



Universita' degli studi di Palermo

Diploma Universitario in Informatica

Programmazione e
Laboratorio di Programmazione

Tesina di fine anno: Otto Carte

Realizzato dagli alunni:

Davide Caracausi - Dario Maratta - Luigi Maniscalco

A.A. 2000/2001



UNIVERSITA' DEGLI STUDI DI PALERMO

Anno Accademico 2000/2001

MANUALE DI RIFERIMENTO AL CODICE SORGENTE DI

OTTO CARTE 1.4



A CURA DI:

DAVIDE CARACAUSI
DARIO MARATTA
LUIGI MANISCALCO

COMPLIMENTI PER LA SCELTA!



Genius at work ...

Introduzione

Se avessimo potuto scegliere tra tutte le possibili cose da fare non avremmo mai scelto un gioco di carte! Per la verità non avremmo mai nemmeno scelto un gioco; di fronte ad una nuova sfida, però, la nostra filosofia non è mai stata quella della resa. In tre settimane ci siamo scontrati con i nostri peggiori incubi (access violations, abstract errors, testardaggine di Delphi che nemmeno vuole mettere da solo i punti e virgola ovvi!!), ma alla fine i risultati ci hanno strappato innumerevoli sorrisi! Indubbiamente ci siamo divertiti a trovare soluzioni fantasiose e bizzarre, per poi convertirle nel massimo del tecnicismo; ma in questo ci siamo accorti che se anche il Delphi avesse potuto scegliere, nemmeno lui avrebbe voluto un gioco!

Abbiamo visto crescere il codice giorno dopo giorno, donandogli mille piccole sfumature che ognuna delle nostre dure teste ha preteso di inserire come contributo alla caratterizzazione. Ci siamo presi cura che disponesse di tutte quelle “futilità” tanto gradite a chi pretende di giocare... infine non pretendiamo affatto di aver fatto un capolavoro, infondo il tempo è stato molto tiranno, specie avendo sul groppone il peso di altre tre materie, ma in definitiva ci riteniamo soddisfatti come speriamo, scusando le spiritosaggini, lo siano anche i nostri esaminatori!

ED ORA SI FA SUL SERIO...

Descrizione della struttura dati

Le strutture dati utilizzate sono relative ai motori grafico e di gioco. La descrizione degli oggetti grafici è indicata nella sezione relativa alle procedure del motore grafico. Gli oggetti *Mazzo* (di tipo *CardSet*) e *Giocatori* (di tipo *Players*) contengono un set di sottostrutture relative alla duplice natura delle carte; se da un lato esse si possono rappresentare facilmente con elementi numerici, dall'altro devono essere visualizzate a video nella loro veste grafica. Il set di oggetti è meglio descritto nel paragrafo sui tipi complessi utilizzati; ci si limiterà in questo a descrivere la struttura numerica utilizzata per la rappresentazione delle carte. Preliminarmente bisogna fissare i punti chiave del motore di gioco che sono fortemente connessi alle regole del gioco stesso:

1. Le carte da giocare sono 52 distribuite fra il mazzo da cui pescare, i giocatori, ed il mazzo degli scarti;
2. La tipica struttura di un mazzo da cui pescare è una LIFO il cui stack pointer punta alla carta appena pescata o da pescare;
3. La tipica struttura di un mazzo di scarti è una pila il cui stack pointer, salendo ad ogni push, punta sempre all'ultima carta scartata;
4. Essendo le carte 52, essendocene una immediatamente scartata, dovendo possederne una il giocatore avversario, il giocatore tipo avrebbe la certezza di poterne scartare una solo avendone in mano 37 (tutte le carte degli altri semi, eccezion fatta per quelle di pari valore addizionate di una unità), questo è quindi il massimo numero di carte di cui ogni giocatore può disporre.

Secondo il punto 1 ogni carta può trovarsi in quattro monti:

- Nel mazzo di pescaggio;
- In mano al giocatore 1;
- In mano al giocatore 2;
- Nel mazzo degli scarti;

da questo si deduce facilmente che via via liberandosi il monte delle carte di pescaggio si liberano allocazioni in quantità certamente superiore all'esigenza del mazzo degli scarti.

