19　トランプ (2)

19.1　マウスを使う

前回の例では，あらかじめ決められた札（ハート全部）しか裏返すことができません。マウスで札を選べるようにしたいのですが，それにはマウスのイベントハンドラーが必要です。コンポーネントを設計時にフォームに置く場合は，オブジェクトインスペクターのイベントの欄をダブルクリックすると，必要なことが自動的に書かれるので苦労無しでしたが，実行時に置く場合は自分で書かないといけません。

また，イベントハンドラーを書くユニット ExampleU と，コンポーネント（TCard）を生成するユニット TrumpU が別なので，両者の間で情報を交換するための工夫が必要です。

19.1.1　TrumpU の変更点

TrumpU が表示されてないときは,［表示｜ユニットの表示］をして TrumpU を選択して表示させて，変更してください。

(1) イベントハンドラーの型を宣言する。

イベントハンドラーは，イベントによって引数の個数や型が異なります。その引数の情報をイベントハンドラーの型として宣言しておきます。

```
type
  TMouseUpDown = procedure (Sender: TObject; Button: TMouseButton;
                            Shift: TShiftState; X,Y: Integer)
                 of object;
  TMouseMove   = procedure (Sender: TObject;
                            Shift: TShiftState; X,Y: Integer)
                 of object;

  TSuit   = (stClub,stDiamond,stHeart,stSpade,stOther);
  TRank   = 1..13;
```

(2) TTrump のメソッドを追加。

```
  end; { Sakusei }

procedure TTrump.MouseHandler(OnDown : TMouseUpDown;
                              OnUp   : TMouseUpDown;
                              OnMove : TMouseMove);
          (* Mouse のイベントハンドラーを定義する *)
  var
    CardNo: TCardNo;
  begin
    for CardNo := 0 to 51 do
      with Cards[CardNo] do
        begin
          OnMouseDown := OnDown;   // MouseDown したときのハンドラー
```

```
        OnMouseUp   := OnUp;     // MouseUp   したときのハンドラー
        OnMouseMove := OnMove;   // MouseMove したときのハンドラー
      end;
  end; {MouseHandler}
```

19.1.2 ExampleU の変更点

(1) イベントハンドラーを宣言する。

設計時に置いたコンポーネントならば，オブジェクトインスペクターのイベント欄をダブルクリックすると自動生成される部分ですが，実行時に生成するコンポーネントの場合は自分で書かないといけません。

```
  procedure ButtonCloseClick(Sender: TObject);
  procedure ButtonJikkouClick(Sender: TObject);
  procedure ButtonSakuseiClick(Sender: TObject);
  procedure CardMouseDown(Sender: TObject; Button: TMouseButton;
                          Shift: TShiftState; X,Y: Integer);
  procedure CardMouseUp(Sender: TObject; Button: TMouseButton;
                        Shift: TShiftState; X,Y: Integer);
  procedure CardMouseMove(Sender: TObject;
                          Shift: TShiftState; X,Y: Integer);
```

(2) 実現部に骨格を書く。

これも今までは自動作成された部分です。

```
(********** イベントハンドラー **********)
procedure TFormMain.CardMouseDown(Sender: TObject; Button: TMouseButton;
                                  Shift: TShiftState; X, Y: Integer);
begin
end; \{CardMouseDown\}

procedure TFormMain.CardMouseUp(Sender: TObject; Button: TMouseButton;
                                Shift: TShiftState; X, Y: Integer);
begin
end; \{CardMouseUp\}

procedure TFormMain.CardMouseMove(Sender: TObject;
                                  Shift: TShiftState; X, Y: Integer);
begin
end; \{CardMouseMove\}
```

(3) トランプを作成したらマウスハンドラーを設定する。

```
procedure TFormMain.ButtonSakuseiClick(Sender: TObject);
begin
  Trump.Sakusei(Self);
```

```
  Trump.MouseHandler(CardMouseDown,CardMouseUp,CardMouseMove);
  ButtonSakusei.Visible := False;     // 作成は 1 度だけ
  ButtonJikkou.Enabled := True;
end; {ButtonSakuseiClick}
```

これで準備はＯＫです。
翻訳してエラーがないことを確認してください。

19.1.3 OnMouseDown ハンドラーのテスト

カードをクリックしたら，裏返したり反転したりするようにしてみましょう。

(1) CardMouseDown の中身を書く。

```
procedure TFormMain.CardMouseDown(Sender: TObject; Button: TMouseButton;
                                  Shift: TShiftState; X, Y: Integer);
begin
  with TCard(Sender) do                          // クリックしたカード
    begin
      if ssShift in Shift                        // Shift キーを押しながらクリック
        then begin
               case Button of
                 mbLeft  : OmoteNiSuru;
                 mbRight : UraNiSuru;
               end;
             end;
      if ssAlt in Shift                          // Alt キーを押しながらクリック
        then begin
               case Button of
                 mbLeft  : PosiNiSuru;
                 mbRight : NegaNiSuru;
               end;
             end;
      if ssCtrl in Shift                         // Ctrl キーを押しながらクリック
        then begin
               case Button of
                 mbLeft  : BringToFront;
                 mbRight : SendToBack;
               end;
             end;
    end;
end; {CardMouseDown}
```

