

trabajo práctico obligatorio final

Programación II | 2018

Gatica Miguel Elías

18 de febrero de 2021

Profesor: Claudio Álvarez | Enc.Trab.Pcos: Raquel Bros

Indice

[1 DER (Diagrama Entidad Relación) 1](#_Toc64539462)

[2 Estructuras 2](#_Toc64539463)

[2.1 Librería Auxiliar de Estructuras 2](#_Toc64539464)

[2.2 TDA 3](#_Toc64539465)

[2.2.1 Árbol AVL 3](#_Toc64539466)

[2.2.2 Árbol Trinario AVL 26](#_Toc64539467)

[2.2.3 Colas 48](#_Toc64539468)

[2.2.4 Hash 52](#_Toc64539469)

[2.2.4.1 Type Hash 56](#_Toc64539470)

[2.2.5 Lista Simple Enlace 56](#_Toc64539471)

[2.2.6 Lista Doble Enlace 64](#_Toc64539472)

[2.2.7 Lista Parcial Doble Enlace 74](#_Toc64539473)

[2.3 ME 82](#_Toc64539474)

[2.3.1 Jugadores 82](#_Toc64539475)

[2.3.1.1 Type Jugador 88](#_Toc64539476)

[2.3.2 Cuenta Corriente 89](#_Toc64539477)

[2.3.2.1 Type Almacén 100](#_Toc64539478)

[2.3.3 Juego 102](#_Toc64539479)

[2.3.3.1 Type Juego 110](#_Toc64539480)

[2.3.4 Apuestas 111](#_Toc64539481)

[2.3.4.1 Type Apuesta 119](#_Toc64539482)

[2.3.5 Ganadores 120](#_Toc64539483)

[2.3.5.1 Type Ganador 128](#_Toc64539484)

[3 Proyecto 129](#_Toc64539485)

[3.1 Librería Auxiliar Juego 129](#_Toc64539486)

[3.2 Formularios 185](#_Toc64539487)

[3.2.1 Formulario Login 185](#_Toc64539488)

[3.2.2 Formulario Croupier 201](#_Toc64539489)

[3.2.3 Formulario Jugadores 214](#_Toc64539490)

[3.2.4 Formulario Juego 231](#_Toc64539491)

[3.2.5 Formulario Apostar 327](#_Toc64539492)

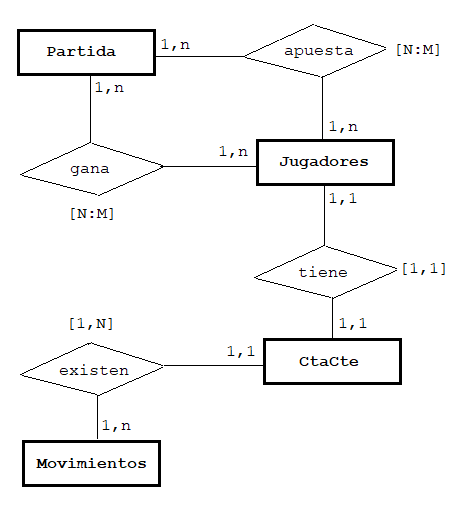
[3.2.6 Formulario Ficticios 339](#_Toc64539493)

[3.2.7 Formulario Premios 350](#_Toc64539494)

[3.2.8 Formulario Listados 353](#_Toc64539495)

[3.2.9 Formulario Premio/Bloqueo 363](#_Toc64539496)

# DER (Diagrama Entidad Relación)



# Estructuras

## Librería Auxiliar de Estructuras

unit Lib\_Auxiliar;

interface

uses

Sysutils;

Const

ApuestaMaxima = 1000;

ApuestaMinima = 50;

PosNula = -1;

IniFicticios = 0;

ClaveNula ='';

Fict\_IniNick = 'X\_';

type

tClave = string[15];

tPos = longint;

tNivel = 0.. maxint;

tProfundidad = 0.. maxint;

tCantidad = 0.. maxint;

Implementation

END.

