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 [1] 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

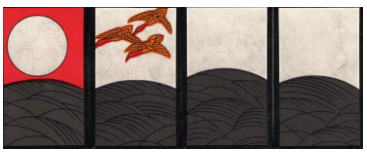

図 8: 8 月 ススキに月・雁

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.

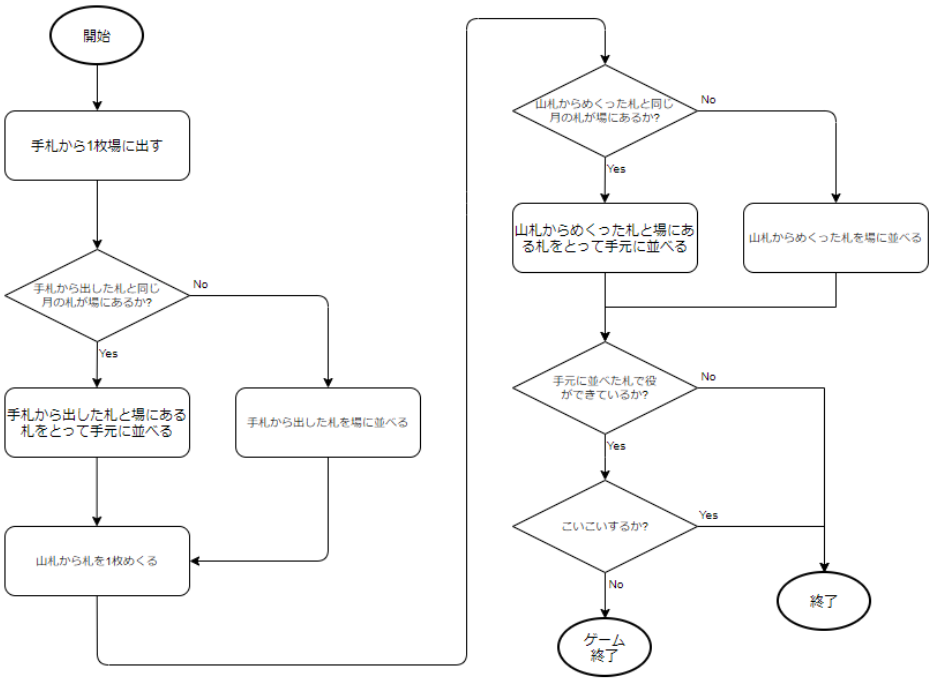

図 13: ターン中の行動手順

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. また, 自分の役が 7 点以上になると得点が 2 倍になるというルールがある. このルールがあるため 12 月戦で点数を稼ぐためには 2 つ以上の役を作って 7 点以上稼ぐことで得点を伸ばすことができる.

- 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. 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 10 行目の glutReshapeWindow 関数でウィンドウサイズを 600x600 に固定している.

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 }
```

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, リスト 8 に示す Timer 関数が実行され, Timer 関数が Display 関数を呼び出す. Display 関数はゲーム全体の流れを管理する処理を行うため 4.3.6 章で説明する.

リスト 8: Timer 関数

```
// タイマーの処理
void Timer(int value){
    glutPostRedisplay(); // 再描画
    glutTimerFunc(100,Timer,0); // 100秒後にタイマーを呼ぶ
}
```

4.3.4 マウス, キーボードの入力処理

マウスの処理について説明する. マウス入力はターン中のプレイヤーが手札から出す札, 場札から取る札を選択するために用いる. リスト 9 にマウスの処理を行う関数を示す. マウスの処理を行う関数は, マウスのクリック入力を処理する Mouse 関数 (2 行目から 56 行目) とウィンドウ内マウスの座標を処理する PassiveMotion 関数 (59 行目から 62 行目) の 2 つである. Mouse 関数が呼び出されると引数として b にマウス入力がされた場所 (右クリックか左クリックか) が入り, s にボタンを押したか離したかの情報が入る. x,y はマウスクリックがされた座標である. 札の選択はマウスが重なっている札を赤枠で表示し, クリックした札を青枠で表示することである. さらにクリックされた手札でとれる札が無ければその札を場札にだし, 手札と場札のクリックされた札の月が一致していれば獲得する処理を行う状態に処理を以降することである. 画面描画と札の出し入れの処理は Display 関数で行い, マウス入力を処理する関数では座標情報からどの札の上にマウスがあるか, どの札がクリックされたかを計算する処理を行う.

リスト 9: マウス入力の処理

```
// マウス入力の処理
void Mouse(int b,int s,int x,int y){
    int tmpclick=-1;
    int tmpclickplace=-1;
    if((turn==0)&&(role==0)){
        if(b==GLUT_LEFT_BUTTON){
            if(s==GLUT_UP){
                tmpclick = calWhichMycard(x,y);
                tmpclickplace = calWhichPlacecard(x,y);
                if(tmpclick!=-1){
                    clickedCard = tmpclick;
                }
                if(tmpclickplace!=-1){
                    clickedPlaceCard=tmpclickplace;
                }
                // 札を場札に捨てる状態のとき
                if((clickedCard!=-1)&&(clickedPlaceCard==-1)){
                    if(isGetCard(clickedCard)==0){
                        status=2;
                    }
                }else if((clickedCard!=-1)&&(clickedPlaceCard!=-1)){// 札を取れる状態のとき
                    if(cards[mycard[clickedCard]].month != cards[place[clickedPlaceCard]].month)
                    {
                        clickedPlaceCard=-1;
                    }else{
                        status=1;
                    }
                }
            }
        }
    }else if((turn==1)&&(role==1)){
        if(b==GLUT_LEFT_BUTTON){
            if(s==GLUT_UP){
                tmpclick = calWhichMycard(x,y);
                tmpclickplace = calWhichPlacecard(x,y);
                if(tmpclick!=-1){
                    clickedCard = tmpclick;
                }
```