(2) 実行

同じマウスボタンの操作（たとえば，“裏にする” と “ネガにする”）は，2 つ以上のキーを押しながらクリックして一度にできます。

19.1.4　OnMouseMove と OnMouseUp のテスト

選んだカードを好きな位置に引きずっていくようにします。

カード上のマウスダウンしたときの点 A を憶えておいて，カード上を点 P までマウスムーブしたら，$\overrightarrow{\mathrm{AP}}$ だけカードの位置をずらします。

(1) FormMain の private 部に変数を追加する。

```
  private
    Trump : TTrump;
    Idouchuu : Boolean;
    XDown,YDown : Integer;
```

(2) CardMouseDown に “キーを押さないで左クリック” の処理を追加する。

```
      if Shift * [ssShift,ssAlt,ssCtrl] = []   // キーを押さないでクリック
        then begin
               case Button of
                 mbLeft  : begin
                             XDown := X;
                             YDown := Y;
                             Idouchuu := True;
                           end;
               end;
             end;
```

(3) CardMouseMove と CardMouseUp の中身を書く。

```
procedure TFormMain.CardMouseUp(Sender: TObject; Button: TMouseButton;
                                Shift: TShiftState; X, Y: Integer);
begin
  Idouchuu := False;
end; {CardMouseUp}

procedure TFormMain.CardMouseMove(Sender: TObject;
                                  Shift: TShiftState; X, Y: Integer);
begin
  with TCard(Sender) do
    begin
      if Idouchuu
        then begin
               Left := Left+X-XDown;
               Top  := Top +Y-YDown;
             end;
    end;
end; {CardMouseMove}
```

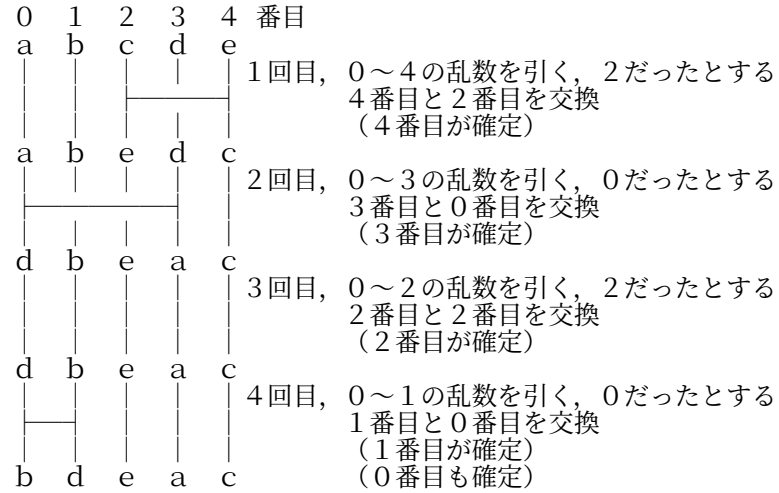
(4) 実行

19.2　シャッフル

トランプのゲームをするには，カードをきる（Shuffle）必要があります。

19.2.1　アルゴリズム

5 個のデータをシャッフルするには次のようにします。わかりやすいように，阿弥陀くじで表してみました。ただし，この阿弥陀くじは隣の線路（縦線）を結ぶ橋（横線）だけでなく，離れた線路を結ぶ跨線橋もあります。



乱数の出方によって，$5! = 5 \cdot 4 \cdot 3 \cdot 2$ 通りの順列が，すなわちすべての順列が生成されることがわかりますね。

19.2.2　TrumpU のメソッドを追加

```
(********** TTrump のメソッド **********)
procedure TTrump.Shuffle;
          (* でたらめに並べかえる *)
  var
    CardNo,RandomNo : TCardNo;
  begin
    for CardNo := 51 downto 1 do
      begin
        RandomNo := Random(CardNo+1);              // 0 ～ CardNo の乱数
        Swap(CardNo,RandomNo);                     // データを交換する
      end;
  end; {Shuffle}
```

交換するデータは，Suit，Rank および表の絵です。
コメントとを参考にして完成させてください。

```
(********** TTrump のメソッド **********)
procedure TTrump.Swap(No1,No2 : TCardNo);
          (* Cards の No1 番目と No2 番目のデータを交換する *)
  var
    SuitSave : TSuit;      // Suit 保存用
```

```
  RankSave : TRank;       // Rank 保存用
begin
                                                    // ┐
                                                    //   Cards[No1] と
                                                    //   Cards[No2] の
                                                    //   Suit と Rank を
                                                    //   交換する
                                                    // ┘
                                                    // それぞれを表にする
                                                    //
end; {Swap}
```