Da quest'ultima intuizione viene l'idea di utilizzare per i due mazzi una unica matrice di consistenza 52 utile a contenere nella parte bassa la pila delle carte da pescare, il cui cielo è puntato dall'indice *Mazzo.IndPesca*, ed in quella alta la pila inversa del mazzo degli scarti, il cui fondo è puntato dall'indice *Mazzo.IndScarto* che rappresenta in di volta in volta l'ultima carta scartata. Questo sistema ciclico consente una notevole riduzione di spazio mnemonico e garantisce una struttura compatta richiusa tutta nell'unico oggetto *Mazzo*. Dal mazzo le carte vengono pescate ed inviate ad una delle arrays di consistenza 37 contenute nell'oggetto

Giocatori a sua volta array binario del tipo *Player* in cui sono definiti oggetti e strutture che costituiscono il set di ogni giocatore. Nel massimo rispetto del rispetto mnemonico la struttura garantisce una alta funzionalità legata ad una grande semplicità di gestione, prerogativa tutt'altro che presente nell'alternativa prima costituita dalle liste circolari.

Nell'ottica della simulazione ogni motore è stato concepito per effettuare le operazioni proprietarie con la massima verosimiglianza alle operazioni reali. Il mazzo viene mescolato grazie a 4 algoritmi che ne simulano una disposizione quasi umana, in prima istanza scambi indicizzati casuali copiano la classica "smazzettata" o mescolazione da solitario, poi stravolgimenti larghi e stretti simulano le mirabolanti evoluzioni dei croupiers ed, infine il classico taglio del mazzo da parte dell'avversario che suggella la legittimazione della mano in corso. Sempre nella stessa ottica il mazzo, alla fine di ogni mano, viene riunito similmente a come avviene manualmente e su questo avviene la nuova mescolazione per avere il massimo della fedeltà di gioco.

Numericamente ogni carta è rappresentato da un intero da 0 a 51, le carte sono ordinate da asso a re e seguendo la classica regola del "Come, Quando, Fuori, Piove". In tal modo ogni carta è rappresentabile in valore e seme utilizzando i semplici operandi DIV e MOD, apposite funzioni ne rendono immediatamente disponibili le proprietà. Con apposite procedure il contenuto delle matrici numeriche vengono collegate alle relative matrici grafiche che contengono le immagini delle carte come descritto nella sezione del motore grafico.

Descrizione del processo logico di sintesi

La realizzazione di un gioco di carte presenta una serie di problematiche legate al fatto che i giocatori non debbano cambiare turno alternativamente ad ogni singola mossa ma piuttosto ad un numero imprecisato; ciò conduce, nel processo di minimalizzazione del codice, a ricercare punti univoci per effettuare gli scambi. Questo approccio è di tipo kernel/modules, infatti una volta individuata e scritta una procedura di cambio turno (kernel) da essa dipartono le procedure delle diverse modalità di gioco, selezionate in loco da una IF chain o da un CASE statment: aggiungerne di nuove (modules) è tutt'altro che difficoltoso. Il grafo procedurale tipicamente converge in questo punto che è nodale per il programma. La filosofia onion's skin ha guidato il processo accrescitivo del programma, il cui embrione è rappresentato dall'insieme motore di gioco ed environment definition che all'inizio è stato testato abbondantemente senza l'ausilio delle sovrastrutture grafiche, queste ultime, in maniera sempre più profonda, sono state inserite aggiungendo semplicemente le proprietà necessarie agli oggetti che già ben rappresentavano la situazione numerica. Seguitando ad involucrazione si passa via via per i livelli dell'I.A. ed infine dell' I/O file. Discorso a parte è lo stack di rete il quale risponde a comandi asincroni che generano nel top level requests diretti al kernel, questa categoria di procedure sono di tipo affiancativo da localizzarsi in un livello non dissimile da quello dello del kernel ma da questo disgiunto.

Descrizione dei tipi complessi (astratti)

CardSet = Record

| | |
|---------------------------------|---|
| Carte: array[1..52] of Integer; | Array numerica contenente le carte del mazzo; |
| IndPesca, IndScarto: NelMazzo; | Indici della testa delle pile di pescaggio e di scarto, implementate all'interno dell'array numerica; |
| Pics: TPicClip; | Contenente le immagini delle 52 carte; |
| Scarto, Dorso: TImage; | Immagini dello scarto attuale e del dorso delle carte; |

end;