## TDA

### Árbol AVL

unit LO\_ArbolAVL;

{

Un árbol AVL es aquel que por cada nodo la altura del subárbol izquierdo menos la altura del subárbol derecho difiera a lo sumo en 1

Antes de insertar hace un buscar, que sino encuentra la clave devuelvo posicion del padre

Es por eso que al insertar el pos parametrizado corresponde a la posicion fisica del padre y parado en el ve si corresponde que sea el hijo izquierdo o el hijo derecho

Una vez insertado el nodo, pregunto por el balanceo del árbol.

Se pregunta por el balanceo de cada nodo desde el nodo padre del recién insertado hasta la raíz

Un nodo esta balanceado si su factor de equilibrio es -1, 1 o 0.., teniendo presenete que el factor de equilibrio de un árbol binario es la diferencia en altura entre los subárboles derecho e izquierdo.

Para ello la funcion 'factor de equilibrio' va a verificar que el nodo este equilibrado. Si un nodo esta desequilibrado, se invoca a ‘Caso Desquilibrio’ donde verifica que tipo de rotación se llevara a cabo para balancear dicho nodo

Si un nodo esta desequilibrado, se invoca a ‘Caso Desquilibrio’ donde verifica que tipo de rotación se llevara a cabo para balancear dicho nodo.

Si el factor de equilibrio del nodo desbalanceado es positivo, está cargado a la izquierda..., se revisa con hijo izquierdo

Si el factor de equilibrio del hijo izquierdo es positivo, se lleva a cabo una RotacionSimpleDerecha sino una RotacionCompuestaDerecha

En el caso de que el factor de equilibrio del nodo desbalanceado es negativo, significa que está cargado a la derecha...,

se revisa con hijo derecho. Si el factor de equilibrio del hijo derecho es positivo, se hace una RotacionCompuestaIzquierda sino

una RotacionSimpleIzquierda

}

interface

USES

SysUtils, Lib\_Auxiliar;

TYPE

tNodoIndice = record

clave:tClave;

padre, hi, hd:tPos;

nivel:integer;

PosEnDatos:tPos;

end;

tControlArbol= record

Raiz,Borrados: tPos;

Cantidad, NivelMax: TCantidad;

ficticio: tCantidad;

end;

tArchivoNodos= file of tNodoIndice;

tArchivoControlArbol= file of tControlArbol;

tArbol= record

D:tArchivoNodos;

C:tArchivoControlArbol;

end;

Procedure CrearMe\_Arbol (var Me:tArbol;

\_NombreArchivoControl,\_NombreArchivoIndice:String;

\_Directorio:String);

Procedure AbrirMe\_Arbol(var Me:tArbol);

Procedure CerrarMe\_Arbol(var Me:tArbol);

Procedure DestruirMe\_Arbol(var Me:tArbol);

Function Arbol\_Vacio(var Me: tArbol):Boolean;

Procedure InsertarNodo\_Arbol(var Me:tArbol; nodo:tNodoIndice;pos:tPos; tipo:boolean);

Procedure EliminarNodo\_Arbol(var Me:tArbol; pos:tPos);

Procedure ModificarNodo\_Arbol(var Me: tArbol; RegInd:tNodoIndice;Pos:tPos);

Function BuscarNodo\_Arbol (var Me:tArbol; clave:tClave;var pos:tPos):boolean;

Procedure CapturarNodo\_Arbol(var Me: tArbol; var RegInd:tNodoIndice; Pos:tPos);

Function PosNula\_Arbol(Var Me:tArbol):tPos;

Function Raiz\_Arbol(var Me: tArbol): tPos;

Function Padre\_Arbol(var Me: tArbol; var Reg: tNodoIndice): tPos;

Function HijoDer\_Arbol(var Me: tArbol; pos: tPos): tPos;

Function HijoIzq\_Arbol(var Me: tArbol; pos: tPos): tPos;

{---------------------------(BALANCEO)----------------------------------------}

Procedure RotacionSimpleIzquierda(var me: tArbol; PosNodo: tPos);

Procedure RotacionSimpleDerecha(var me: tArbol; PosNodo: tPos);

Procedure RotacionCompuestaDerecha(var me: tArbol; PosNodo: tPos);

Procedure RotacionCompuestaIzquierda(var me: tArbol; PosNodo: tPos);

