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: 佐藤 誠 / 古野 雅大)

{teigo, m_satou, furuno}@dumbo.ai.kyutech.ac.jp

sin@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完成させること.

1 目的

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)」をプレイする思考ルーチンを作成し，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あがりとなるタイプのゲームである[1]．52 枚のカードと多くの場合ジョーカー 1 枚を使用する.

まず親（ディーラー）は，カードをよくシャフルし，カードを一枚ずつ大富豪から時計回りに各プレーヤーに配る．すべてを配り終えたら，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を組として，場に出せる．このカードの組をリードという．ただし，既に場にリードがある場合は，同じ枚数のより強いランク（数）の札の組でなければ出すことはできない．出せない，出したくない場合は, 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して，プレーヤーのターンは終了する．ランクは，ジョーカーが最も強く, 続いて $2 > \mathrm{A} > \mathrm{K} > \mathrm{Q} > \mathrm{J} > 10 > \cdots > 3$ となる．リードを出したプレーヤーの後全員がパスをつづけ，そのプレーヤーに再びターンが回ってきた場合は，場は流れてリードはなくなったことになる．そして，そのリードを出したプレイヤーが新たな親（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.

[1]参考： 草場純著，“夢中になる！トランプの本,” 主婦の友社, 2007.

これを繰り返して，最初に手札をなくしてあがったプレーヤーが大富豪，つまり 1 位となる．残るプレーヤーが勝ち抜け，全員の順位が決まるまでゲームを続ける．

その他にもバリエーションとして多くのルールがあるが，今回の演習では時間に限りがあり思考ルーチンに反映できるルール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[2]や（3） 革命[3]，（4）大富豪と大貧民の間のカードの交換（搾取）と都落ち，といったポピュラーなルールも使用しないものとする．

2 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，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としてクラスを設計する．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，大富豪ゲームをプログラムにするにあたっては，次のような考え方を採用している．

(1) ディーラーは，プレーヤーたちとは独立したゲームの進行役と考える[4]．

すなわち札を配るだけでなく，ターンをすすめる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，他のプレーヤに見せる，などのコミュニケーションを司る．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，手札を持っている限りターンが回ってくる「箱」である．

人間が実際にカードゲームをする場合とは異なり，プログラムではカードに表裏はなく，テーブルもない．そこで，カードのやりとりもディーラーに任せたものを考える．プレーヤーは，「カードの出し入れのみ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．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．プレーヤーは，ターンがまわってきたことを，ディーラーが箱の引き出しを動かしたことで知る．ディーラーは，プレーヤーの引き出しにリードの写しを入れ，札を出すかパスするかを促し，引き出しにより強い組が入れられたなら新たなリードとして認め，他のプレーヤーに見せる．そうでなければ，入れられたカードを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，順位とする．

人間同士でゲームをする場合は，これらのやりとりはカードという道具をつかって暗黙のうちに行われる．しかし，機械的なゲームの進行を実現するには，上記のような作業を考えなければならない．ネットワーク対戦を考えれば，その必要性が理解できるだろう．

次に，これらがどのようにプログラム化されており，思考ルーチンを作成するために何をプログラミングすればよいかを説明する．

[2]同じスートで 3 枚以上の連続する数字

[3]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．

[4]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．

3　メイン関数プログラムとサンプルクラス

カードゲームを作成するための Card.h/.cc, CardSet.h/cc は，Part 1 で使用したものとほぼ同じだがこの Part 2 用のものが用意されている．ゲームのシミュレーションとプレイ環境のための Dealer.h/cc および main.cc も用意されている．また，これらとともに使用して，プレーヤーをシミュレートするクラス Player も Player.h/cc として用意されている．これらは，ファイルをコピーしてそのまま使用する．

グループで行うことは，プレーヤーをシミュレートするクラス Player の (1) リードに対して札を出す，あるいは出さないという行動をとるための関数 Player::follow() を改良し，より高い順位であがれるようにすること，また必要であれば，(2) 他のプレーヤーが新たなリードを出した際に，そのリードを見せるため呼ばれる Player::approve() に，場札を情報として活用できるようなプログラムを組むことである．

