計算機言語自由課題レポート

計算実験 405
澤端日華瑠

平成 27 年 2 月 14 日

目 次

1　概要

ブラックジャックのゲームが遊べるプログラム Blackjack.c を作成した. 主な特徴的な機能は

1. ディーラーと対戦してお金を稼ぎ, クリア金額を目指す.
2. カードの減りに合わせて確率が変動する.
3. クリア履歴や途中までの記録を残す事が可能.

である. 構想から機能の実現について説明する.

2　ブラックジャックのルール

ブラックジャックのルールを知らない方向けの簡単なルール解説を載せた. もしルールを知っている場合は読み飛ばして構わない.

2.1　用語解説

以下に出てくる用語を説明しておく.

1. ディール ··· カードを配ること
2. ベット ··· お金を掛けること
3. ヒット (Hit)··· カードをもう一枚引くこと
4. ステイ (Stay)··· カードをもう引かないと宣言すること
5. バースト ··· カードの合計点が 21 を超えること
6. ディーラー ··· 胴元, カードを配り, プレイヤーの対戦相手となる人.

2.2　ルール概説

主なルールを箇条書きにすると

1. ディーラーとカードの合計点で対戦する.
2. 合計点は 21 に近い方が勝ち.
3. 勝てば対戦前に掛けたお金の同額が手に入る.

というものである. 次節以降にてゲームの流れ, カードの数え方を説明しよう.

2.3　ゲームの流れ

ゲームの流れを図 1 にて概説した. このフローチャートを参考に main 関数を設計した.
補足であるが, ディーラーの Hit or Stay の選択はルールによって決まっており, 「15 以下なら Hit,16 以上なら Stay」を必ず選択する.

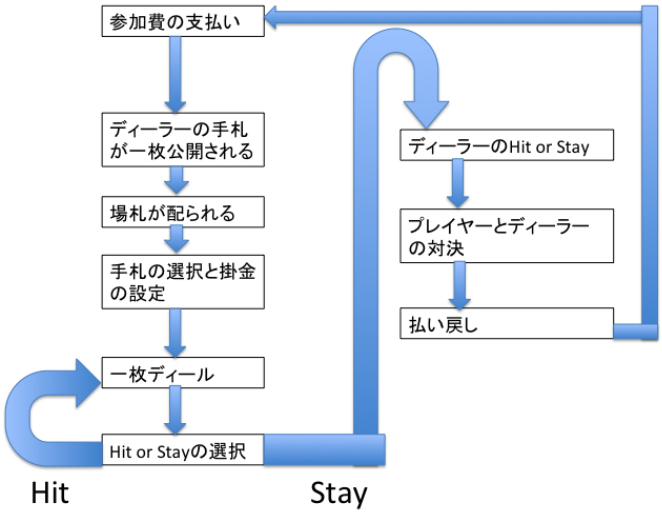

図 1: ゲームの流れ

2.4　カードの合計点の数え方

カードの数え方のルールである.

1. 数札はその数字のまま数える
2. 絵札 (J,Q,K) は 10 と数える
3. A は 1 にも 11 にもなる

出来るだけ 21 に近く,21 を 超えないように数える.

3　機能実装に向けて

特に工夫した機能のみ解説する.

3.1　main 関数の設計

図 1 を参考にメイン関数の設計の為のフローチャートを作った. 図 2 が main 関数の概要である. 詳しくはソースコードを参照して欲しい. カード切れ判断等の細かい部分は省略した.