HitParade = Record

| | |
|--|--|
| | {48 byte in totale}; |
| {Formato: cccccccccccccccccceeeaaaabbbbfffgggghhhddddd}; | |
| MosseMed: Integer; | {04 byte in chiaro bbbb}; |
| MosseMin: Integer; | {04 byte in chiaro aaaa}; |
| Nome: String[20]; | {20 byte in chiaro ccccccccccccccccccc}; |
| Perse: Integer; | {04 byte in chiaro dddd}; |
| Punteggio: 0..2400; | {04 byte in chiaro eeee}; |
| TempoMed: Integer; | {04 byte in chiaro gggg}; |
| TempoMin: Integer; | {04 byte in chiaro ffff}; |
| Vinte: Integer; | {04 byte in chiaro hhhh}; |

end;

Player = Record

| | |
|-------------------------------|---|
| Carte: CarteInMano; | Array numerica contenete le carte in possesso del giocatore ; |
| Enabled: Boolean; | Abilitazione dell'oggetto; |
| Indice: InMano; | Numero delle carte in possesso del giocatore ; |
| ManiVinte: 0..8; | Numero delle partite vinte durante un singolo gioco; |
| Mosse: Integer; | Numero delle mosse vinte durante un singolo gioco; |
| Nome: String[20]; | Nome del giocatore;; |
| Pics: array[1..37] of TImage; | Array grafica contenente le immagini della carte in possesso del giocatore; |
| Punteggio: 0..2400; | Punti accumulati durante un singolo gioco; |
| Tempo: Integer; | Tempo di gioco riguardante un singolo gioco; |

end;

Descrizione dei blocchi del gioco

Le procedure e le funzioni del gioco sono state riordinate secondo l'appartenenza ad una delle categorie:

- | | |
|------------------------------|---|
| 1. Generale | relative a inizializzazioni o strumenti di utilità generale; |
| 2. Grafica | relative alla gestione delle grafiche e degli oggetti visuali; |
| 3. Intelligenza Artificiale | relative alla ribattezzata “stupidità artificiale” o motore I.A.; |
| 4. I/O su file | relative alla gestione del TextFile della classifica; |
| 5. Motore di gioco | relative alla gestione numerica del gioco; |
| 6. Motore di gioco e grafica | relative alla gestione integrata numerico-grafica; |
| 7. Stack di rete | relative alla gestione dell'I/O da rete. |

Descrizione delle procedure Generali

function CvStr(z, n: Integer): String;

Restituisce la conversione dall'Intero **z** a Stringa formattata di lunghezza **n**.

function Sign(x: Integer): Turno;

Restituisce il segno dell'Intero relativo **x**.

procedure Delay(n: LongInt);

Introduce un ritardo di esecuzione di **n** millisecondi.

procedure ErroreFile;

Verifica la presenza, nella directory specificata, del file mazzo.

procedure Fine;

Effettua il controllo di fine gioco e l'abilitazione delle procedure di salvataggio classifica.

Descrizione delle procedure Grafiche

procedure CreaCarta(chi: Turno; n: InMano);

La procedura genera in runtime un oggetto di tipo Timage non visibile, contenente l'immagine della n-esima carta in mano a "chi", estratta dall'immagine del mazzo completo. Inoltre, a seconda della modalità di gioco, assegna a tale oggetto un handler dell'evento OnClick, in modo da consentire materialmente al giocatore di turno di poter scartare le carte. La procedura è richiamata in fase di distribuzione delle carte all'inizio di ogni mano, e ogni qual volta un giocatore peschi una carta dal monte.

N.B. La procedura non visualizza la carta, bensì si limita a predisporla per l'utilizzo all'interno del motore grafico.

procedure Gioca(Sender: TObject);

La procedura individua la Timage (carta) sulla quale il giocatore ha cliccato tramite un ciclo di confronti fra gli oggetti di tipo Timage ed il parametro 'Sender'.

Avendo implementato anche una modalità 'gioco in due a carte scoperte' nasce il problema di identificare a quale dei due giocatori appartenesse la carta (variabile locale 'chi'), per cui il ciclo di ricerca risulta essere sdoppiato in due.

