知能情報工学実験演習 II

C++演習 (Part 2)

担当: 下薗真一・嶋田和孝 (TA: 香野/小松)
E-mail: sin@ai.kyutech.ac.jp,
{shimada,d_kouno,k_komatsu}@pluto.ai.kyutech.ac.jp

注意: Part 2 では実験はグループ単位でプログラムを作成すること.

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.

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 $2 > \mathrm{A} > \mathrm{K} > \mathrm{Q} > \mathrm{J} > 10 > \cdots > 3$ 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.

その他にも大富豪には多くのルールがバリエーションとして存在する．ただし，今回の演習では時間にも限りがあるため,（1） ジョーカーや 2 を最後の札としてあがってもよいこととし,（2） シー

クエンス[1] や（３） 革命[2] ,（４）大富豪と大貧民の間のカードの交換（搾取）と，都落ちのルールは，使用しないものとする．

プログラミングのためのモデル

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）を作らなければならない．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た，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[3] ．

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，それぞれがターンの回ってくる参加者である．人間がカードゲームをする場合とは異なり，プログラムにはあらかじめカードを識別する能力はなく，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ものを考える．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「コミュニケーション手順」は言葉や視覚によって暗黙のうちに完成する．しかし，機械的なゲームの進行を実現するには，これらの作業を明確にしなければならない．プレーヤーが人であっても，機械であっても，たとえばネットワーク対戦を考えれば，その必要性が容易に理解できるだろう．

サンプルプログラム

ゲームを行う環境をつくるためのプログラム，`Dealer.h` および `Dealer.cc` , `main.cc` は, 変更を加えずにそのまま使用する．グループで作成するのは，プレーヤーをシミュレートするクラス `Player` に関する部分のみである．

サンプルプログラムとして，`Player.h` および `Player.cc` が用意した．このクラスのデータ構造や手続きを拡張，整備して，より強いプレーヤーを作るのが目的である．クラス継承のしくみを使え

[1] 同じスーツで 3 枚以上の連続する数字

[2] 4 カードを出した場合は，強さが逆転する．すなわち，強さが 3 ＞ 4 ＞.... の順となる．

[3] 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．

ば，Player クラスを継承した別のプレーヤー用クラスを作り，新たに必要な変数と関数の作成もしくは変更だけを行えば，そのまま既存の Dealer クラスとともに使用できる．

プレイヤーが，ターンにおいてどのカードを出すかは Player::follow() によって決定される．サンプルプログラムにおいては，

```
void Player::follow(const CardSet & tbl, CardSet & s) {
    Card tmp;
    s.makeempty();
    hand.pickup(&tmp, -1) );
    s.insert(tmp);
    return;
}
```

となっている．Player::follow() は引数として場のカードと自分のカードが定義されているが，サンプルプログラムでは，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 枚引き抜くという最もナイーブな戦略に基づいている．

また，現在は何も行っていないが，自分の番以外の時に他人が場に捨てたカードセットなどの状況を確認する関数として Player::approve() というものも考える．

実際の実装で考えなければならないのは，この 2 つの関数のみである．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[4] ．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．その場合，通らないカードセットは自動的に手持ち札に戻される．

演習 1 Player::follow() と必要であれば Player::approve() を改良し，より知的な戦略に基づく Player を実現せよ．

演習を進める上での注意

演習に際しては，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．

基本的に Player::follow() と Player::approve() 以外の関数を変更しないこと．この 2 つの関数を拡張する場合，使用して良いのは Card と CardSet のメンバ関数である[5] ．複数のプレイヤーを実装する方法はいくつか考えられるが，継承によりサブクラスを作成すること．そのためのサンプルとして，LittleThinkPlayer を用意している．最終的には，各班の名前で新たなサブクラス（例えば 1 班なら，Group1）を作成すること．

[4] 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．

[5] 他の班と対戦する際に，改良した Player::follow() と Player::approve() をコピーすれば動作するように心がけて作ること． Player 内に必要に応じて新しい関数を作っても良いが，あくまで， 他のクラスや Player 内に既に定義されているメンバ関数などに影響しないようにすること．

```
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[]);
};
```

この方法に基づき，戦略ごとに適当なサブクラスを定義すればよい．そして，main 関数で，以下のように使い分ければよい．

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

2 レポートについて

レポートは以下の書式・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，上部をステープラでとめること．
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)，班員の学生番号と名前，提出日を明記すること．
3. 各班員の役割分担 (例えば，各繰り返しの段階で実装した戦略や作業などを箇条書きする程度でよい[6])
4. プログラムリスト (ただし，最終的に実装した部分のみでよい)
5. 実装した戦略ルーチンの考え方 (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[7] とまとめ

レポートは，このテーマの実験最終日の翌週の同一曜日までに，研究棟 7 階 (E716) の嶋田のもとまで提出すること．レポートは評価され，不備のあるものは〆切から 1 週間以内に知能の掲示板に再レポートの掲示をするので，注意しておくこと．

班ごとのレポートとは別に，個人単位で A4 サイズ 1 枚の個人評価票 (書式指定) も提出すること．

[6] ただし,「みんなで頑張った.」などの抽象的な記述ではなく，誰が何を担当したか具体的に書くこと．最終的に採用されなかった戦略などもこの中に加えて良い．

[7] 感想ではなく考察をすること.「面白かった.」や「難しかった.」だけではダメ.