これらの拡張や改良を加える方法の参考として，Player クラスを基底クラスとして継承した拡張クラス LittleThinkPlayer を LittleThinkPlayer.h/.cc として用意した．このクラス名をグループ名称に変更し，必要なデータ構造や手続きをくわえ，上記の二つの関数を拡張，整備して，より強いプレーヤーを作るのが目的である．

follow 関数

ターンにおいてどのカードを出すかを決める follow() 関数は，Player クラスで

```
virtual bool follow(CardSet & pile, CardSet & cards);
```

と定義されている．(virtual 修飾子は，この関数が拡張クラスで上書きされる可能性があることを示す.）第一引数 pile は場にでているリード（のコピー）を格納した CardSet 型変数（への参照)，第二引数 cards が自分の出す札（の組）を入れディーラーに渡すための CardSet 型変数 (への参照）で，この関数に返値はない．

Player では，場のカードを考慮せず，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 (pickup を第二引数 −1 でコール) という動作になっている．ターンがまわってきたプレーヤーは，この follow() 関数が呼ばれるので，カードを出す作業をおこなって，出す札を第二引数にいれ，また自分の手札から削除して関数を終了する．

この関数の入出力仕様は変更せず，関数内の動作のみを変更し,「強い」プレーヤーにする．プレイヤーが出したカードがリードになるかどうかの判定は， Dealer が行うので考えなくてもよい[5]．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) を渡せばよい．その場合，通らないカードセットはディーラーによって手持ち札に戻される．

[5]ゲーム進行上の判定はディーラーが行うが，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 自身が判断しなければ，通らないカードを出しパス扱いになり，不利になる．

approve 関数

また，自分以外のプレーヤーのターンで他人が場に出したカード組を知りたいプレーヤーのために，関数 Player::approve() が呼び出される．

```
virtual bool approve(CardSet & pile, int cardnum[] );
```

第一引数 pile はディーラーが認めたリード（のコピー）を格納した CardSet 型変数（への参照），第二引数 cardnum は他のプレーヤーの手持ち札をディーラーが知らせるための整数配列で，添字は各プレーヤーの ID 番号（0 からプレーヤー数 −1 まで），値はそのプレーヤーの手札の枚数である．プレーヤーは自分の ID を getId 関数で得られる．他のプレーヤーは，値が Player::NO_MORE_PLAYERS (= −1) となっている直前の添字までで自分の ID でないもの，ということになる．この関数は，確認したという意味で bool 型で true (非 0 値) を返す．ただし，現在その返値は一切使用されていない．この関数は，たとえば自分の札が切り札（トランプ）として通用するかどうか，場に捨てられた強い札を見ておぼえておく，等のために活用できる．

実装で考えなければならないのは，この 2 つの関数の動作のみである．

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．
実際の改良は，LittleThinkPlayer クラスなどを参考にして，直接 Player クラスを変更せず，自分たちのプレーヤークラスを作ることで行うこと．当然だが，Player クラスに用意してある他の関数を使用したり，クラスに新しくメンバ変数や関数を追加して使う，呼び出す，等は自由にやってよい．

4 演習を進める上でのアドバイス

演習は，リーダーを中心にグループ内で役割分担をし，プログラムを作成すること．

例えば，まず，何らかの戦略に基づくプレイヤーを各メンバーがアイディアを出す，あるいは実装する．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．そして，スケジュールを，それぞれを対戦させたり，アイディアを討論して，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．

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．またその際には，コンストラクタを参考にデフォルトの名前もつけておくこと．

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．

5 レポートについて

レポートは，Part 1 と同じ要領で，各班ごとに 1 つ提出する．

班ごとのレポートとは別に，全員に A4 サイズ 1 枚の個人評価票 (書式指定) も提出させる．