QCard DLL Lesson 6

Lesson 5 Lesson 7

-
Alyce

QCard DLL Lesson 6 | Keeping Track of Cards on the Table | Checking to see if a card is still on the table. | Removing a card from the table. | DEMO

See Lesson 1 for QCard DLL and WAV files needed for the demo code.

Keeping Track of Cards on the Table

This demo removes cards from the table after the user clicks on them. If he clicks in the same spot again, our program knows that the card has already been removed. The program does not procede to the routine that retrieves the suit and value of the card. It just waits for the user to click somewhere else. If a card has been removed, it can no longer be accessed.

To accomplish this, we add a second dimension to card array to indicate whether it is on the table or if it has been removed.

```
'card(n,1)=index of card in deck
'card(n,2)=visible on table? 1=yes, 0=no
```

After filling the card array with suits and values, we set each card's second dimension to 1, to indicate that it is present on the table. We do this quite easily in a FOR...NEXT loop.

```
[fillCardArray]
    'fill card array
    'cards 1 to 52 are in the first deck
    'cards 53 to 104 are in the second deck
    'use cards Jack through King in each suit, first deck
    card(1,1)=11  'jack of clubs
    card(2,1)=12  'queen
    card(3,1)=13  'king
    card(4,1)=24  'jack of diamonds
    card(5,1)=25  'queen
    card(6,1)=26  'king
    card(7,1)=37  'jack of hearts
    card(8,1)=38  'queen
```

```
    card(9,1)=39  'king
    card(10,1)=50  'jack of spades
    card(11,1)=51  'queen
    card(12,1)=52  'king

    'now use second deck, to fill second half of array
    for i = 1 to 12
        card(i+12,1)=card(i,1)+52
    next
    RETURN

[shuffleCards]
    'first set all cards as visible, card(n,2)=1
    for i = 1 to 24
        card(i,2)=1
    next
```

Checking to see if a card is still on the table.

In the routine to see which card is clicked, we also check to see if the card has already been removed. If the second dimension in the card array for that index is 0, the card has already been removed. If it is 1, it is still on the table. If it has been removed, we simply wait. If it hasn't, we go on to read the value and do other stuff.

```
    'if card is not visible (has been removed), then wait
    if card(clickCard,2)=0 then wait

    gosub [readValue]
    wait
```

Removing a card from the table.

If the card has not already been removed, we remove it and set the second dimension of the card array for that index to 0.

```
    'remove card
    call RemoveCard hBox, card(clickCard,1)

    'set visible to 'off'
    card(clickCard,2)=0
```

In the next lesson we'll remove cards if they match. We now have our methods in place to see if a card is still on the table, and to retrieve its suit and value, so checking for a match will be easy! Feel free to work out the method yourself!