```
                if(tmpclickplace!=-1){
                    clickedPlaceCard=tmpclickplace;
                }
                // 札を場札に捨てる状態のとき
                if((clickedCard!=-1)&&(clickedPlaceCard==-1)){
                    if(isGetCard(clickedCard)==0){
                        status=2;
                    }
                }else if((clickedCard!=-1)&&(clickedPlaceCard!=-1)){// 札を取れる状態のとき
                    if(cards[peercard[clickedCard]].month !=
                    cards[place[clickedPlaceCard]].month){
                        clickedPlaceCard=-1;
                    }else{
                        status=1;
                    }
                }
            }
        }
    }
}

// マウスモーション
void PassiveMotion(int x,int y){
    selectedCard = calWhichMycard(x,y);
    selectedPlaceCard = calWhichPlacecard(x,y);
}
```

クリックされた札の判別はリスト 10 に示す関数で行う. 場の札は calWhichPlacecard 関数, 手札は calWhichMycard 関数で判定している. calWhichPlacecard 関数の場合は引数で与えられた座標からどの札の上にマウスがあるか計算する. どの札の上にもマウスがないとき-1, 札の上にマウスがあるときその札の配列 place におけるインデックスを返す. calWhichMycard 関数も同様にどの札の上にもマウスがないとき-1, 札の上にマウスがあるときその札の配列 mycard(親のとき) もしくは配列 peercard(子のとき) におけるインデックスを返す. 札の表示は図 14 に示す札の位置を左上として, 札の横幅 50, 縦幅 80 として判定しているため関数中の数字の参考にしてほしい.

リスト 10: クリックされた札の判別

```
// 場札のどの札の上にマウスがあるか計算する処理
int calWhichPlacecard(int x,int y){
    int index=-1;
    // 上段の判定
    if((220<y)&&(y<300)){
        if((200<=x)&&(x<=500)){
            index=x-200;
            index/=50;
            if(place_num<6){
                if(index>=place_num){
                    index=-1;
                }
            }
        }
        return index;
    }
    // 下段の判定
    if(place_num>6){
        if((300<y)&&(y<380)){
            if((200<=x)&&(x<=500)){
                index=x-200;
                index/=50;
                index+=6;
                if(index>=place_num){
                    index=-1;
                }
            }
        }
    }
```

```
    return index;
}

// 手札のどの札の上にマウスがあるか計算する処理
// -1:手札の上にマウスがない
// 0～7:手札の番号
int calWhichMycard(int x,int y){
    int index;
    if(y<410||y>490){
        index=-1;
    }else if((100<=x)&&(x<=500)){
        index=x-100;
        index/=50;
        if(role==0){
            if(index>=mycard_num){
                index=-1;
            }
        }else{
            if(index>=peercard_num){
                index=-1;
            }
        }
    }else{
        index=-1;
    }
    return index;
}
```

図 14: 札の描画位置

キーボード入力はリスト 11 に示す関数で処理している. ゲームの進行状況を表す status が 4 のときのみキーボード入力を受け付けるようにしている. 入力文字が y のときこいこいを表すためのフラグ mykoikoi もしくは peerkoikoi を 1 にして, 入力文字が n のとき, あがりを表すためフラグを 0 にしている.

リスト 11: キーボード入力の処理

```
// キーボード入力の処理
void Keyboard(unsigned char key,int x,int y){
    if(status==4){
        if(key=='y'){ // こいこいするとき
            if(role==0){
                mykoikoi=1;
            }else{
                peerkoikoi=1;
            }
        }else if(key=='n'){ // こいこいしないとき
            if(role==0){
                mykoikoi=0;
            }else{
                peerkoikoi=0;
            }
        }

    }
}
```

4.3.5 役の処理

役の処理は手札から出した札と山札からめくった札の処理が完了した後で実行される. リスト 12 に示すように役の処理は役ができたかチェックする checkYaku 関数, でき役の得点を計算する calcPoint 関数, 役の表示を行う printYaku 関数の 3 つで構成されている. checkYaku 関数は自分の所持している札の 1 枚ずつについて役に該当するかどうかチェックして役に該当すればフラグを立てる処理を行っている. 役のフラグはまとめて配列 myyaku(親) と配列 peeryaku(子) に格納される. calcPoint 関数は配列 myyaku または配列 peeryaku のフラグをもとに得点を計算する関数である. 役に対する得点は hanahuda.c の配列 pointlist に定義されている. printYaku 関数は calPoint 関数の計算結果をもとに役をコンソールに表示する関数である. またこの関数で相手がこいこいしているときに得点 2 倍や 7 点以上 2 倍の計算を行ってコンソールに表示している.

リスト 12: 役の処理

```
// 役のチェック
void checkYaku(void){
    int i,num;
    int checkkou=0;
    int flgrain=0; // 雨の光札用フラグ
    int hanami=0;
    int tukimi=0;
    int inosikacho=0;
    int akatan=0;
    int aotan=0;
    int tane=0;
    int tan=0;
    int kasu=0;
    if(role==0){
        for(i=0;i<mygetcard_num;i++){
        // 光札の数をカウント
        if(cards[mygetcard[i]].rank==4){
            // 雨かどうか
            if(cards[mygetcard[i]].month==11){
                flgrain=1;
            }
            checkkou++;
```

```
        }
        // 盃をチェック
        if((cards[mygetcard[i]].month==9)&&(cards[mygetcard[i]].rank==3)){
            hanami++;
            tukimi++;
            kasu++;
        }
        // 花見チェック
        if((cards[mygetcard[i]].month==3)&&(cards[mygetcard[i]].rank==4)){
            hanami++;
        }
        // 月見チェック
        if((cards[mygetcard[i]].month==8)&&(cards[mygetcard[i]].rank==4)){
            tukimi++;
        }
        // 猪鹿蝶チェック
        // 猪
        if((cards[mygetcard[i]].month==7)&&(cards[mygetcard[i]].rank==3)){
            inosikacho++;
        }
        // 鹿
        if((cards[mygetcard[i]].month==10)&&(cards[mygetcard[i]].rank==3)){
            inosikacho++;
        }
        // 蝶
        if((cards[mygetcard[i]].month==6)&&(cards[mygetcard[i]].rank==3)){
            inosikacho++;
        }