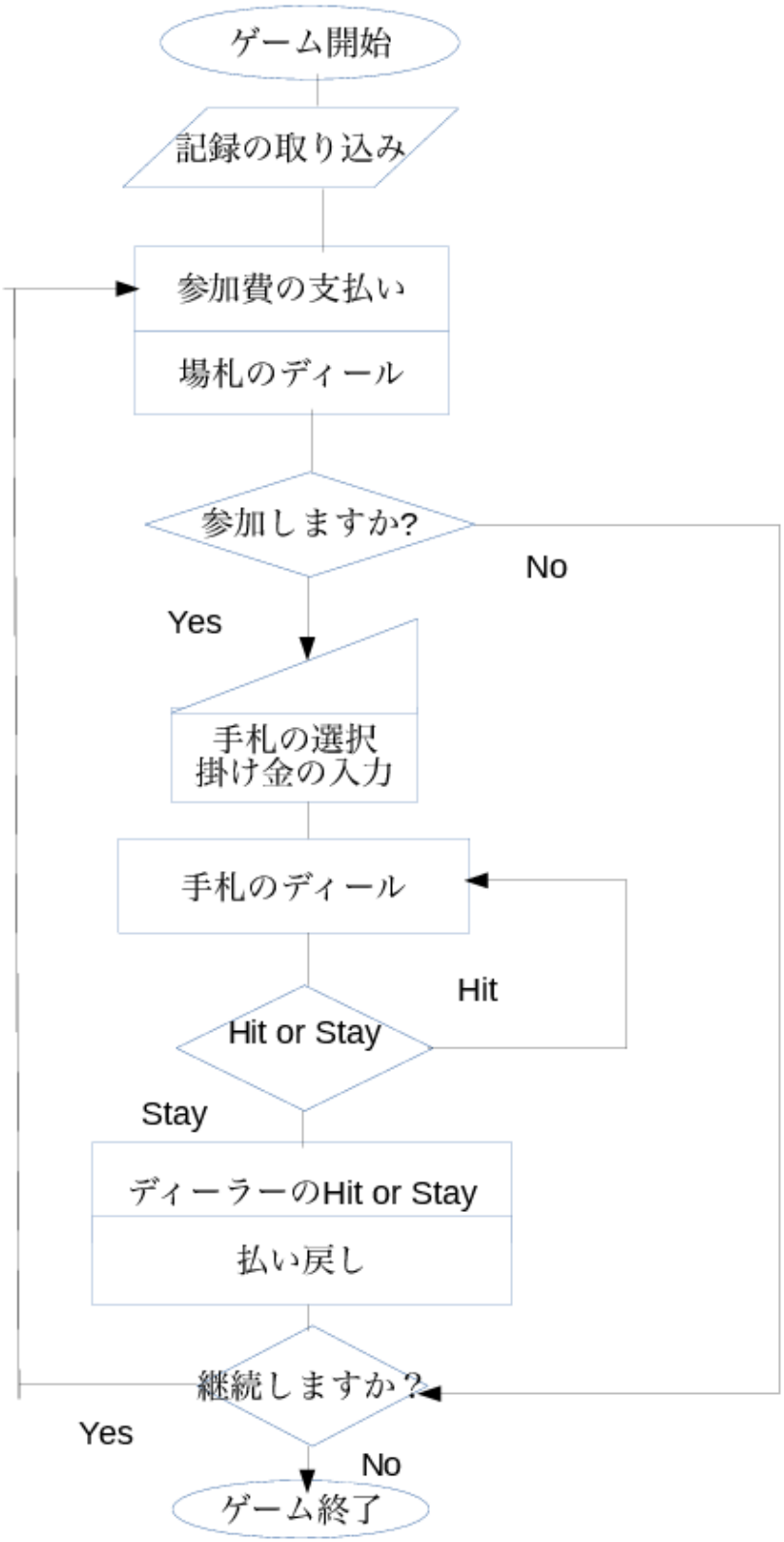

図 2: main 関数フローチャート

3.2　プレイヤー構造体

Blackjack.c ではプレイヤーとディーラーの手札や合計点をコード上で管理し, 制御する必要がある. 変数を徒に定義するとコードが煩雑になるので, 手札, 合計点, 所持金を Player 構造体

プレイヤー構造体

```
typedef struct{
int card[21]; //手札
int number;   //手札の数の合計
int money;    //所持金 (プレイヤーのみ)
} Player;
```

で管理し,

```
Player player,dealer
```

として, プレイヤーとディーラーの情報をコード上で制御する.
例えば, プレイヤーにカードをディールするときは後述のカードシューター関数を利用して,

```
player.card[ i ] = random_card_shooter(cardcount);
```

とすればよいし, ディーラーのカードの合計点を出したいときには, 後述のサムカード関数を利用して

```
dealer.number = sum_card_number(dealer.card)
```

とするだけでよいので分かり易くなる.

3.3　カードシューター関数

カードシューターをプログラム内で再現する.
カードシューターとはカードを入れる箱のことであり, 通常カジノでブラックジャックをプレイする際には, カードを数組使い, 一枚ずつ取り出せるこの箱に入れて一度出したカードを破棄する. こうすることで不正防止, またはゲームの戦略性を向上させている. 今回コード内で再現したかった部分はこの「一度出したカードを破棄する」点である. 通常通り rand 関数を呼び出し,1〜13 までの数字を生成すると同じ数字のカードが始めに箱にセットしたカード以上に出てくる可能性がある. これを防ぐ為にカードシューター関数というこの箱の機構を再現する関数を作成した.cardcount[] は 1〜13 のカードがそれぞれ何枚排出されたかをカウントしておく配列である. この cardcount を頼りに条件を分岐させ, rand 関数が呼び出した数字のカードが尽きていた場合は再帰構造で関数をもう一度呼び出し新たな数字を生成する. これによって今まで出てきたカードの種類によって其れ以降出てくるカードの数字が偏るようにし, ゲームとしての戦略性を向上させた.