Function ProfundidadNodo\_Arbol(me: tArbol; raiz: tPos): integer;

procedure ProfundidadTotal\_Arbol(var me:tArbol; Raiz:tPos; var Profundidad:tProfundidad);

Procedure CasoDesequilibrio\_Arbol(me: tArbol; PosNodo: tPos);

Function FactorEquilibrio\_Arbol(me: tArbol; PosNodo: tPos): integer;

Procedure ActualizarNiveles\_Arbol(var me: tArbol; Pos:TPos);

procedure Balancear\_Arbol(var me: tArbol; Pos:tpos);

{-----------------------------------------------------------------------------}

Function UltFicticio\_Arbol(var Me: tArbol): tCantidad;

Function CantidadNodos\_Arbol(var Me: tArbol): tCantidad;

Function ObtenerNivel\_Arbol(var Me:tArbol; clave:tClave): tCantidad;

VAR

Me: tArbol;

Implementation

{-----------------------------------------------------------------------------------}

Procedure CrearMe\_Arbol (var Me:tArbol; \_NombreArchivoControl,\_NombreArchivoIndice:String;\_Directorio:String);

Var

ioC,ioD:integer;

Rc: tControlArbol;

begin

//AssignFile reemplaza el procedimiento Assign que estaba disponible en las primeras versiones de Delphi.

assignFile (Me.C, \_Directorio + \_NombreArchivoControl);

assignFile (Me.D, \_Directorio + \_NombreArchivoIndice);

{$i-}

reset(Me.C); ioC:=IoResult; //Si IoResult es cero, la operacion es exitosa

reset(Me.D); ioD:=IoResult;

if (IoC<>0) or (ioD<>0) then

begin

Rewrite(Me.C);

ReWrite(Me.D);

RC.Raiz:=Lib\_Auxiliar.PosNula;

RC.Borrados:=Lib\_Auxiliar.PosNula;

RC.ficticio:= Lib\_Auxiliar.IniFicticios;

Rc.Cantidad:= 0;

Write(Me.C,RC);

end;

Close(Me.D);

Close(Me.C);

{$I+}

end;

{-----------------------------------------------------------------------------------}

procedure AbrirMe\_Arbol(var Me:tArbol);

begin

Reset(Me.C);

Reset(Me.D);

end;

{-----------------------------------------------------------------------------------}

procedure CerrarMe\_Arbol(var Me:tArbol);

begin

Close(Me.D);

Close(Me.C);

end;

{-----------------------------------------------------------------------------------}

procedure DestruirMe\_Arbol(var Me:tArbol);

begin

Erase(Me.D);

Erase(Me.C);

end;

{-----------------------------------------------------------------------------------}

Function Arbol\_Vacio(var Me: tArbol):Boolean;

var

Rc:tControlArbol;

begin

Seek(Me.C,0); Read(Me.c, Rc);

Arbol\_Vacio:= (Rc.Raiz = Lib\_Auxiliar.PosNula) ;

end;

{-----------------------------------------------------------------------------------}

Procedure InsertarNodo\_Arbol(var Me:tArbol; nodo:tNodoIndice; pos:tPos; tipo:boolean);

{Antes de insertar hace un buscar, que sino encuentra la clave devuelvo posicion del padre. Es por eso que al insertar el pos parametrizado corresponde a la posicion fisica del padre y parado en el ve si corresponde que sea el hijo izquierdo o el hijo derecho}

var

posnueva:tPos;

reg,rd:tNodoIndice;

RC:tControlArbol;

begin

seek(Me.C,0); read(Me.C,RC);

if (RC.borrados = Lib\_Auxiliar.PosNula) then //no hay borrados

begin

posnueva:=filesize(Me.D);

end

else //hay borrados

begin

Posnueva:= RC.Borrados;

seek(Me.D,posnueva); read(Me.D,rD);

RC.Borrados:= rD.hi;

end;

//Grabo al final ...

if (RC.Raiz=Lib\_Auxiliar.PosNula) then//inserto en árbol vacio

begin

RC.Raiz:= posnueva;

nodo.padre:= Lib\_Auxiliar.PosNula;

nodo.hi:= Lib\_Auxiliar.PosNula;

nodo.hd:= Lib\_Auxiliar.PosNula;

nodo.nivel:= 1;

end

else //inserto como hoja

begin

seek(Me.D,pos); read(Me.D,reg);

nodo.padre:= pos;

nodo.hd:= Lib\_Auxiliar.PosNula;

nodo.hi:= Lib\_Auxiliar.PosNula;

nodo.nivel:= Reg.Nivel+1;