        // 赤短チェック
        if((cards[mygetcard[i]].month==1)&&(cards[mygetcard[i]].rank==2)){
            akatan++;
        }
        if((cards[mygetcard[i]].month==2)&&(cards[mygetcard[i]].rank==2)){
            akatan++;
        }
        if((cards[mygetcard[i]].month==3)&&(cards[mygetcard[i]].rank==2)){
            akatan++;
        }

        // 青短チェック
        if((cards[mygetcard[i]].month==6)&&(cards[mygetcard[i]].rank==2)){
            aotan++;
        }
        if((cards[mygetcard[i]].month==9)&&(cards[mygetcard[i]].rank==2)){
            aotan++;
        }
        if((cards[mygetcard[i]].month==10)&&(cards[mygetcard[i]].rank==2)){
            aotan++;
        }
        // 取得した札の枚数をカウント
        if(cards[mygetcard[i]].rank==3){
            tane++;
        }
        if(cards[mygetcard[i]].rank==2){
            tan++;
        }
        if(cards[mygetcard[i]].rank==1){
            kasu++;
        }
        }
       // 配列に役を格納
        if(checkkou==5){  // 五光
            myyaku[0]=1;
            myyaku[1]=0; // 四光, 雨四光, 三光を破棄
            myyaku[2]=0;
            myyaku[3]=0;
        }
        if((checkkou==4)&&(flgrain==0)){ // 四光
            myyaku[1]=1;
```

```
            myyaku[2]=0;
            myyaku[3]=0;  // 三光を破棄
        }
        if((checkkou==4)&&(flgrain==1)){ // 雨四光
            myyaku[1]=0;
            myyaku[2]=1;
            myyaku[3]=0;  // 三光を破棄
        }
        if((checkkou==3)&&(flgrain==0)){ myyaku[3]=1; } // 三光
        if(hanami==2){ myyaku[4]=1;} // 花見で一杯
        if(tukimi==2){ myyaku[5]=1;} // 月見で一杯
        if(inosikacho==3){ myyaku[6]=1;} // 猪鹿蝶
        if(akatan==3){ myyaku[7]=1;} // 赤短
        if(aotan==3){ myyaku[8]=1;} // 青短
        if(tane>=5){// タネ
            myyaku[9]=tane-4;
        }else{
            myyaku[9]=0;
        }
        if(tan>=5){ // タン
            myyaku[10]=tan-4;
        }else{
            myyaku[10]=0;
        }
        if(kasu>=10){ // カス
            myyaku[11]=kasu-9;
        }else{
            myyaku[11]=0;
        }
    }else{
        for(i=0;i<peergetcard_num;i++){
        // 光札の数をカウント
        if(cards[peergetcard[i]].rank==4){
            // 雨かどうか
            if(cards[peergetcard[i]].month==11){
                flgrain=1;
            }else{
                checkkou++;
            }
        }
        // 盃をチェック
        if((cards[peergetcard[i]].month==9)&&(cards[peergetcard[i]].rank==3)){
            hanami++;
            tukimi++;
        }
        // 花見チェック
        if((cards[peergetcard[i]].month==3)&&(cards[peergetcard[i]].rank==4)){
            hanami++;
        }
        // 月見チェック
        if((cards[peergetcard[i]].month==8)&&(cards[peergetcard[i]].rank==4)){
            tukimi++;
        }
        // 猪鹿蝶チェック
        // 猪
        if((cards[peergetcard[i]].month==7)&&(cards[peergetcard[i]].rank==3)){
            inosikacho++;
        }
        // 鹿
        if((cards[peergetcard[i]].month==10)&&(cards[peergetcard[i]].rank==3)){
            inosikacho++;
        }
        // 蝶
        if((cards[peergetcard[i]].month==6)&&(cards[peergetcard[i]].rank==3)){
            inosikacho++;
        }

        // 赤短チェック
        if((cards[peergetcard[i]].month==1)&&(cards[peergetcard[i]].rank==2)){
            akatan++;
```

```
163             }
164             if((cards[peergetcard[i]].month==2)&&(cards[peergetcard[i]].rank==2)){
165                 akatan++;
166             }
167             if((cards[peergetcard[i]].month==3)&&(cards[peergetcard[i]].rank==2)){
168                 akatan++;
169             }
170 
171             // 青短チェック
172             if((cards[peergetcard[i]].month==6)&&(cards[peergetcard[i]].rank==2)){
173                 aotan++;
174             }
175             if((cards[peergetcard[i]].month==9)&&(cards[peergetcard[i]].rank==2)){
176                 aotan++;
177             }
178             if((cards[peergetcard[i]].month==10)&&(cards[peergetcard[i]].rank==2)){
179                 aotan++;
180             }
181             // 取得した札の枚数をカウント
182             if(cards[peergetcard[i]].rank==3){
183                 tane++;
184             }
185             if(cards[peergetcard[i]].rank==2){
186                 tan++;
187             }
188             if(cards[peergetcard[i]].rank==1){
189                 kasu++;
190             }
191             }
192             // 配列に役を格納
193             if(checkkou==5){  // 五光
194                 peeryaku[0]=1;
195                 peeryaku[1]=0; // 四光, 雨四光, 三光を破棄
196                 peeryaku[2]=0;
197                 peeryaku[3]=0;
198             }
199             if((checkkou==4)&&(flgrain==0)){ // 四光
200                 peeryaku[1]=1;
201                 peeryaku[2]=0;
202                 peeryaku[3]=0;  // 三光を破棄
203             }
204             if((checkkou==4)&&(flgrain==1)){ // 雨四光
205                 peeryaku[1]=0;
206                 peeryaku[2]=1;
207                 peeryaku[3]=0;  // 三光を破棄
208             }
209             if((checkkou==3)&&(flgrain==0)){ peeryaku[3]=1; } // 三光
210             if(hanami==2){ peeryaku[4]=1;} // 花見で一杯
211             if(tukimi==2){ peeryaku[5]=1;} // 月見で一杯
212             if(inosikacho==3){ peeryaku[6]=1;} // 猪鹿蝶
213             if(akatan==3){ peeryaku[7]=1;} // 赤短
214             if(aotan==3){ peeryaku[8]=1;} // 青短
215             if(tane>=5){// タネ
216                 peeryaku[9]=tane-4;
217             }else{
218                 peeryaku[9]=0;
219             }
220             if(tan>=5){ // タン
221                 peeryaku[10]=tan-4;
222             }else{
223                 peeryaku[10]=0;
224             }
225             if(kasu>=10){ // カス
226                 peeryaku[11]=kasu-9;
227             }else{
228                 peeryaku[11]=0;
229             }
230     }
231 }
232 
```