カードシューター関数

```
int random_card_shooter(int cardcount[])
{
  int number;
  int i;

  number                = rand() % 13 + 1;
    //カードを引く
  if(
     cardcount[ number  ] < CARD_SET * 4
  && cardcount[ 0       ] < CARD_SET * 52
    )
  {
       cardcount[ number  ] += 1;
       cardcount[ 0       ] += 1;
    return number;
  }
  else if (
       cardcount[ 0       ] < CARD_SET * 52
       )
  {
        //もう一度引く（再帰構造）
       random_card_shooter( cardcount );
  }
  else{
       cardcount[0] = CARD_SET * 52;
       return 0;
  }
}
```

3.4 サムカード関数

節 2.4 で述べたように合計点の数え方のルールは複雑である為, その特殊な足し算を出来る関数を実装した.
まず条件分岐で絵札, 数札を数える. その後一先ず A を 11 として数え上げる. それでバーストした場合は A を 1 として数え直し, バーストしない合計点になるまで引いた.

サムカード関数

```
int sum_card_number(Player player,int hitcount)
{
  int i;
  int sum = 0 ;
  int acecount = 0;
  for( i = 0; i < hitcount + 2; i++)
  {
    //数札はそのまま数える.
    if( player.card[i] != 1
     && player.card[i] != 11
     && player.card[i] != 12
     && player.card[i] != 13)

      sum += player.card[i];

    //絵札は全て 10 で数える.
    else if(   player.card[i] == 11
            || player.card[i] == 12
            || player.card[i] == 13)
      sum += 10;

      //A は 11 で数える.
    else if( player.card[i] == 1 ){
      sum += 11;
      acecount++;
    }
  }
      //バーストした場合の
      //A の数え直し
  if(sum > 21 && acecount >=1 ){
    do{
      sum -= 10;
      acecount--;
    }while( sum > 21 && acecount > 0);
  }

  return sum;
}
```

3.5 記録の管理とクリア履歴のファイル出力

ファイルの入出力を利用してクリアー履歴, プレイ記録が残るようにした. C 言語の教科書に載っているような簡単なコードなので説明は省く.

クリア記録の出力

```
FILE *fpclear;
if ((fpclear = fopen( "Clear.dat", "a+")) == NULL)
printf("記録に失敗しました.\n");
else{
  //クリアー日時
  struct tm *local;
  time_t         t;
  time(&t);
  local = localtime(&t);
  fprintf(fpclear, "%d 年%d 月%d 日%d 時%d 分",
  local->tm_year + 1900, local->tm_mon + 1, local->tm_mday,
  local->tm_hour,   local->tm_min);
  char str[10];
  printf("10 文字以内で名前を入力して下さい:");
  scanf("%s",str);
  fprintf(fpclear,"name:%s %d yen\n",str,player.money);
}
```

所持金の出力

```
if( player.money >= 1 + FIELD_BET && player.money < GOAL_MONEY){
    if ((fp = fopen("kiroku.dat", "w")) == NULL)
  printf("\a ファイルをオープンできません。\n");
    else{
      printf("記録を残しました\n");
      fprintf( fp, "%d" , player.money );
    }
  }
  else if( remove( "kiroku.dat"  ) == 0  ){
          printf( "記録を消しました.\n");
  }
```

4　今後の課題

現在考えている拡張機能は,

1. 掛け金, 手札などの詳細なデータを記録するシミュレータとしての活用.
2. ダブルダウン, スプリット等の特殊なベット方法に対応させる.
3. 名前を入力したらその人の前回のプレイデータを呼び出せる.
4. カードシューター関数を利用して別のカードゲームを実装する.

の 4 つである.

5　プログラム, コメント

ソースコードはメールに添付した. このプログラムは殆どがオリジナルである. よって再配布する場合, 改良する場合は署名を残して欲しい. また, 友人何人かに体験してもらったが, それなりの中毒性があるので注意したいところである.