{Si la clave del nodo a insertar es menor a la clave del nodo padre, entonces la posicion nueva la tendra su hijo izquierdo (su hijo izquierdo estará al final físico),

Sino, si la clave del nodo a insertar es mayor a la clave del nodo padre, entonces la posicion nueva la tendra su hijo derecho (su hijo derecho estará al final físico)}

if (nodo.clave < reg.clave) then

reg.hi:=posnueva

else

reg.hd:=posnueva;

//grabo nodo padre con nodo hijo actualizado

seek(Me.d,pos); write(Me.d,reg);

end;

//Si la profuncididad del nodo es mayor a la prfundidad en el control actualizo.

If nodo.nivel > RC.NivelMax then

RC.NivelMax := nodo.nivel;

//si tipo (jugador) es true es real sino ficticio

if not (tipo) then

RC.ficticio:= RC.ficticio+1;

rc.cantidad:=rc.cantidad+1;

//se graba el elemento nuevo insertado en el archivo

seek(Me.d,posnueva); write(Me.d,nodo);

//se actualiza el registro control

seek(Me.c,0); write(Me.c,rc);

//invoco al balanceo del árbol

Balancear\_Arbol(me, PosNueva);

end;

{-----------------------------------------------------------------------------------}

Procedure EliminarNodo\_Arbol(var Me:tArbol; pos:tPos);

var

RD:tNodoIndice;

RC:tControlArbol;

rAux,RHD,RDP,RHI:tNodoIndice;

posAux:tPos;

begin

seek(Me.c,0); read(Me.c,rc);

seek(Me.d,pos); read(Me.d,rd); //Capturo el registro a eliminar

//no tiene padre, no tiene hijos

if (pos=rc.raiz) and (rd.hi=Lib\_Auxiliar.PosNula) and (rd.hd=Lib\_Auxiliar.PosNula) then

Rc.Raiz:=Lib\_Auxiliar.PosNula

else

begin

if (rd.hi=Lib\_Auxiliar.PosNula) and (rd.hd=Lib\_Auxiliar.PosNula) then

//Tiene padre, no tiene hijos. Eliminar de una hoja

begin

Seek(Me.D,rd.padre);read(Me.d,rdp);

If rdp.hi=pos then

rdp.hi:=Lib\_Auxiliar.PosNula

else

rdp.hd:=Lib\_Auxiliar.PosNula;

Seek(Me.d,rd.padre);write(Me.d,RDP);

end

else

begin

if (rd.hi <> Lib\_Auxiliar.PosNula) and (rd.hd <> Lib\_Auxiliar.PosNula) then

//Caso general, tiene padre y tiene hijos...

begin

If pos<>RC.raiz then //...no es raiz

begin

Seek(Me.d,rd.padre);read(Me.d,rdp);

If rdp.hi=pos then

rdp.hi:=rd.hd

else

rdp.hd:=rd.hd;

Seek(Me.d,RD.padre);Write(Me.d,rdp);

end

else

RC.raiz:=rd.hd;

Seek(Me.d,rd.hd);read(Me.d,RHD);

RHD.padre:=RD.padre;

posAux:=RD.hd;

rAux:=RHD;

While rAux.hi<>Lib\_Auxiliar.PosNula do

begin

Seek(Me.d,posAux);read(Me.d,RAux);

If raux.hi<>Lib\_Auxiliar.PosNula then

posaux:=rAux.hi;

end;

Seek(Me.d,RD.hi);Read(Me.d,RHI);

RHI.padre:=posAux;

RAux.hi:=RD.hi;

Seek(Me.d,RD.hi);Write(Me.d,RHI);

Seek(Me.d,posAux);Write(Me.d,raux);

If posAux<>RD.hd then

begin

Seek(Me.d,RD.hd);Write(Me.d,RHD);

end;

end

else

begin

If pos<>RC.Raiz then

begin

Seek(Me.D,RD.padre);Read(Me.D,RDP);