```
// 得点の計算
int calcPoint(void){
    int point=0;
    int i;
    for(i=0;i<YAKU_NUM;i++){
        if(role==0){
            point+=myyaku[i]*pointlist[i];
        }else{
            point+=peeryaku[i]*pointlist[i];
        }
    }
    return point;
}

// 役の表示
void printYaku(void){
    int i,point;
    printf("---------------\n");
    for(i=0;i<YAKU_NUM;i++){
        if(role==0){
            if(myyaku[i]!=0){
                printf("%s : %d点\n", yakuname[i],pointlist[i]*myyaku[i]);
            }
        }else{
            if(peeryaku[i]!=0){
                printf("%s : %d点\n", yakuname[i],pointlist[i]*peeryaku[i]);
            }
        }
    }
    point=calcPoint();

    if(role==0){
        if(peerkoikoi==1){
            point*=2;
            printf("相手こいこい得点2倍 ×2\n");
        }
    }else{
        if(mykoikoi==1){
            point*=2;
            printf("相手こいこい得点2倍 ×2\n");
        }
    }

    if(point>=7){
        point*=2;
        printf("7点以上得点2倍 ×2\n");
    }

    printf("合計 : %d点\n",point);
}
```

4.3.6 ゲームの進行処理

ゲームの進行処理を行う Display と, ウィンドウに札の描画を行う関数, パケットの送受信について説明する. リスト 13 にウィンドウに札の描画を行う関数を示す. PutSprite 関数は「Springs of C 楽しく身につくプログラミング」[2] 定義されている画像を描画する関数を拡大縮小できるように改良したものである. PaintCards 関数は図 14 の座標に従って札を描画し, マウスと重なっている手札または場札を赤で囲み, 選択されている札を青で囲む処理を行う.

リスト 13: 札の描画を行う関数

```
// (x,y)に大きさscaleの画像を表示
void PutSprite(int num, int x, int y, pngInfo *info,double scale)
{
```

```
    int w, h;  //  テクスチャの幅と高さ

    w = info->Width*scale;   //  テクスチャの幅と高さを取得する
    h = info->Height*scale;

    glPushMatrix();
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, num);
    glColor4ub(255, 255, 255, 255);

    glBegin(GL_QUADS);  //  幅w, 高さhの四角形

    glTexCoord2i(0, 0);
    glVertex2i(x, y);

    glTexCoord2i(0, 1);
    glVertex2i(x, y + h);

    glTexCoord2i(1, 1);
    glVertex2i(x + w, y + h);

    glTexCoord2i(1, 0);
    glVertex2i(x + w, y);

    glEnd();

    glDisable(GL_TEXTURE_2D);
    glPopMatrix();
}

// 札を描画
void PaintCards(void){
    int i,j,num;
    int x,y;
    // 場の札を描画
    x=200;
    y=220;

    // 場を描画
    for(i=0;i<place_num;i++){
        PutSprite(cardimg[place[i]],x,y,&cardinfo[place[i]],0.5);
        x+=50;
        if(i!=0 && i%6==5){
            y+=80;
            x=200;
        }
    }

    // 山札を描画
    PutSprite(cardimg[ALL_CARD-1],100,260,&cardinfo[ALL_CARD-1],0.5);

    // 自分の手札を描画
    x=100;
    y=410;
    if(role==0){
        num=mycard_num;
    }else{
        num=peercard_num;
    }
    for(i=0;i<num;i++){
            if(role==0){
                PutSprite(cardimg[mycard[i]],x,y,&cardinfo[mycard[i]],0.5);
            }else{
                PutSprite(cardimg[peercard[i]],x,y,&cardinfo[peercard[i]],0.5);
            }
        x+=50;
    }

    // 相手の手札を裏面にして描画
    x=100;
```

```
    y=110;
    if(role==0){
        num=peercard_num;
    }else{
        num=mycard_num;
    }
    for(i=0;i<num;i++){
        PutSprite(cardimg[ALL_CARD-1],x,y,&cardinfo[ALL_CARD-1],0.5);
        x+=50;
    }

    y=510;
    x=10;
    // 自分の持ち札を描画
    if(role==0){
        num=mygetcard_num;
    }else{
        num=peergetcard_num;
    }
    for(i=4;i>=1;i--){ // 札の種類別に
        for(j=0;j<num;j++){
            if(role==0){
                if(cards[mygetcard[j]].rank==i){
                    PutSprite(cardimg[mygetcard[j]],x,y,&cardinfo[mygetcard[j]],0.5);
                    x+=20;
                }
            }else{
                if(cards[peergetcard[j]].rank==i){
                    PutSprite(cardimg[peergetcard[j]],x,y,&cardinfo[peergetcard[j]],0.5);
                    x+=20;
                }
            }
        }
        x=10+150*(5-i);
    }

    y=10;
    x=10;
    // 相手の持ち札を描画
    if(role==0){
        num=peergetcard_num;
    }else{
        num=mygetcard_num;
    }
    for(i=4;i>=1;i--){ // 札の種類別に
        for(j=0;j<num;j++){
            if(role==1){
                if(cards[mygetcard[j]].rank==i){
                    PutSprite(cardimg[mygetcard[j]],x,y,&cardinfo[mygetcard[j]],0.5);
                    x+=20;
                }
            }else{
                if(cards[peergetcard[j]].rank==i){
                    PutSprite(cardimg[peergetcard[j]],x,y,&cardinfo[peergetcard[j]],0.5);
                    x+=20;
                }
            }
        }
        x=10+150*(5-i);
    }

    if(status==0){
        if(turn==0){
            if(role==0){
                if(selectedCard!=-1){
                    surroundCard(100+selectedCard*50,410,255,0,0);
                }
                if(clickedCard!=-1){
                    surroundCard(100+clickedCard*50,410,0,0,255);
                }
```