Una volta individuato il sender, identificato dagli indici i (per il giocatore 0) ed h (per il giocatore 1), questo viene mandato, insieme al proprietario della carta ('chi'), alla procedura GiocaCarta che si occupa della mossa vera e propria, almeno che non si giochi come client, nel cui caso la mossa è passata direttamente al server come stringa.

procedure MmCambiaMazzo Click(Sender: TObject);

Una delle tante piccole "chicche" di questo gioco. Questa procedura infatti, richiamata dalla corrispondente voce del menu "Opzioni/Cambia Mazzo", apre un OpenFileDialog, consentendo di caricare un file immagine delle carte personalizzato. Una volta caricato il file, l'immagine viene trasferita sull'oggetto "Mazzo.Pics" di tipo TpicClip, che provvede a suddividere equamente l'immagine in una griglia di 53 colonne per una riga. Essendo una carta dimensionata a 71x96 pixels, è pertanto necessario che l'immagine caricata abbia le seguenti caratteristiche:

- 3763 pixels in orizzontale ((52 carte + 1 dorso) x 71 pixels);
- 96 pixels in verticale;

Per come è stato costruito il mazzo di carte, occorre inoltre tener conto che il colore \$007F00 viene reso trasparente (per ottenere gli angoli delle carte smussati). Le carte sono ordinate secondo l'ordine Cuori, Quadri, Fiori, Picche, da 1 a K, più il dorso della carta posto alla cinquantatreesima posizione.

Effettuato il cambio, le carte vengono quindi aggiornate già durante il gioco.

procedure MmSfondoCaricaClick(Sender: TObject);

Discorso analogo per quanto riguarda lo sfondo, ad eccezione della suddivisione e della trasparenza. L'unica limitazione in questo caso è data dalle dimensioni dell'immagine, che deve essere di 612x404 pixels. L'oggetto utilizzato per aprire il file è questa volta un OpenPictureDialog che offre un'anteprima dello sfondo.

procedure MmSfondoNessunoClick(Sender: TObject);

Non a tutti piace avere uno sfondo dietro le carte... questa voce di menu attiva/disattiva l'attributo "visible" dello sfondo.

procedure PescaDalMazzo (Sender: TObject);

Analogamente alla procedura “Gioca”, questa procedura entra in azione quando viene generato un evento OnClick sul monte. A seconda della modalità e del turno (“ToccaA”) viene quindi richiamata la procedura “pesca(chi)” nel caso in cui non si stia giocando in modalità client, altrimenti viene inviata direttamente la stringa di comando al server.

procedure SemeScelto(Sender: TObject);

Anche questa procedura viene richiamata quando si clicca su quel fantomatico menu che appare sulla sinistra del tavolo di gioco quando un giocatore gioca un dannatissimo 8... Questo menu infatti è costituito da quattro TImage (una per seme) non visibili. Quando un 8 viene giocato, il menu appare e il gioco rimane in attesa di una scelta da parte dell’utente(vedi procedura ScegliSeme): nel momento in cui viene cliccata una delle immagini la procedura individua il sender e comunica il seme corrispondente alla procedura “CambiaSeme”, ripristina l’indicatore del seme attuale nascondendo il menu di scelta, riattiva il giocatore di turno e passa infine alla procedura “Cambia Turno”. Nel caso di gioco come client si limita ad inviare la stringa di comando al server.

procedure SpostaCarta(var Carta: TImage; xdest,ydest: Integer; n: NelMazzo);

Questa procedura rileva la posizione attuale di “Carta” e la sposta lungo una linea retta fino alle coordinate “xdest” e “ydest”, attribuendole alla fine l’immagine della carta “n”;

procedure Visualizza(chi: Turno);

La procedura ridisegna le carte del giocatore “chi” riposizionandole orizzionalmente centrate nella form1. Il tutto si svolge riassegnando ad ogni immagine quella corrispondente alla i-esima carta in mano al giocatore (con i che varia da 1 a Giocatori[chi].Indice). Ogni immagine è quindi portata davanti alle altre già disegnate, ed infine resa visibile. In questo modo si garantisce una corretta visualizzazione dopo ogni mossa.

Descrizione delle procedure di Intelligenza Artificiale

procedure Motore;

Il questa procedura il computer “ragiona” su quale carta giocare. L’idea di fondo è molto semplice; controlla, tra le carte in mano, se ce ne sono giocabili, ed in caso positivo tira la prima disponibile. In caso contrario, pesca una carta dal monte e riesegue l’operazione ricorsivamente. Nel caso in cui giochi un otto, viene selezionato semplicemente il seme successivo a quello attuale.

La nostra intenzione era quella di creare tre motori di I.A. differenti, caratterizzati con la tattica di gioco di ognuno di noi. Ci siamo però accorti che la “stupidità artificiale” risultava essere più efficace di qualsiasi altra strategia logica. Abbiamo pertanto deciso di lasciare, in ultima istanza, la nostra beneamata “stupidità artificiale”.