If RDP.hi=pos then

begin

If RD.hi=Lib\_Auxiliar.PosNula then

begin

Rdp.hi:=RD.hd;

Seek(Me.D,RD.hd);read(Me.d,RHD);

RHD.padre:=RD.padre;

Seek(Me.d,RD.hd);write(Me.d,RHD);

end

else

begin

Rdp.hi:=RD.hi;

Seek(Me.D,RD.hi);read(Me.d,RHI);

RHI.padre:=RD.padre;

Seek(Me.d,RD.hi);write(Me.d,RHI);

end

end

else

If RD.hi=Lib\_Auxiliar.PosNula then

begin

Rdp.hd:=RD.hd;

Seek(Me.D,RD.hd);read(Me.d,RHD);

RHD.padre:=RD.padre;

Seek(Me.d,RD.hd);write(Me.d,RHD);

end

else

begin

Rdp.hd:=RD.hi;

Seek(Me.D,RD.hi);read(Me.d,RHI);

RHI.padre:=RD.padre;

Seek(Me.d,RD.hi);write(Me.d,RHI);

end;

Seek(Me.d,rd.padre);write(Me.d,rdp);

end

else

If RD.hi=Lib\_Auxiliar.PosNula then

RC.Raiz:=RD.hd

else

RC.raiz:=RD.hi;

end;

end;

end;

RD.hi:=RC.borrados;

RD.hd:=RC.borrados;

RC.borrados:=pos;

RC.Cantidad:=RC.Cantidad-1;

Seek(Me.c,0);write(Me.c,rc); //Grabo la cabecera de control

end;

{-----------------------------------------------------------------------------------}

Procedure ModificarNodo\_Arbol(var Me: tArbol; RegInd:tNodoIndice;Pos:tPos);

begin

seek(Me.D,pos); write(Me.D,RegInd);

end;

{-----------------------------------------------------------------------------------}

Function BuscarNodo\_Arbol (var Me:tArbol; clave:tClave;var pos:tPos):boolean;

//El buscar sino encuentra la clave devuelvo la posicion del padre

var

reg:tNodoIndice;

RC:tControlArbol;

encont:boolean;

posPadre:tpos;

begin

seek(Me.c,0); read(Me.c,rc);

pos:=rc.raiz;

encont:=false;

posPadre:=Lib\_Auxiliar.PosNula;

while (not encont) and (pos<>Lib\_Auxiliar.PosNula) do

begin

seek(Me.d,pos); read(Me.d,reg);

if (clave=reg.clave) then

encont:=true

else

begin

if (reg.clave>clave) then

begin

posPadre:=pos;

pos:=reg.hi;

end

else

begin

posPadre:=pos;

pos:=reg.hd;

end;

end;

end;

if (not encont) then

pos:=posPadre;

BuscarNodo\_Arbol:=encont;

end;

{-----------------------------------------------------------------------------------}

Procedure CapturarNodo\_Arbol(var Me: tArbol; var RegInd:tNodoIndice; Pos:tPos);

begin

seek(Me.D,Pos); Read(Me.D,RegInd);

end;

{-----------------------------------------------------------------------------------}

function PosNula\_Arbol(Var Me:tArbol):tPos;

begin

PosNula\_Arbol:= Lib\_Auxiliar.PosNula;

end;

{-----------------------------------------------------------------------------------}

function Raiz\_Arbol(var Me: tArbol): tPos;

var

Rc: tControlArbol;

begin

Seek(Me.c, 0);

Read(Me.c, Rc);

Raiz\_Arbol:= Rc.Raiz;

end;

{-----------------------------------------------------------------------------------}

Function Padre\_Arbol(var Me: tArbol; var Reg: tNodoIndice): tPos;

begin

Padre\_Arbol:= Reg.padre;

end;

{-----------------------------------------------------------------------------------}

function HijoDer\_Arbol(var Me: tArbol; pos: tPos): tPos;

var

reg: tNodoIndice;

begin

seek(me.D,pos); read(me.D,reg);

HijoDer\_Arbol:= reg.hd;

end;

