知能情報工学実験演習 II

C++演習 (Part 2)*

担当: 中村貞吾，下薗真一
TA: 石田竹至，蓑代成功
e-mail: {teigo, t_ishida, minoshiro } @dumbo.ai.kyutech.ac.jp,
sin@ai.kyutech.ac.jp

May, 2015

注意: Part 2 では，グループ単位でプログラムを作成し，レポートを作成，提出すること.

1 グループ演習

ババ抜きで利用したトランプカードを表す型 `Card` とカードの集合の実装である `CardSet` を用いて，トランプ・ゲームの「大富豪（大貧民)」をプレイする環境と思考ルーチンをそなえるプレーヤを作成し，オブジェクト指向言語によるモデル化にもとづくプログラミングとチームでのソフトウェア開発を実践する.

実現するシステム（体系）のよっている規則を完全に把握することが，コンピュータによるシステム構築に不可欠である．まず，実現するゲームのルールを確認しよう.

2 大富豪/大貧民のルール

大富豪は，ラミーや UNO と同様，場札の上に手札（の組）を捨てていき, 手札をなくしたプレーヤーが勝ちとなるタイプのゲームで, 52 枚のカードとジョーカー 1 枚を使用する．以下にルールを簡単に説明するが, 作成するプログラムのモデルを確認するために必要な程度にとどめる.

まず親（ディーラー）は，カードをよくシャフルして，すべてのカードを大富豪から時計回りに一枚ずつ各プレーヤーに配る．配り終えたら，ゲームを親のターンから開始する．ターンが回ってきたプレーヤーは，手札から同じランクのカードを一枚以上，ジョーカーを含め四枚まで組として場に出せる．このカードの組をリードという．ただし，既に場に他人の出したリードがある場合は，同じ枚数のより強いランクの札の組でなければ出すことはできない．リードがペア, 3 カード，あるいは 4 カードならば，次のプレイヤーは同じ枚数でより強いカードの組を出さねばならない．出せない，出したくない場合は, 場に札を出さずにパスすることができる．場にカードを出すか，パスすればそのプレーヤー

*2013 年度版 下薗，嶋田 作

のターンは終了である．ランクはジョーカーが最も強く，続いて 2 > A > K > Q > J > 10 > ··· > 3 の順となる．全員がパスをつづけ，リードを出したプレーヤーにターンが回ってきた場合，リードは流れたとして，最後にカードを出した，つまりリードを出したプレイヤーがリーダーとなる．リーダーは，自由にリードとなる札（の組）を選んで出すことができる．

最初に手札をなくしたプレーヤーが大富豪，つまり 1 位となる．残るプレーヤーも勝ち抜けるまでゲームをつづけ，全員の順位を決める．次のゲームは順位が低い方からターンがまわる．

その他，大富豪には多くのルールがあり，バリエーションやローカルルールとして実際にプレイされなじんでいることも多いと思われる．ただし，今回の演習では時間にも限りがあるため上記に述べた程度にとどめる．次にあげるような点は，採用されないルールによる相違点の例となる．

(1) 最後に出す札には制限なし．ジョーカーや 2 を最後の札としてあがってもよい．

(2) シークエンス (同じスートで複数の連続する数字のカードからなるもの）は，組としてみとめない．

(3) 革命 (4 枚組，あるいは 3 の 4 枚組を出した場合，カードのランクの強さが逆転する．すなわち，強さが 3 ＞ 4 ＞.... の順となる) は起きない．

（4）大富豪と大貧民の間のカードの交換（搾取），および都落ちは，ない．

3　プログラミングのためのモデル：大富豪プレーヤーの箱

プログラミングを行う場合，特にオブジェクト指向言語では，プログラミングする対象を，担う機能や使用する資源にしたがって独立した部分や部品，単位にわけてデータ型（クラス）を設計する．その場合，現実のものにあわせて，あるいは似せて作るのが最も効率的である．

今回，大富豪ゲームを複数プレーヤーが行う状況をプログラムにするにあたっては，次のような考え方で作ることにする：

（1）ディーラーは，プレーヤーの一人（大貧民）が兼ねるのではなく，プレーヤーたちとは独立して中立に，ゲームをルールどおりに進行させる役割を担うものと考える[1]．

すなわち，札を配るだけでなく，ターンなどルールに従った行動をプレーヤーに対してうながす，またプレーヤーが出した札がリードとして通るかどうかを確認し，また他のプレーヤに確認させる，などコミュニケーションを司る役割を負う．ゲームに参加するプレーヤーとその順位の管理も行う．つまり，ディーラーは万能で完全に公平な立場をとるゲームの管理人である．