```
                if(selectedPlaceCard!=-1){
                    //下段
                    if(selectedPlaceCard>=6){
                        surroundCard(200+(selectedPlaceCard-6)*50,300,255,0,0);
                    }else{ // 上段
                        surroundCard(200+selectedPlaceCard*50,220,255,0,0);
                    }
                }
                if(clickedPlaceCard!=-1){
                    //下段
                    if(clickedPlaceCard>6){
                        surroundCard(200+(clickedPlaceCard-6)*50,300,0,0,255);
                    }else{ // 上段
                        surroundCard(200+clickedPlaceCard*50,220,0,0,255);
                    }
                }
            }
        }else{
            if(role==1){
                if(selectedCard!=-1){
                    surroundCard(100+selectedCard*50,410,255,0,0);
                }
                if(clickedCard!=-1){
                    surroundCard(100+clickedCard*50,410,0,0,255);
                }
                if(selectedPlaceCard!=-1){
                    //下段
                    if(selectedPlaceCard>=6){
                        surroundCard(200+(selectedPlaceCard-6)*50,300,255,0,0);
                    }else{ // 上段
                        surroundCard(200+selectedPlaceCard*50,220,255,0,0);
                    }
                }
                if(clickedPlaceCard!=-1){
                    //下段
                    if(clickedPlaceCard>=6){
                        surroundCard(200+(clickedPlaceCard-6)*50,300,0,0,255);
                    }else{ // 上段
                        surroundCard(200+clickedPlaceCard*50,220,0,0,255);
                    }
                }
            }
        }
    }
}
```

次に同期通信で送受信するパケットの内容について説明する. サーバー側とクライアント側で同じデッキを共有するためにパケットを送受信する必要がある. 送受信するパケットは int 型の長さ 158 の配列である. パケットの内容は図 15 に示す通りである.

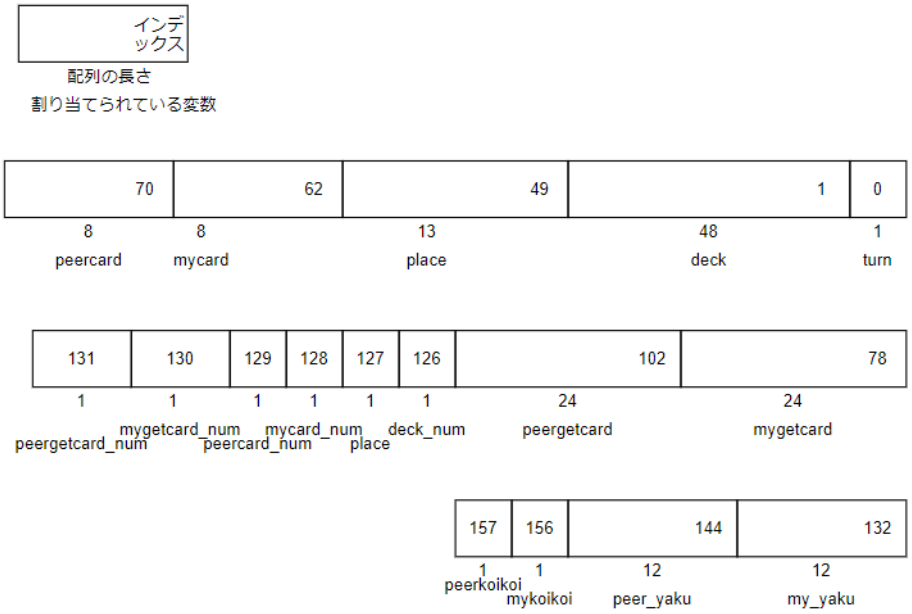

図 15: パケットの内容

図 15 に示したように送受信するパケット data は複数の配列や変数から構成されるため, 値の入出力をまとめて行う関数を作成した. リスト 14 にパケットの送受信の処理を行う関数を示す. setPacket 関数が引数で受け取った data に図 15 に示したパケットの内容をセットする関数, getPacket 関数が受信したパケットの内容を配列や変数に代入する関数である.

リスト 14: パケットの処理

```
// 送信するパケットの設定
void setPacket(int *data){
    int i,j;
    for(i=0;i<17;i++){
        if(i==0){ // turn
            data[packet_sep[i]]=turn;
        }else if(i==1){
            for(j=0;j<CARD_NUM;j++){
                data[packet_sep[i]+j] = deck[j];
            }
        }else if(i==2){ // place
            for(j=0;j<PLACE_MAX;j++){
                data[packet_sep[i]+j] = place[j];
            }
        }else if(i==3){ // mycard
            for(j=0;j<INIT_PLACE;j++){
                data[packet_sep[i]+j] = mycard[j];
            }
        }else if(i==4){ // peercard
            for(j=0;j<INIT_PLACE;j++){
                data[packet_sep[i]+j] = peercard[j];
            }
        }else if(i==5){ // mygetcard
            for(j=0;j<CARD_NUM/2;j++){
                data[packet_sep[i]+j] = mygetcard[j];
            }
        }else if(i==6){
            for(j=0;j<CARD_NUM/2;j++){ // peergetcard
                data[packet_sep[i]+j] = peergetcard[j];
            }
        }else if(i==7){
```

```
            data[packet_sep[i]] = deck_num;
        }else if(i==8){
            data[packet_sep[i]] = place_num;
        }else if(i==9){
            data[packet_sep[i]] = mycard_num;
        }else if(i==10){
            data[packet_sep[i]] = peercard_num;
        }else if(i==11){
            data[packet_sep[i]] = mygetcard_num;
        }else if(i==12){
            data[packet_sep[i]] = peergetcard_num;
        }else if(i==13){
            for(j=0;j<YAKU_NUM;j++){ // myyaku
                data[packet_sep[i]+j] = myyaku[j];
            }
        }else if(i==14){
            for(j=0;j<YAKU_NUM;j++){ // peeryaku
                data[packet_sep[i]+j] = peeryaku[j];
            }
        }else if(i==15){
            data[packet_sep[i]] = mykoikoi;
        }else if(i==16){
            data[packet_sep[i]] = peerkoikoi;
        }
    }
}