19.2.3　ExampleU の ButtonJikkouClick ハンドラーを変更

実行部の先頭に追加する。

```
Trump.Shuffle;
```

実行して，実行ボタンを押すたびにシャッフルされることを確認してください。同じ札が 2 枚あったりしてはいけませんよ。

19.3　逆引き機能

シャッフルした後で，たとえば 23 番目のカードが何かは スート　Trump.Cards[23].Suit / ランク　Trump.Cards[23].Rank でわかりますが，逆に，たとえば “ハートのA” が何番目にあるかは，Suit と Rank から計算して得ることができなくなりました。探さなくてもわかるように，逆引き用のデータを持たせることにします。

19.3.1　TTrump の public 部に変数 No を追加

```
  public
    Cards : array [TCardNo] of TCard;
    No    : array [TSuit,TRank] of TCardNo;
```

19.3.2　TTrump.Sakusei に No の初期設定を追加

```
with Cards[CardNo] do
  begin
    Left    := CardNo*12;              // 左端の位置
    Top     := CardNo*8;               // 上端の位置
    Tag     := CardNo;                 // 札から番号がわかるようにするため
    Suit    := TSuit(CardNo div 13);   // スート Ord(Suit) = 0〜3
    Rank    := (CardNo mod 13)+1;      // ランク 1〜13
    OmoteNiSuru;                       // 表の絵にする
    No[Suit,Rank] := CardNo;           // 逆引き
  end;
```

19.3.3 TTrump.Swap で No も変更

```
begin
  途中省略（先の問題の部分）
  No[Cards[No1].Suit,Cards[No1].Rank] := No1;      // 逆引き
  No[Cards[No2].Suit,Cards[No2].Rank] := No2;      //
end; {Swap}
```

19.3.4 動作確認のための変更

カードを右クリックしたら，その相方のカードと交換するようにします。

```
procedure TFormMain.CardMouseDown(Sender: TObject; Button: TMouseButton;
                                  Shift: TShiftState; X, Y: Integer);
var
  TargetNo   : TCardNo;       // 交換するカードの番号
  TargetSuit : TSuit;         //                 スート
  TargetRank : TRank;         //                 ランク
begin
      途中省略
      if Shift * [ssShift,ssAlt,ssCtrl] = []   // キーを押さないでクリック
        then begin
               case Button of
                 mbLeft  : begin
                             XDown := X;
                             YDown := Y;
                             Idouchuu := True;
                           end;
                 mbRight : begin
                             TargetSuit := TSuit(Ord(Suit) xor 1);
                             TargetRank := (Rank mod 13)+1;
                             TargetNo := Trump.No[TargetSuit,TargetRank];
                             Trump.Swap(Tag,TargetNo);
                           end;
               end;
             end;
    end;
end; {CardMouseDown}
```

相方は次のように決めています。

列挙型 TSuit が (stClub,stDiamond,stHeart,stSpade) と定義されているので，型キャスト TSuit(整数) と 関数 Ord(スート) の値は次のようになります。

オーダー	→	TSuit(オーダー)
0		stClub
1		stDiamond
2		stHeart
3		stSpade
Ord(スート)	←	スート

ここでは xor は論理演算子ではなく，算術演算子として用いられています。and，or，not も論理演算子としても算術演算子としても用いることができます。算術演算子のときは，数を 2 進法で

考えて各桁ごとに, (0 = `False`, 1 = `True` として）論理演算した結果を値とします。なお，`xor` は排他的論理和（eXclusive OR）です。ペンモードでも出てきましたね。

例

			2 進法	16 進法	10 進法
		M	1010	`$A`	10
		N	1100	`$C`	12
	not	M	0101	`$5`	5
M	and	N	1000	`$8`	8
M	or	N	1110	`$E`	14
M	xor	N	0110	`$6`	6

上のプログラムでは，`Ord(Suit) xor 1` としているので，`Ord(Suit)` を 2 進法で書いた数の最も右の桁の 0 と 1 を反転します。したがって，相方のカードは次のようになります。

`Suit`	クラブ $\leftrightarrow$ ダイヤ,　ハート $\leftrightarrow$ スペード
`Rank`	A $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow \cdots \rightarrow$ Q $\rightarrow$ K $\rightarrow$ A

問題　相方が下記のようになるように書き換えなさい。

(1)

`Suit`	クラブ $\rightarrow$ スペード $\rightarrow$ ハート $\rightarrow$ ダイヤ $\rightarrow$ クラブ
`Rank`	A $\leftrightarrow$ K,　2 $\leftrightarrow$ Q,　3 $\leftrightarrow$ J,　$\cdots$,　6 $\leftrightarrow$ 8,　7 $\leftrightarrow$ 7

(2)

`Suit`	クラブ $\leftrightarrow$ ハート,　ダイヤ $\leftrightarrow$ スペード
`Rank`	K $\rightarrow$ Q $\rightarrow$ J $\rightarrow \cdots \rightarrow$ 2 $\rightarrow$ A $\rightarrow$ K