{-----------------------------------------------------------------------------------}

function HijoIzq\_Arbol(var Me: tArbol; pos: tPos): tPos;

var

reg: tNodoIndice;

begin

seek(me.D,pos); read(me.D,reg);

HijoIzq\_Arbol:= reg.hi;

end;

{-----------------------------------------------------------------------------------}

//Rotacion en arbol tambien conocido como 'Rotacion Derecha-Derecha'

Procedure RotacionSimpleIzquierda (var me: tArbol; PosNodo: tPos);

Var

NodoArriba,NodoAbajo,NodoAnterior:tNodoIndice;

RC:tControlArbol;

PosAux:tPos;

begin

Seek (me.D,PosNodo); Read (me.D,NodoArriba);

Seek (me.D,NodoArriba.hd); Read (me.D,NodoAbajo);

Seek (me.C,0);Read(me.C,RC);

//Cambio enlaces

PosAux:=NodoAbajo.hi;

NodoAbajo.padre:=NodoArriba.padre;

NodoAbajo.hi:=PosNodo;

NodoArriba.padre:=NodoArriba.hd;

NodoArriba.hd := PosAux; ;

//Si el desbalanceado no era raíz

If NodoAbajo.padre <> Lib\_Auxiliar.PosNula then

begin

//Me paro en el padre del hijo desbalanceado, y le pongo como hijo al que era su nieto

Seek (Me.D,NodoAbajo.padre); Read (Me.D,NodoAnterior);

If NodoAnterior.hd = PosNodo then

NodoAnterior.hd :=NodoArriba.padre

else

NodoAnterior.hi :=NodoArriba.padre;

Seek (Me.D,NodoAbajo.padre); Write (Me.D,NodoAnterior);

end;

//Cambio niveles

NodoAbajo.nivel:=NodoAbajo.nivel-1;

NodoArriba.nivel := NodoArriba.nivel+1;

//Verifico que el del problema no haya sido la raiz

//Anteriormeente a NodoArriba.padre le da NodoAbajo.hdecho. Por lo tanto, pone como nueva raiz

//al hijo derecho del que estaba desbalanceado.

If PosNodo=RC.Raiz then

RC.raiz := NodoArriba.padre;

Seek (Me.D,PosNodo); Write (Me.D,NodoArriba);

Seek (Me.D,NodoArriba.padre); Write (Me.D,NodoAbajo);

Seek (Me.C,0);Write(Me.C,RC);

{Llamo a Actualizar Nivel para actualizar todos los nodos por debajo del padre del nodo que estaba desbalanceado

porque ese nodo sigue teniendo el nivel original}

ActualizarNiveles\_Arbol(ME,PosNodo);

end;

{-----------------------------------------------------------------------------------}

//Rotacion en arbol tambien conocido como 'Rotacion Izquierda-Izquierda'

Procedure RotacionSimpleDerecha(var me: tArbol; PosNodo: tPos);

Var

NodoArriba,NodoAbajo,NodoAnterior:tNodoIndice;

RC:tControlArbol;

PosAux:tPos;

begin

Seek (me.D,PosNodo); Read (me.D,NodoArriba);

Seek (me.D,NodoArriba.hi); Read (me.D,NodoAbajo);

Seek (me.C,0);Read(me.C,RC);

{Cambio enlaces}

PosAux:=NodoAbajo.hd;

NodoAbajo.padre:=NodoArriba.padre;

NodoAbajo.hd:=PosNodo;

NodoArriba.padre:=NodoArriba.hi;

NodoArriba.hi := PosAux; ;

If NodoAbajo.padre <> Lib\_Auxiliar.PosNula then

begin

Seek (Me.D,NodoAbajo.padre); Read (Me.D,NodoAnterior);

If NodoAnterior.hd = PosNodo then

NodoAnterior.hd :=NodoArriba.padre

else

NodoAnterior.hi :=NodoArriba.padre;

Seek (Me.D,NodoAbajo.padre); Write (Me.D,NodoAnterior);

end;

{Cambio el campo nivel}

NodoAbajo.nivel:=NodoAbajo.nivel-1;

NodoArriba.nivel := NodoArriba.nivel+1;

{Verifico que el del drama no haya sido la raiz}

If PosNodo=RC.Raiz then

RC.raiz := NodoArriba.padre;