// 受信したパケットの解読
void getPacket(int *data){
    int i,j;
    for(i=0;i<17;i++){
        if(i==0){ // turn
            turn = data[packet_sep[i]];
        }else if(i==1){
            for(j=0;j<CARD_NUM;j++){
                deck[j]=data[packet_sep[i]+j];
            }
        }else if(i==2){ // place
            for(j=0;j<PLACE_MAX;j++){
                place[j] = data[packet_sep[i]+j];
            }
        }else if(i==3){ // mycard
            for(j=0;j<INIT_PLACE;j++){
                mycard[j] = data[packet_sep[i]+j];
            }
        }else if(i==4){ // peercard
            for(j=0;j<INIT_PLACE;j++){
                peercard[j] = data[packet_sep[i]+j];
            }
        }else if(i==5){ // mygetcard
            for(j=0;j<CARD_NUM/2;j++){
                mygetcard[j] = data[packet_sep[i]+j];
            }
        }else if(i==6){
            for(j=0;j<CARD_NUM/2;j++){ // peergetcard
                peergetcard[j] = data[packet_sep[i]+j];
            }
        }else if(i==7){
            deck_num = data[packet_sep[i]];
        }else if(i==8){
            place_num = data[packet_sep[i]];
        }else if(i==9){
            mycard_num = data[packet_sep[i]];
        }else if(i==10){
            peercard_num = data[packet_sep[i]];
        }else if(i==11){
            mygetcard_num = data[packet_sep[i]];
        }else if(i==12){
            peergetcard_num = data[packet_sep[i]];
        }else if(i==13){
```

```
102             for(j=0;j<YAKU_NUM;j++){ // myyaku
103                 myyaku[j]=data[packet_sep[i]+j];
104             }
105         }else if(i==14){
106             for(j=0;j<YAKU_NUM;j++){ // peeryaku
107                 peeryaku[j]=data[packet_sep[i]+j];
108             }
109         }else if(i==15){
110             mykoikoi=data[packet_sep[i]];
111         }else if(i==16){
112             peerkoikoi=data[packet_sep[i]];
113         }
114     }
115 }
```

ゲームの進行処理を行う Display 関数について説明する. リスト 15 に Display 関数のプログラムを示す. Display 関数はゲームの進行状況を分ける status という変数によって処理を分けている. status=0 のとき, パケットの送受信を行う処理とマウスの入力を受け付ける処理を行っている. また, パケットの受信結果においてどちらかが上がった場合に status=6(後述) に処理がジャンプする. パケットの送信は write 関数, 受信は read 関数で行っている. どちらの関数も引数にソケット, データ, データ長をもつ. データ長は配列の長さと int 型の 4byte を掛けた数を与えている.

status=0 の状態で手札の札と場の札が選択されるとマウスを扱う関数の処理によって status=1 になる. status=1 の処理は札の獲得と山札のめくりである. 獲得した札を配列に格納した後で, 山札から一枚 pop する. pop した札でとれる場札が無ければその札を場において status=3 にする. とれる札が 1 枚のときはその札を場から pop し, 配列に格納する. とれる札が 2 枚以上あるときは, よりレア度の高い札をとる処理を行う. status=0 の状態で手札の札のみが選択されマウスを扱う関数によって status=2 になっているとき, その札を場に出して山札をめくる処理を行う.

山札をめくる処理が完了すると status=3 の処理が行われる. status=3 の処理は役を判定して画面に表示する処理である. 役を判定する前に, 一つ前の役をとっておき, 役を判定した後で前の役と判定することで新しい役ができたか判定している. もし役ができていれば, status=4, できていなければ status=5 の処理を行う. status=4 のとき画面に図 16 のポップアップを表示してキーボード入力を待つ. キーボード入力に応じてこいこいかあがりかを判定して, status=5 の処理に進む.

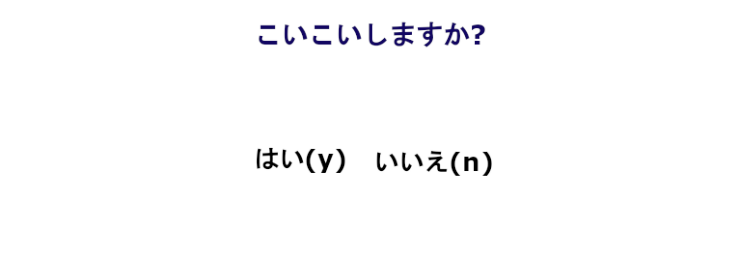

図 16: ポップアップ表示

status=5 のとき, ターン終了処理として, クリック選択を初期化し, turn を切り替える. そしてパケットを送信してこいこいのとき status=0, そうでないとき status=6 の処理を行う. このタイミングで相手にパケットを送信することでターンが入れ替わるのと同時にデータの同期がまとめて行われる. status=6 のとき役と勝敗を表示してゲームを終了する処理を行う. status=5 でパケットを交換しているためターン交代時にどちらかの koikoi フラグが 0 になっていれば printYaku 関数で役を表示して勝敗を表示できる. 最後に status=7, つまり何も処理をしない状態にする. ゲーム終了時に exit 関数を実行しないのは良い役が出来た時にスクリーンショットを取ることができるためである.

リスト 15: Display 関数

```
// メインループ
void Display(void){
    int data[DATA_LENGTH];
    char dispstr[100];
    int r,i,j;
    int tmp;
    int tmpcard;
    int tmpyaku[YAKU_NUM];

