http://www.nagano-nct.ac.jp/ , 閲覧日 2020 年 8 月 5 日