DEMO

See Lesson 1 for QCard DLL and WAV files needed for the demo code.

```
'An open project card game, begun by Alyce Watson, May 27, 2003.
'Uses Qcard32.dll, a freeware library of playing card images.
'DLL by Stephen Murphy.  Qcard32.DLL website:
'http://www.telusplanet.net/public/stevem/

'new in cards6.bas:
'This demo displays a message when cards are clicked,
'and it removes the cards.
'If a card has been removed, it can no longer be accessed.

'new this time
'Add second dimension to card array to indicate
'if it is on the table or has been removed.
'card(n,1)=index of card in deck
'card(n,2)=visible on table? 1=yes, 0=no

dim card(24,2)            'array to hold cards

[varSetup]
i=0         'i will be our counter var in for/next loops
design=1    'default design is circles
newIndex=0  'used when shuffling
tempCard=0  'temp var used when shuffling
clickCard=0 'index of current card clicked by user
gosub [fillCardArray]   'fill array with card values


nomainwin

    WindowWidth=640:WindowHeight=480
    UpperLeftX=1:UpperLeftY=1
```

```
    menu #1, "&File", "&New",[new],"E&xit", [quit]
    menu #1, "&Card Back Design","&Circles",[circles],"&Blue",[blue],_
    "&Red",[red],"&Mountain",[mountain],"&Purple",[purple],
"M&usic",[music]
    graphicbox #1.g, 0, 0, 640, 440
    open "Memory Card Game" for window_nf as #1
    #1 "trapclose [quit]"

    'trap mouse clicks:
    #1.g "setfocus; when leftButtonUp [checkIndex]"

    'get graphicbox handle
    hBox=hwnd(#1.g)

    'open the dll
    open "qcard32.dll" for dll as #qc
    'initialize the deck
    Call InitializeDeck hBox

[new]
    Call SetDefaultValues
    Call SetCurrentBack design

    'draw a nice background
    #1.g "down; fill 10 190 225"
    #1.g "backcolor 10 190 225"

    gosub [shuffleCards]

    'set xy location to start deal
    x=10:y=2
    for i = 1 to 24
        'set status of all cards to 0, which is face down
        Call SetCardStatus card(i,1), 0

        'deal cards
        Call DealCard hBox,card(i,1),x,y

        x=x+100
        if x>510 then   'move to next row
            x=10
            y=y+100
        end if
        playwave "card.wav",sync

       'pause 100 milliseconds between cards
```

```
        call Pause 100
        scan
    next
    wait


[checkIndex]
    clickCard=0:x=0:y=0 'reset values
    mx=MouseX   : my=MouseY 'mouse x and y location
    'Cards are placed in a grid that is 100x100,
    'so it is easy to determine which card is clicked
    'by checking mouse position. Card height is about
    '100, and width is about 80.
    'Index of clicked card is placed in var called clickCard
    'and x,y locations are placed in vars called x and y.
    'MouseY determines row, and MouseX determines column.
    select case
    case my<=102                    'first row
        y=2
        if mx<=90 then clickCard=1:x=10
        if (mx>=110) and (mx<=190) then clickCard=2:x=110
        if (mx>=210) and (mx<=290) then clickCard=3:x=210
        if (mx>=310) and (mx<=390) then clickCard=4:x=310
        if (mx>=410) and (mx<=490) then clickCard=5:x=410
        if (mx>=510) and (mx<=590) then clickCard=6:x=510
    case (my>=102) and (my<202)     'second row
        y=102
        if mx<=90 then clickCard=7:x=10
        if (mx>=110) and (mx<=190) then clickCard=8:x=110
        if (mx>=210) and (mx<=290) then clickCard=9:x=210
        if (mx>=310) and (mx<=390) then clickCard=10:x=310
        if (mx>=410) and (mx<=490) then clickCard=11:x=410
        if (mx>=510) and (mx<=590) then clickCard=12:x=510
    case (my>=202) and (my<302)     'third row
        y=202
        if mx<=90 then clickCard=13:x=10
        if (mx>=110) and (mx<=190) then clickCard=14:x=110
        if (mx>=210) and (mx<=290) then clickCard=15:x=210
        if (mx>=310) and (mx<=390) then clickCard=16:x=310
        if (mx>=410) and (mx<=490) then clickCard=17:x=410
        if (mx>=510) and (mx<=590) then clickCard=18:x=510
    case (my>=302) and (my<402)     'fourth row
        y=302
        if mx<=90 then clickCard=19:x=10
        if (mx>=110) and (mx<=190) then clickCard=20:x=110
        if (mx>=210) and (mx<=290) then clickCard=21:x=210
```

```
        if (mx>=310) and (mx<=390) then clickCard=22:x=310