    // 描画クリア
    glClear(GL_COLOR_BUFFER_BIT);
    // 札描画
    PaintCards();
    // 札選択
    if(status==0){ // 同期処理と札選択
        if(turn==0){
            if(role==0){
                setPacket(data);
                write(hanahuda_soc,&data[0],4*DATA_LENGTH);
                glColor3ub(0,0,0);
                glRasterPos2i(90,250);
                sprintf(dispstr,"your turn");
                for(i=0;i<strlen(dispstr);i++){
                    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,dispstr[i]);
                }
            }else{
                read(hanahuda_soc,&data[0],4*DATA_LENGTH);
                getPacket(data);
                glColor3ub(0,0,0);
                glRasterPos2i(110,250);
                sprintf(dispstr,"wait");
                for(i=0;i<strlen(dispstr);i++){
                    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,dispstr[i]);
                }
            }
         }else{
            if(turn==1){
                if(role==1){
                    setPacket(data);
                    write(hanahuda_soc,&data[0],4*DATA_LENGTH);
                    glColor3ub(0,0,0);
                    glRasterPos2i(90,250);
                    sprintf(dispstr,"your turn");
                    for(i=0;i<strlen(dispstr);i++){
                        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,dispstr[i]);
                    }
                }
                if(role==0){
                    read(hanahuda_soc,&data[0],4*DATA_LENGTH);
                    getPacket(data);
                    glColor3ub(0,0,0);
                    glRasterPos2i(110,250);
                    sprintf(dispstr,"wait");
                    for(i=0;i<strlen(dispstr);i++){
                        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,dispstr[i]);
                    }
                }
            }
        }
        if((mykoikoi==0)||(peerkoikoi==0)){
            status=6;
        }
        if((mycard_num==0)&&(peercard_num==0)){
            status=6;
        }
    }else if(status==1){ // 手札から出した札で札が取れるとき
        // 場札の獲得
        r = popHandCard(clickedCard);
        pushgetCard(r);
```

```
70          r = popPlace(clickedPlaceCard);
71          pushgetCard(r);
72
73          // 山札から出した札の処理
74          r = popDeck();
75          if(isGetPlace(r)==0){ // 取れる札がないとき
76              pushPlace(r);
77          }else if(isGetPlace(r)==1){ // 取れる札が1枚のとき
78              pushgetCard(r);
79              for(i=0;i<place_num;i++){
80                  if(cards[r].month==cards[place[i]].month){
81                      r=popPlace(i);
82                      break;
83                  }
84              }
85              pushgetCard(r);
86          }else if(isGetPlace(r)>=2){ // 取れる札が2枚のとき
87              tmp=0;
88              for(i=0;i<place_num;i++){ // 得点の高い札を探索
89                  if(cards[place[i]].month==cards[r].month){ // rと月が一緒なら
90                      if(tmp<cards[place[i]].rank){
91                          tmpcard=i;
92                          tmp=cards[place[i]].rank;
93                      }
94                  }
95              }
96              pushgetCard(r);
97              r = popPlace(tmpcard);
98              pushgetCard(r);
99          }
100         status=3; // 役の処理へ
101     }else if(status==2){ // 手札から出した札で札が取れないとき
102         // 手札から1枚だす
103         r = popHandCard(clickedCard);
104         pushPlace(r);
105
106         // 山札から出した札の処理
107         r = popDeck();
108         if(isGetPlace(r)==0){ // 取れる札がないとき
109             pushPlace(r);
110         }else if(isGetPlace(r)==1){ // 取れる札が1枚のとき
111             pushgetCard(r);
112             for(i=0;i<place_num;i++){
113                 if(cards[r].month==cards[place[i]].month){
114                     r=popPlace(i);
115                     break;
116                 }
117             }
118             pushgetCard(r);
119         }else if(isGetPlace(r)>=2){ // 取れる札が2枚のとき
120             tmp=0;
121             for(i=0;i<place_num;i++){ // 得点の高い札を探索
122                 if(cards[place[i]].month==cards[r].month){ // rと月が一緒なら
123                     if(tmp<cards[place[i]].rank){
124                         tmpcard=i;
125                         tmp=cards[place[i]].rank;
126                     }
127                 }
128             }
129             pushgetCard(r);
130             r = popPlace(tmpcard);
131             pushgetCard(r);
132         }
133         status=3; // 役の処理へ
134     }else if(status==3){ // 役の処理
135     // 1つ前の役の和を計算
136     tmp=0;
137     if(role==0){
138         for(i=0;i<YAKU_NUM;i++){
139             tmpyaku[i]=myyaku[i];
```

```
140         }
141     }else{
142         for(i=0;i<YAKU_NUM;i++){
143             tmpyaku[i]=peeryaku[i];
144         }
145     }
146     checkYaku();
147     if(role==0){
148         for(i=0;i<YAKU_NUM;i++){
149             if(tmpyaku[i]!=myyaku[i]){
150                 tmp=1;
151                 break;
152             }
153         }
154     }else{
155         for(i=0;i<YAKU_NUM;i++){
156             if(tmpyaku[i]!=peeryaku[i]){
157                 tmp=1;
158                 break;
159             }
160         }
161     }
162 
163     if(role==0){
164         if(tmp==1){
165             printYaku();
166             mykoikoi=-1;
167             status=4;
168         }else{status=5;}
169     }else{
170         if(tmp==1){
171             printYaku();
172             peerkoikoi=-1;
173             status=4;
174         }else{status=5;}
175     }
176     }else if(status==4){ // こいこいの処理
177         if((turn==0)&&(role==0)){
178             PutSprite(koikoiimg,200,200,&koikoiinfo,1);
179         }else if((turn==1)&&(role==1)){
180             PutSprite(koikoiimg,200,200,&koikoiinfo,1);
181         }
182 
183         if(role==0){
184             if(mykoikoi!=-1){
185                 status=5;
186             }
187         }else{
188             if(peerkoikoi!=-1){
189                 status=5;
190             }
191         }
192     }else if(status==5){ // ターン終了処理
193         selectedCard=-1;
194         clickedCard=-1;
195         selectedPlaceCard=-1;
196         clickedPlaceCard=-1;
197         turn=1-turn;
198         // パケット送信準備
199         setPacket(data);
200         write(hanahuda_soc,data,4*DATA_LENGTH);
201         if(role==0){
202             if(mykoikoi==0){
203                 status=6;
204             }else{
205                 status=0;
206             }
207         }else{
208             if(peerkoikoi==0){
209                 status=6;
```