Descrizione delle procedure di I/O su File

```
procedure AggiornaClassifica;
```

Controlla il file classifica e lo aggiorna in funzione dell'esito del gioco appena svolto.

[illegible]

Apri ed effettua un ordinamento per campo sul file classifica.

Descrizione delle procedure del Motore di gioco e Grafica

procedure AggiornaScarto(Carta: NelMazzo);

La procedura aggiorna le variabili globali ValoreAttuale e SemeAttuale in funzione di “Carta”, inoltre aggiorna la piccola immagine sulla sinistra della form1 con il nuovo seme.

procedure CambiaSeme(n: CQFP);

Analoga alla precedente, aggiorna solo il seme ponendolo a “n” (0=cuori, 1=quadri, 2=fiori, 3=picche).

procedure CambiaTurno;

Cuore del gioco, questa procedura gestisce tutte le operazioni necessarie tra una mossa e l'altra, nell'ordine:

1. Ferma il tempo totale (statistiche);
2. Aggiorna il tempo di gioco complessivo del giocatore al turno (per le statistiche);
3. Azzera il tempo totale e lo riattiva;
4. Aggiorna il numero di mosse complessive del giocatore (per le statistiche);
5. Cambia il turno(“toccaA”);
6. Verifica che il nuovo giocatore di turno possa giocare (se deve pescare e le carte del monte sono finite assegna la vittoria a chi ha meno carte);
7. Aggiorna la label del Turno con il nome del giocatore
8. Disattiva il giocatore attivo e viceversa;
9. Ferma il cronometro parziale;
10. Azzera il cronometro come impostato nel menu TempoMossa;
11. Toglie un secondo al cronometro parziale (perso durante l'elaborazione);
12. Riattiva il Cronometro;
13. Aggiorna i messaggi nella StatusBar;
14. Aggiorna i punteggi;
15. Passa il gioco al Computer nel caso di gioco contro il Computer (semprech  sia il suo turno!);

procedure ContaTempo;

Richiamata dal Cronometro e da CambiaTurno, decrementa di 1 il valore del cronometro ed assegna la vittoria all'avversario qualora il tempo sia scaduto.

procedure GiocaCarta(chi: Turno; n: InMano);

La procedura verifica che la carta n-esima in mano a “chi” sia effettivamente giocabile.

Se s , provvede ad eseguire l'animazione (Sposta Carta), ad aggiornare il mazzo degli scarti sia come array che graficamente, ed infine a distruggere l'immagine della carta giocata (Free). Qui occorre ricordare un bug riscontrato in Delphi in merito al “free” di una immagine: per ragioni che ci sono oscure, se l'immagine di cui si effettua il free risulta avere Zposition=True, cio  se   in fronte rispetto alle altre, si genera un Access Violation Error. A niente   servito un SendToBack prima del free: il problema   stato aggirato evitando il free di queste immagini. Questo implica che all'atto di un nuovo create in runtime della medesima immagine superstite, il puntatore che la identifica venga reindirizzato ad una nuova area di memoria contenente una

nuova Timage, lasciando la vecchia non referenziata (Garbage).La soluzione è chiaramente poco pulita, ma senz'altro rapida e indolore (o quasi).

Viene quindi aggiornato l'array delle carte in mano a "chi" , per passare infine alle ultime verifiche:

1. Se il giocatore vince, si passa alla procedura Vittoria;
2. Se la carta giocata è un otto (grrrrr...) viene attivato il menu di scelta (procedura ScegliSeme);
3. Se si gioca in rete la nuova situazione è inviata al client (la procedura non è accessibile al client);
4. Se la mano non è conclusa, richiama CambiaTurno.

procedure Inizializza;

La procedura inizializza ad inizio partita la grafica del monte, del mazzo degli scarti ed il menu nascosto per la scelta del seme

procedure IniziaPartita(chi: Turno; n: InMano);

Genera una nuova mano di gioco, appoggiandosi alle varie procedure di inizializzazione (riunisciMazzo, Resetta, mescola, daicarte, inviasituazione) ,e riazzera i contatori (cronometro, Indscarto, indpesca, toccaa). "n" è il numero di carte da distribuire, "chi" il giocatore che inizia per primo.

procedure Resetta;

Questa procedura azzerà gli array delle carte in mano ai giocatori ed effettua il free di tutte le immagini corrispondenti (Bug!: vedi la procedura giocacarte per maggiori informazioni sul free).