（2）プレイヤーは，プレイ中ターンの回ってくる参加者である．プログラムにはカードやゲームについての前提知識はなく，大富豪をするためのテーブルもない．そこで，カードのやりとり全てをディーラーに任せた系を考える．プレーヤーは,「カードの出し入れだけができる引き出しがついた箱（に入っている人)」と考えればよいだろう．プレーヤーのターンがまわってくる順番は，ディーラー

[1] 実際のカードゲームにおけるディーラーは，大貧民もしくはほかの誰かが行うが，このモデルではゲーム管理を行うクラスであり，実際にプレイするプレイヤーではないことに注意．

がどの箱の引き出しをさわるかで決まる．カードの配布は，ディーラーが引き出しにカードを入れることで行われる．ディーラーはターンがきたプレーヤーの引き出しにリードを入れ，札を出すかパスするかを促す．引き出しにリードとより強い組が入っていたら，新たなリードとして受け入れ，他のプレーヤーの引き出しに入れて見せる．そうでなければ，リードを取り出し，その他のカードはそのままにして引き出しを押し返す．そして，プレーヤーの手札がなくなったら，勝ち抜けを記録し，次のゲームでの順番とするのである．

人間同士でゲームをする場合，これらの「手順」にかわるものは，言葉や視覚によって暗黙のうちに完成している．しかし，機械的なゲームの進行を実現するには，これらを限定的で明確なものにしなければならない．たとえばネットワーク対戦を考えれば，プレーヤーが人であっても，機械であっても，その必要があることを理解できるだろう．

4 基本となるソースコードと拡張の方法

ゲームを行う環境となる，Dealer.h/cpp, Player.h/cpp, GameState.h および main.cpp は, 変更を加えずにそのまま使用する．クラス継承のしくみを使って，Player クラスを継承した別のプレーヤー用クラスを作り，ゲームの進行のために呼び出される関数を再定義する．継承したクラスで新たに必要となる変数や関数も用意して，そのまま既存の Dealer クラス等とともにコンパイル，実行する．

グループでは，プレーヤーをシミュレートする Player の拡張クラスを作り，カード配付後のゲーム開始直前に呼び出される ready(), ターンがきた時に呼び出される follow( ), および他のプレーヤーが捨てた札の確認を求められる approve の 3 つの関数（のうち必要なもの）を作成する．継承による拡張のサンプルとして LittleThinkPlayer がある．Player には手札をランダムに出す follow 関数が実装されており，動作の確認は可能である．これを LittleThinkPlayer のように継承し，新たなプレーヤークラスで必要なデータ構造や手続きを作成，follow( ) 関数などを書き換えるなどして，グループ独自のより強いプレーヤーを作るのが目的となる．

4.1 follow( ) 関数

プレイヤーがまわってきたターンでカードを出すアクションは，Player::follow() のオーバーライド（定義の上書き）によって行う．引数は，場札 (CardSet) pile フィールドなどを持つゲーム状況をあらわす構造体 GameState 型データと，プレーヤーが出す札を入れるカードの集合で，返値は常に true である．

基底クラスでありサンプルにもなっている Player クラスでは，

```
bool Player::follow(const GameState & gstat, CardSet & s) {
    CardSet pile(gstat.pile);   // 作業用に場札のカードセットをコピー
    Card tmp;
    s.makeempty();              // 念のためカラにする
    hand.pickup(&tmp, -1) );    // 手札からランダムに一枚選ぶ
    s.insert(tmp);              // 結局いいかげんに一枚出してみるだけ
```

```
        return true;                    // 常に true を返す
    }
```

となっている．つまり場のカードについては考慮せず，自分の手持ちのカードセットからランダムに 1 枚出す，という動作を行っている．

4.2 GameState 型

場札をあらわす CardSet データのほか，プレーヤーの並び，各プレーヤーの手札の枚数など，プレーヤーがプレイ中に得られる情報をまとめた構造体である．各フィールドの名称と意味は，以下のようになっている．

```
struct GameState {
    ...
CardSet pile;         // 場札
int nofPlayers;       // プレーヤー数
int inTurn;           // ターンのまわってきたプレーヤー（プレー順での添字）
int leader;           // 現在のリードを出したプレーヤー
int nofCards[MAXIMUM_NUM_OF_PLAYERS];    // 各プレーヤーの手札の枚数
int playerIDs[MAXIMUM_NUM_OF_PLAYERS];   // 各プレーヤーの ID 番号
    ...
};
```

場に札を出す上で最も重要なのは，pile フィールドであろう．プレーヤーを表す整数は，プレー順でならんだプレーヤーの配列の添字である．同じプレーヤーに対しても，ゲームが終了し順番がかわるとかわってしまう．異なるゲームの間で同じプレーヤーを追跡したい場合は，プレーヤーをあつめた時点でディーラーが割り振った ID を確認する．