        if (mx>=410) and (mx<=490) then clickCard=23:x=410
        if (mx>=510) and (mx<=590) then clickCard=24:x=510
    case else
        clickCard=0
    end select

    if clickCard=0 then wait

    'if card is not visible (has been removed), then wait
    if card(clickCard,2)=0 then wait

    gosub [readValue]
    wait


[readValue]
    thisVal = GetCardValue(card(clickCard,1))
    'ace=1,deuce=2....jack=11,queen=12,king=13
    thisSuit = GetCardSuit(card(clickCard,1))
    'returns 1=Clubs, 2=Diamonds, 3=Hearts, 4=Spades.


    'remove card
    call RemoveCard hBox, card(clickCard,1)

    'set visible to 'off'
    card(clickCard,2)=0

    msg$="Card ";clickCard;" is removed."
    #1.g "place 10 420"
    #1.g "\" ; msg$; space$(400)
    RETURN


'setting new card back doesn't restart game,
'so new back won't show until new game is started:
[circles] design=1:goto [setDesign]
[blue] design=2:goto [setDesign]
[red] design=3:goto [setDesign]
[mountain] design=4:goto [setDesign]
[purple] design=5:goto [setDesign]
[music] design=6:goto [setDesign]

[setDesign]
    Call SetCurrentBack design
```

```
    'design can be 1,2,3,4,5,6 for 6 possible designs
    wait


[fillCardArray]
    'fill card array
    'cards 1 to 52 are in the first deck
    'cards 53 to 104 are in the second deck
    'use cards Jack through King in each suit, first deck
    card(1,1)=11  'jack of clubs
    card(2,1)=12  'queen
    card(3,1)=13  'king
    card(4,1)=24  'jack of diamonds
    card(5,1)=25  'queen
    card(6,1)=26  'king
    card(7,1)=37  'jack of hearts
    card(8,1)=38  'queen
    card(9,1)=39  'king
    card(10,1)=50  'jack of spades
    card(11,1)=51  'queen
    card(12,1)=52  'king

    'now use second deck, to fill second half of array
    for i = 1 to 12
        card(i+12,1)=card(i,1)+52
    next
    RETURN


[shuffleCards]
    'first set all cards as visible, card(n,2)=1
    for i = 1 to 24
        card(i,2)=1
    next

    playwave "shuffle.wav",async

    'now shuffle cards
    for i = 1 to 24
        newIndex=int(rnd(0)*24)+1
        tempCard=card(i,1)  'temp var to allow switching values
        card(i,1)=card(newIndex,1)
'this index now contains value from random index
        card(newIndex,1)=tempCard
'random index now contains value from other index
```

```
'now card(i,1) has switched values with a random card in the array
    next
    playwave "shuffle.wav",sync
    RETURN

[quit] close #qc:close #1:end


''''''''''''''''''''
'subs and functions:
Sub Pause ms
    'pause ms number of milliseconds
    calldll #kernel32,"Sleep",_
    ms as long, re as void
    End Sub

Function GetCardSuit(nC)
    'returns 1=Clubs, 2=Diamonds, 3=Hearts, 4=Spades.
    calldll #qc, "GetCardSuit",nC as long,_
    GetCardSuit as long
    End Function

Function GetCardValue(nC)
    'ace=1,deuce=2....jack=11,queen=12,king=13
    calldll #qc, "GetCardValue",nC as long,_
    GetCardValue as long
    End Function

Sub InitializeDeck hndle
    calldll #qc, "InitializeDeck",_
    hndle as ulong,r as long
    End Sub

Sub SetCardStatus nC,face
    'nC is number of card - 1-52 in first deck and
    '53-104 in second deck, if used
    'face:  0=facedown,1=faceup
    calldll #qc, "SetCardStatus",nC as long,_
    face as long,r as void
    End Sub

Sub DealCard hndle,nC,x,y
    'places card on window whose handle is hndle at x,y
    'nC is number of card - 1-52 in first deck and
    '53-104 in second deck, if used
    calldll #qc, "DealCard",hndle as ulong,nC as long,_
```

```
    x as long,y as long,r as void
    End Sub

Sub SetCurrentBack nV
    'nV can be 1,2,3,4,5,6 for 6 possible designs
    calldll #qc, "SetCurrentBack",nV as long,r as void
    End Sub

Sub SetDefaultValues
    'reset all card properties back to their default values.
    calldll #qc, "SetDefaultValues",r as void
    End Sub

Sub RemoveCard hndle,nC
    'removes a card from screen that was
    'drawn with DealCard, replacing screen background
    calldll #qc, "RemoveCard",hndle as ulong,_
    nC as long,r as void
    End Sub
```

QCard DLL Lesson 6 | Keeping Track of Cards on the Table | Checking to see if a card is still on the table. | Removing a card from the table. | DEMO

Lesson 5 Lesson 7