Azzerà infine le variabili globali ValoreAttuale e SemeAttuale.

procedure ScegliSeme(chi: Turno);

La procedura disattiva temporaneamente entrambi i giocatori, in attesa che quello di turno scelga il seme.

Nasconde l'immagine del seme attuale e visualizza le quattro Timage per la scelta del seme.

Viene richiamata dalla procedura GiocaCarta e dalla Procedura ScegliSeme ogni volta che viene giocato un 8 (a rigrrrrrrr...).

Descrizione delle procedure di Motore di gioco

function CalcolaPunteggio(chi: Turno): Word;

Procedura che calcola il punteggio delle carte del giocatore “chi”. Il punteggio, contenuto nella variabile “p” ha un valore massimo calcolato in 300 punti circa.

function Giocabile(Carta: NelMazzo): Boolean;

Indica, in valore booleano, se la carta cliccata è giocabile secondo le regole del gioco.

function PuòPescare(chi: Turno): Boolean;

Indica, in valore booleano, se il giocatore può pescare dal monte delle carte.

function Seme(n: Integer): Integer;

Da il seme della carta passata.

function SemeAsString(n: Integer): String;

Converte il seme “numerico” della carta in stringa.

function Valore(n: Integer): Integer;

Da il valore della carta.

function ValoreAsString(n: Integer): String;

Restituisce il valore della carta come stringa.

procedure cGlobalTimeTimer(Sender: TObject);

Incrementa il tempo della mossa.

procedure CronometroTimer(Sender: TObject);

Richiama la procedura *CalcolaTempo*. Inoltre, riscontrando degli sporadici problemi di sincronizzazione tra carte utilizzate ed immagini, abbiamo anche dovuto inserire dei controlli che, al massimo dopo un secondo, rimettono a posti la situazione.

procedure DaiCarte(n: Integer);

Procedura, riservata soltanto al server, che tramite il parametro passato, distribuisce opportunamente le carte dal monte.

procedure GiocaClient(n: InMano);

Corrispondente di *GiocaCarta*, spedisce la carta giocata al server.

procedure Mescola(var MazzoLocale: CardSet);

Questa procedura simula la mescolazione delle carte. Infatti consta di quattro fasi:

1. la mescolata;
2. lo stravolgimento stretto (con il quinto del mazzo);
3. lo stravolgimento largo (con il mazzo intero)
4. il taglio del mazzo.

procedure MmNuovaPartitaClick(Sender: TObject);

Disabilita il form del “campo di gioco” e visualizza la finestra della scelta della modalità

procedure MmTempoXXClick(Sender: TObject);

Reimposta il tempo massimo della mossa a 15/30/60/99 secondi, sottraendogli i secondi già trascorsi, e riposiziona opportunamente il menù.

procedure MmVelocitaFullClick(Sender: TObject);

Imposta a 10 millisecondi il tempo della mossa da parte del computer.

procedure MmVelocitaUmanaClick(Sender: TObject);

Imposta ad 1 secondo il tempo della mossa da parte del computer.

procedure MmVisualizzaAllaFineClick(Sender: TObject);

Permette di scegliere se visualizzare, alla fine del gioco, la classifica.

procedure Ordina(var a: CarteInMano; n: InMano);

Permette la scelta o meno dell’ordinamento, secondo valore e seme, della carte “in mano”. Si è preferito non utilizzare un algoritmo di ordinamento più complesso vista l’esiguità degli elementi (37 al massimo).

procedure Pesca(chi: Turno);

Questa procedura distribuisce, di volta in volta, una carta dal mazzo. Se si sta giocando in rete, la procedura invia inoltre la carta pescata all’**Utente**.

In caso di ultimazione delle carte ed impossibilità a giocare, la procedura assegna la vittoria al giocatore avversari.

Una funzionalità importante, soprattutto per gli “sbadati” è quella di informare, tramite la barra di stato, se si deve pescare o se si può pescare una carta.

procedure RiunisciMazzo;

Richiamata da *IniziaPartita* (a meno che non si tratti della prima partita), ricomponi il mazzo delle carte unendo gli scarti con le carte dei giocatori. In questo caso si simula proprio quest’operazione, che precede il rimescolamento delle carte.

procedure Stravolgi(var MazzoLocale: CardSet; AGruppiDi: NelMazzo);

Riproduce fedelmente le mirabolanti evoluzioni dei croupiers.

procedure Vittoria(chi: Turno);

E’ la procedura che assegna la vittoria al vincitore. Attiva poi un piccolo file wave, in base all’esito della partita, e se si arriva alla nona partita attiva il riepilogo delle informazioni del gioco.