```
            }else{
                status=0;
            }
        }
    }else if(status==6){ // ゲーム終了処理
        printYaku();
        if((mykoikoi!=0)&&(peerkoikoi!=0)){
            printf("引き分け\n");
        }
        if(mykoikoi==0){
            printf("親の勝ちです\n");
        }
        if(peerkoikoi==0){
            printf("子の勝ちです\n");
        }
        printf("ゲームを終了してください\n");
        status=7;
    }
    // 描画の反映
    glFlush();
}
```

4.4 hanahuda.h

リスト 16 に hanahuda.h のコードを示す.

リスト 16: hanahuda.h

```
#include <sys/types.h>
#include<GL/glut.h>
#include <GL/glpng.h>
#include "mylib.h"

#define PORT (in_port_t)50000 // ポート番号
#define HOSTNAME_LENGTH 64 // ホストネーム長
#define DATA_LENGTH 158 // 同期通信でやりとりするパケット長

#define WINDOW_W 600 // ウィンドウの横の長さ
#define WINDOW_H 600 // ウィンドウの縦の長さ
#define CARD_WIDTH 128 // 札の縦の長さ(読み込み時)
#define CARD_HEIGHT 256 // 札の横の長さ(読み込み時)
#define CARD_NUM 48 // 12*4
#define ALL_CARD 49 // 全札+裏面の札
#define MONTH 12 // 月の数
#define MONTH_CARD 4 // 1つの月の札の数
#define PLACE_MAX 13 // 場における最大札数
#define INIT_PLACE 8 // 最初の場の数
#define SHUFFLE_TIME 1000 // 山札をシャッフルする回数
#define YAKU_NUM 12 // 役の数

static int pointlist[YAKU_NUM] = {10,8,7,5,5,5,5,5,5,1,1,1}; // 役を取った時の得点
static int packet_sep[17] ={0,1,49,62,70,78,102,126,127,128,129,130,131,132,144,156,157};
// パケットの区切り目
static GLuint cardimg[ALL_CARD]; // 札描画のための構造体
static pngInfo cardinfo[ALL_CARD]; // 札描画のための構造体
static GLuint koikoiimg; // こいこいポップアップの画像
static pngInfo koikoiinfo; // こいこいポップアップの画像

static char month[MONTH][4]={"jan","feb","mar","apr","may","jun",
                             "jul","aug","sep","oct","nov","dec"}; // 月の英語表記(略記)
static char yakuname[YAKU_NUM][20] = {"五光","四光","雨四光","三光","花見で一杯","月見で一杯",
                                      "猪鹿蝶","赤短","青短","タネ","タン","カス"}; // 役名

// 札の点数
static int cardrank[MONTH][4] ={{4,2,1,1}, // 松に鶴
                                {3,2,1,1}, // 梅にうぐいす
                                {4,2,1,1}, // 桜に幕
```

```
                                    {3,2,1,1}, // 藤にほととぎす
                                    {3,2,1,1}, // 菖蒲に八ツ橋
                                    {3,2,1,1}, // 牡丹に蝶
                                    {3,2,1,1}, // 萩に猪
                                    {4,3,1,1}, // ススキに月・雁
                                    {3,2,1,1}, // 菊に盃
                                    {3,2,1,1}, // 紅葉に鹿
                                    {4,3,2,1}, // 小野道風にカエル・柳にツバメ
                                    {4,1,1,1}}; // 桐に鳳凰

// 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];

static int deck_num=0; // 次に出る山札配列のindex
static int deck[CARD_NUM]; // 山札
static int place_num=0; // 場に出ている札の数
static int place[PLACE_MAX]; // 場の配列
static int mycard_num=INIT_PLACE; //親の札の数
static int mycard[INIT_PLACE]; //親の札
static int peercard_num=INIT_PLACE; //子の札の数
static int peercard[INIT_PLACE]; //子の札
static int role; // 親(0)か子(1)か
static int turn=0; // 誰のターンか
static int selectedCard=-1; // 選択中の手札の札の番号を保持
static int clickedCard=-1; // クリックした手札の札を保持
static int selectedPlaceCard=-1; // 選択中の場札の札の番号を保持
static int clickedPlaceCard=-1; // クリックした場札の札の番号を保持
static int mygetcard_num=0; // 取得した札の枚数
static int mygetcard[CARD_NUM/2]; // 取得した札の番号の配列
static int peergetcard_num=0; // 子が取得した札の枚数
static int peergetcard[CARD_NUM/2]; // 子が取得した札の番号の配列
static int myyaku[YAKU_NUM]; // 取得した札
static int peeryaku[YAKU_NUM]; // 子が取得した札
static int mykoikoi=-1; // こいこいの状態
static int peerkoikoi=-1; // 子のこいこいの状態
static int hanahuda_soc; // ソケット
static int status=0; // ターンの進行状態

void Reshape(int,int);
void Timer(int);
void PutSprite(int,int,int,pngInfo *,double);
void readImg(void);
void array_init(void);
void card_init(void);
void shuffle(void);
void game_init(int,int);
int popDeck(void);
void pushPlace(int c);
int popPlace(int);
int popHandCard(int);
void pushgetCard(int);
int arrangeCard(void);
void surroundCard(int,int,int,int,int);
void PaintCards(void);
int calWhichPlacecard(int,int);
int calWhichMycard(int,int);
int isGetCard(int);
int isGetPlace(int);
void Keyboard(unsigned char,int,int);
void Mouse(int,int,int,int);
void PassiveMotion(int,int);
void checkYaku(void);
void printYaku(void);
```

```
110 int calcPoint(void);
111 void setPacket(int *data);
112 void getPacket(int *data);
113 void Display(void);
```

5　実行結果

本章では実行結果としてマウスの処理, こいこい時の表示, ゲーム終了時の表示の 3 つについて述べる.

5.1　マウスの処理

5.2　こいこい時の表示

5.3　ゲーム終了時の表示

参考文献

[1] Nintendo,　花 札 の 歴 史・遊 び 方,hhttps://www.nintendo.co.jp/others/hanafuda_kabufuda/howtoplay/index.html , 閲覧日 2021 年 7 月 13 日

[2] 伊藤祥一, ”Spring of C 楽しく身ににつくプログラミング”, 森北出版株式会社,2017 年