Briscola

version 1.6

Contents

This demo program for the CardPack ActiveX Control implements a very simple Italian game called *Briscola*. The program can be played one-on one against the computer or against another human opponent across a network. It works with both Microsoft Visual Basic 4 and 5.

Choose one of the following topics:

Game Rules

Network Usage

Hacking and Cheating

Program Structure

See CARDPACK.HLP for further details.

Copyright Notice

Disclaimer

Rules

Your goal is to outscore your opponent by capturing valuable cards.

A 40-card deck is used (ace..7,J,Q,K). Each player starts with three cards. One extra card is placed at the bottom of the stack, facing up. The *suit* of this card determines what suit will be used as trump cards (*Briscola*).

Scores are as follows (suit is not relevant):

ace	11 points
three	10 points
Jack	1 point
Queen	2 points
King	3 points
any other card	0 points

The first player will play a card. If it is valuable, the opponent can try to take it by playing a card of the same suit with a higher score/value or a trump card. If not, the opponent can respond with another (possibly useless) card. In this case the first player will capture the opponent's card.

Within the same suit, taking order follows the score order first, then the normal values (e.g. a 6 takes a 4 or a 5, but is always taken by a 3 or an ace.

After both players have laid down one card, the winner takes both cards, and two more cards are dealt. The winner also gets to play first in the next hand. The game goes on like this until all cards (including the last *briscola* under the deck) have been dealt and played.

Network

This program allows you to challenge an opponent across the network. For that, you need a network supporting NetDDE. We have successfully tested it with Windows for Workgroups 3.11 and Windows 95 (16-bit only).

The program registers its own DDE share when it starts. After that, at any moment, you can receive a challenge. The program will tell you the names of your opponent and his/her workstation. You can either refuse or accept the challenge.

When you accept the challenge, the two programs will cross-link via DDE their card stacks and synchronise the move and score-keeping. The system used to keep the two instances in sync is not yet very robust and, in some occasions, we have seen it fail. In a production-quality game you may want to give it a little more thought than we did for this demo.

Hacking and Cheating

Since you have the source code, you can easily hack the game or cheat to see how it works or to debug it more easily. Besides hacking the source code, you can cheat using some built-in options:

Turn on Trace Mode

Under the Options menu, you will find an item, *Debug Window*. If you check it, you'll get a crude trace window, where various game functions will print their status or their reasoning. Probably you won't understand much of the messages at the start, but you have a framework in place for digging into the program. You can log to this window using the *Trace (msg$)* subroutine.

This built-in window is provided so that you can trace the compiled program even without having the Visual Basic interpreter installed. This is very useful for debugging network games.

Peeking Cards

Another entry under the Options menu, *Peek Cards*, It will turn your opponent's cards face-up, and allows you to peek played cards by clicking and holding the mouse over one of the played cards stacks.

Program Structure

Despite the fact that the game logic of Briscola is very simple, the actual program has grown to be a bit complex. Most of that depends on the fact that we have thrown in *Auto Play* (also called *Demo Mode*) and *Network Play*. In order to allow these two features, even the basic functions of the game need to be a little more complex than usual.

Here are a few points that deserve some explanation before you look at the source code:

Object Orientation

Although the program is not properly object-oriented, it goes in that direction. Since it was written in Visual Basic 3 at the start, there was not a lot more that could be done. If the program was written from scratch now for Visual Basic 5 or later, it could look a lot better. Nevertheless, the game tries to separate the user interface functionality (in the **Briscola.frm** form) from the game logic (in the **Briscola.bas** module). Moreover, you will see that variables and procedures are grouped as pseudo-objects: ("player", "game", "hand"). This makes it easier to understand who controls what.

State Machine

The game is basically driven by couple of state machines. There is a "game" state machine, which is represented by the status of all Cardpack controls on the table, and a "hand" state machine, driven by the number of cards that played and are "on the table", held by the **OnTable** Cardpack control.

The most important point is that state transitions in the game (e.g. player 1 just played, now it's player 2's turn) are triggered by the **Change** event of the controls, rather than by the procedure that caused the change to happen (e.g. *not* the MouseMove that dropped a card). The reason is that a card can be dropped under program control (by the user or by the "robot" that plays against you), but also by a remote opponent over the network via DDE! In that case the only way to detect that something has changed is, indeed, via the **Change** method.

Other actions that the program must take itself, like dealing cards or removing played cards from the table, are only done after checking that this computer is supposed to move cards around. When in network mode, there is some simple logic that prevents both computers to try to deal cards at the same time. Basically, the last to play gets to move cards around, while the other one will just watch the board change through DDE.

The Timer

In order to *pace* the program and make it play at human speed, rather than as fast as possible, all autonomous actions are driven by a slow timer: **GameTimer**. While a game is running, the timer wakes up every 3 seconds, looks at the table and decides what to do. If a hand is over and this computer is supposed to deal, it will deal a new hand. If a hand is in progress and the computer is supposed to play for the side that's up, it will call the **Robot** to think a card and play it.

You will appreciate the timer especially when the program is put in **Demo** mode. It will play for both sides, while still retaining a human pace that lets you understand what is going on and follow the compuer's strategy.