Nel caso si stia giocando in modalità rete, il server invia il relativo comando al client, secondo le specifiche dello stack di rete.

Descrizione delle procedure e delle istanze dello stack di rete

Var Utente: TIdPeerThread;

Variabile globale pubblica contenete l'handle del thread di connessione dell'unico utente attivo rappresentante dell'attività lato client sul server.

procedure ChatInMsg(stringa: String);

Gestitrice dell'evento di presenza nel buffer di ricezione di messaggi di chat.

procedure ChatSenderCS(stringa: String);

Inoltratore dei messaggi di chat, come tutte le procedure contrassegnate con il suffisso CS è valida sia per il client che per il server.

procedure Chiusura;

Effettua la chiusura dell'applicazione e il reset dello stack di rete.

procedure ClientConnected(Sender: TObject);

Rileva l'handle di connessione del client verso il server.

procedure ClientDataAvailable(Buffer: String);

Rileva la presenza di dati nel buffer di ricezione dello stack di rete.

procedure ClientDisconnect;

Rileva quando l'handle di connessione del client commuta su **nil**.

procedure Disconnetti;

Effettua il reset degli handle connettivi, è una procedura di tipo CS anche se non indicato.

procedure InterPingTimer(Sender: TObject);

Temporizza l'invio dei segnali di ping dello stack di rete.

procedure InviaNomeCS(stringa: String);

Produce il sending del field nome side-to-side in modalità peer-to-peer sul relativo slot dello stack di rete; è di tipo CS.

procedure InviaSituazione;

E' una procedura server/client che invia la stringa contenente la RAS (Reduced Array State) corrente in modalità peer-to-peer sul relativo slot unidirezionale dello stack di rete.

procedure InviaVittoria(playerid: Turno);

Invia sul relativo slot unidirezionale dello stack l'impulso di vittoria partita da parte di uno dei due giocatori.

procedure MossaDalClientChange(Sender: TObject);

Gestisce l'handle di pervenimento e redistribuzione degli receive-events dello stack lato server e ne esegue i relativi comandi.

procedure MossaDalServerChange(Sender: TObject);

Gestisce l'handle di pervenimento e redistribuzione degli receive-events dello stack lato client e ne esegue i relativi comandi.

procedure PingCS;

Invia l'impulso di Ping sullo stack bilaterale dell'end-to-end in modalità peer-to-peer; è una procedura di tipo CS.

procedure PongCS;

Invia l'impulso di risposta Pong sullo stack bilaterale dell'end-to-end in modalità peer-to-peer; è una procedura di tipo CS.

procedure ServerConnect(AThread: TIdPeerThread);

Controlla l'attivazione dell'handle connettivo del server in multithread, assegna il primo thread alla variabile globale **Utente** e scarta i thread successivi restituendo sullo stack un impulso di **NA** (Not Acknowledgement) che sul client-end provoca la immediata disconnessione segnalata dal relativo messaggio d'errore.

procedure ServerDisconnect(AThread: TIdPeerThread);

Rileva la commutazione a **nil** dell'handle di rete lato server, resetta il thread attivo e reinizializza la variabile utente predisponendola ad una nuova connessione.

procedure ServerExecute(AThread: TIdPeerThread);

Attiva la ricezione del buffer lato server relativamente al thread puntato da **Utente** nel protocol stack, se il medesimo risulta non vuoto e invia il controllo alla procedura *MossaDalClient*.

procedure TimerTimeoutTimer(Sender: TObject);

Rileva il timing-out della connessione peer-to-peer per assenza di impulsi di pong e disconnette l'offending-side.

Descrizione dello stack di rete

La struttura utilizzata per l'application layer della connessione è quella del command-based stack a due livelli con carattere riservato per l'implementazione futura dell'*encoded checksum* per il controllo d'errore di trasmissione; ogni comando risiede in uno slot il quale, quando attivato dall'apposito comando di rete, esegue le corrispondenti procedure.

Il Formato del command slot a lunghezza variabile dello stack è:

| | | | | |
|---|----|-------|-------|-------|
| . | Cc | X_1 | | X_n |
|---|----|-------|-------|-------|

- Il primo (.) carattere contiene sempre un punto ma è destinato a contenere un *encoded checksum* in eventuali versioni future;
- Il secondo (Cc) identifica il comando come definito nelle tabelle sottoriportate;
- I successivi n byte, da X_1 a X_n , costituiscono il body del messaggio e quindi i parametri del comando.