4.3 approve( ) 関数

他人が場に捨てたカードセットなどの状況を確認する機会をプレーヤーにあたえるため，Player::approve() 関数が呼び出される．Player クラスでは何も行っていない．引数は，GameStatus 型データである．

4.4 ready( ) 関数

ready( ) 関数は，はじめてのゲームで，あるいは 1 ゲームが終了し新しいゲームが始まる際に，一度だけ呼び出される．場に出た強いカードを記憶している場合など，ゲームごとに変数の初期化が必要ならば，この関数の中で行う．

4.5 注意点

基本的に実装を考えるのは，ready(), follow( ), approve( ) の 3 関数のみである．上記以外の Player クラスで定義ずみの関数は，変更をしてはならない．もちろん，必要な機能や操作を自分の拡張クラスに関数や変数として追加し，上記 3 関数で使用する，といったことは自由に行ってよい．

プレイヤーが出したカードが現在の場に出ているカードと比較して受理できるかなどの判定は，Dealer が行うので考えなくて良い[2]．また，パスをしたい場合は，空のセットもしくは通らないカード (現在，場に出ているカードよりも弱いカード) を follow() に渡せばよい．通らないカードセットは，Dealer によって手持ち札に戻される．

演習 1 拡張クラスでオーバーライドして，follow()，また必要があれば approve( ), ready( ) を改良もしくは実装し，より知的な戦略に基づくプレーヤーを実現せよ．

ただし，提供される Card.h/cpp, CardSet.h/cpp, Dealer.h/cpp, Player.h/cpp, GameStatus.h, および main.cpp のゲーム進行ルーチン部分は，変更を加えずにそのまま使用できること．

5 演習を進める上での注意

リーダーを中心にグループ内で事前に役割分担を決め，プログラムを作成すること．演習では，まず，何らかの戦略に基づくプレイヤーを複数実装する．例えば，現在は 1 枚しか出せないが，複数のカード (ペアや 3 カードなど) を出せるようにしたり，選んだカードが場に通るかどうかを判定して出すなどが考えられる．それぞれを対戦させたり，考察することで，より洗練されたプレイヤーに改良するプロセスを繰り返していく．

継承による拡張のサンプルとして，LittleThinkPlayer を用意している．最終的には，各班で Group 班番号 という名前の新たなサブクラス（例えば 1 班なら，Group1）を作成し，大会に参加させる．

```
class LittleThinkPlayer : public Player {
// LittleThinkPlayer は Player のサブクラス (派生クラス)
    CardSet memory;
public:
    LittleThinkPlayer(const char *);
    bool follow(CardSet &, CardSet &);
    bool approve(CardSet &, int[]);
};
```

班内でさまざまなアイディアを試すには，戦略ごとにサブクラスを定義して対戦させればよい．main.cpp 関数で，以下のように参加させることができる．

```
d.regist(new Player(“ Erika ”));
  // Player.cpp の follow() や approve()
d.regist(new LittleThinkPlayer("Warabi"));
  // LittleThinkPlayer.cpp の follow() や approve()
```

実験の最後の時間に各班で作成した思考ルーチンを対戦させる．具体的には，ランダムに割り振られた予選リーグを行い，各予選リーグで 1 位の班で決勝リーグを構成する．この順位は最終的な成績に加味する．

[2] ゲームとしての判定は Dealer が行うが，実際問題としては，場に出そうとしているカードが通るのかを Player も判定した方が良い．それを考慮せず，通らないカードを出してしまうと，パスしたという扱いになり，不利になる．

6　レポートについて

レポートの作成には，全員が参加すること．以下の書式・期日で各班ごとに 1 つ提出すること．

1. A4 のレポート用紙を用い，左上部をステープラでとめる．その他，Part 1 でのレポートに準ずる．
2. 表紙には，演習名 (知能情報工学実験演習 II C++演習：グループ演習)，全班員の学生番号と名前，提出日を記載すること．
3. 各班員の役割分担や貢献内容，レポートの分担を具体的に書くこと．最終的に採用されなかった戦略なども含めて書いてよい．
4. 班で作成したプログラムのリスト (ただし，最終的に実装したものの必要部分のみでよい)．
5. 実装した思考ルーチンの説明 (工夫した点および不完全な点を含む)
6. 実行結果 (主要な部分・レポートでの説明に必要な部分のみでよい)
7. 全体の考察[3] とまとめ

レポートは，このテーマの実験最終授業日の翌週の同一曜日までに，研究棟 6 階のレポートボックスに提出すること．不備のあるレポートは要再提出とし，知能情報工学科掲示板（研究棟東棟６Ｆ）に却下の掲示をする．

[3] 感想ではなく考察をすること．