Il protocollo di comunicazione è il TCP della suite protocollare IPV4.0 pentalivello la cui gestione è delegata ai due componenti, inclusi nelle librerie Indy (Internet Direct) 8.0 di WinShoes, di tipo IdTelnetClient, per il lato client, e IdTCPServer, per il lato server. L'application layer è quindi il protocollo Telnet/TCP/IP in ASCII ClearText. Lo scopo è realizzare una applicazione peer-to-peer client/server su un canale socket multithread che garantisce l'assoluta stabilità dell'astrazione point-to-point del connection channel. L'implementazione di un connection control su TCP (Ping/Pong) è una strategia designata al fine di controllare direttamente il mezzo medesimo. E' prevista per successivi aggiornamenti l'affiancamento o la sostituzione di questa tecnica con una di controllo implementata su ICMP/IP. Lo stack così ottenuto si presta molto facilmente a modifiche che contemplino l'aumento dei clients o la sostituzione o l'ampliamento del kit di comandi di rete.

Specifiche dello stack lato client (ricezione)

| Slot | Command Char | Descrizione |
|------|--------------|---|
| 1 | A | Situazione delle matrici carte: X_1 =Assegnazione del turno, X_2 =Carta scartata, X_3 =Numero delle carte del giocatore sul server, X_4 =Seme attuale, $X_5..X_n$ =matrice delle carte in possesso del giocatore sul client. |
| 2 | B | Assegna vittoria singola partita: X_1 =Vincitore. |
| 3 | C | Dati di classifica parziale. |
| 4 | D | Messaggio di Chat. |
| 5 | E | Nome. |
| 6 | F | Ping (implementato sul bottom level dello stack). |
| 7 | G | Pong (implementato sul bottom level dello stack). |
| 9 | I | Fine delle carte del mazzo. |
| 61 | } | Chiusura della connessione lato server |
| 62 | ~ | Server Busy |

Specifiche dello stack lato server (ricezione)

| Slot | Command Char | Descrizione |
|------|--------------|---|
| 1 | A | Richiesta di pescaggio carta dal mazzo. |
| 2 | B | Richiesta di gioco (scarto) di una carta. |
| 3 | C | Sconfitta del client. |
| 4 | D | Messaggio di Chat. |
| 5 | E | Nome. |
| 6 | F | Ping. |
| 7 | G | Pong. |
| 8 | H | Seme attuale scelto dal cliente. |
| 9 | I | Istanza di rinvio delle matrici carte. |

Elenco dei bug di Delphi riscontrati

| Bug | Proposta di soluzione |
|--|---|
| 1 Scorretta gestione del procedure stack nei salti procedurali senza ritorno; il pointer non risale causando l'allocazione permanente delle procedure dimesse e quindi si ha un inutile spreco di memoria dovuto alle reiterate istanze procedurali che vengono dal compilatore intese di tipo Child. L'iterazione dell'errore può provocare il procedure stack overflow. | Modifica manuale, a mezzo routine di servizio in assembler, della posizione del procedure stack point, effettuando il kill di fatto della parte procedurale seguente l'istruzione di call della procedura figlia. |
| 2 Scorretto management della memoria protetta e bug nell'overlay della memoria utente, sporadici errori nella gestione del paging. Tali grave carenze non consentono un versatile utilizzo di Delphi laddove necessiti una allocazione dinamica e superdinamica della memoria. I bug provocano errori che vengono segnalati con exception di tipo "Access Violation" o "Abstract Error". | Allocazione statica degli spazi mnemonici da effettuarsi in fase dichiarativa con il rischio di disperdere innumerevoli risorse che possono, alla lunga, rimanere inutilizzate. |
| 3 Scorretto comportamento nel gestire event paralleli, durante l'event call contemporaneo di un evento sincrono e di uno asincrono (proveniente dalla rete ad esempio). | Sincronizzazione degli eventi asincroni ed assegnazione di un sender. |
| 4 Inesistenza di procedure di Delay che consentano di chiamare ritardi programmatici utili sia per la sincronizzazione (e quindi risoluzione di time hazard) sia per la umanizzazione delle procedure. | Scrittura di una procedura apposita utilizzando le API di Windows che inducono un Idleing programmato di n millisecondi. |