Seek (Me.D,PosNodo); Write (Me.D,NodoArriba);

Seek (Me.D,NodoArriba.padre); Write (Me.D,NodoAbajo);

Seek (Me.C,0);Write(Me.C,RC);

{Llamo a Actualizar Nivel para actualizar todos los nodos por debajo del padre del nodo que estaba desbalanceado

porque ese nodo sigue teniendo el nivel original}

ActualizarNiveles\_Arbol(ME,PosNodo);

end;

{-----------------------------------------------------------------------------------}

//Rotacion en arbol tambien conocido como 'Rotacion Izquierda-Derecha'

Procedure RotacionCompuestaDerecha(var me: tArbol; PosNodo: tPos);

Var

NodoArriba,NodoMedio,NodoAbajo,NodoAnterior:tNodoIndice;

RC:tControlArbol;

PosAux,PosMedio,PosAbajo:tPos;

begin

Seek (Me.D,PosNodo); Read (Me.D,NodoArriba);

Seek (Me.D,NodoArriba.hi); Read (Me.D,NodoMedio);

Seek (Me.D,NodoMedio.hd); Read (Me.D,NodoAbajo);

Seek (Me.C,0);Read(Me.C,RC);

PosMedio:=NodoArriba.hi;

PosAbajo:=NodoMedio.hd;

PosAux:=NodoArriba.padre;

NodoArriba.padre:=NodoMedio.hd;

NodoArriba.hi:=NodoAbajo.hd;

NodoMedio.padre:=NodoMedio.hd;

NodoMedio.hd:= NodoAbajo.hi;

NodoAbajo.padre:=PosAux;

NodoAbajo.hd:=PosNodo;

NodoAbajo.hi:=PosMedio;

If NodoAbajo.padre <> Lib\_Auxiliar.PosNula then

begin

Seek (Me.D,NodoAbajo.padre); Read (Me.D,NodoAnterior);

If NodoAnterior.hd = PosNodo then

NodoAnterior.hd :=PosAbajo

else

NodoAnterior.hi :=PosAbajo;

Seek (Me.D,NodoAbajo.padre); Write (Me.D,NodoAnterior);

end;

{Cambio el campo nivel}

NodoAbajo.nivel:=NodoAbajo.nivel-2;

NodoArriba.nivel:= NodoArriba.nivel+1;

{Verifico que el del drama no haya sido la raiz}

If PosNodo=RC.Raiz then

RC.raiz := NodoArriba.padre;

Seek (Me.D,PosNodo); write (Me.D,NodoArriba);

Seek (Me.D,PosMedio); write (Me.D,NodoMedio);

Seek (Me.D,PosAbajo); write (Me.D,NodoAbajo);

Seek (Me.C,0);Write (Me.C,RC);

{Llamo a Actualizar Nivel para actualizar todos los nodos por debajo del padre del nodo que estaba desbalanceado

porque ese nodo sigue teniendo el nivel original}

ActualizarNiveles\_Arbol(ME,PosNodo);

end;

{-----------------------------------------------------------------------------------}

//Rotacion en arbol tambien conocido como 'Rotacion Derecha-Izquierda'

Procedure RotacionCompuestaIzquierda(var me: tArbol; PosNodo: tPos);

Var

NodoArriba,NodoMedio,NodoAbajo,NodoAnterior:tNodoIndice;

RC:tControlArbol;

PosAux,PosMedio,PosAbajo:tPos;

begin

Seek (Me.D,PosNodo); Read (Me.D,NodoArriba);

Seek (Me.D,NodoArriba.hd); Read (Me.D,NodoMedio);

Seek (Me.D,NodoMedio.hi); Read (Me.D,NodoAbajo);

Seek (Me.C,0);Read(Me.C,RC);

PosMedio:=NodoArriba.hd;

PosAbajo:=NodoMedio.hi;

PosAux:=NodoArriba.padre;

NodoArriba.padre:=NodoMedio.hi;

NodoArriba.hd:=NodoAbajo.hi;

NodoMedio.padre:=NodoMedio.hi;

NodoMedio.hi:= NodoAbajo.hd;

NodoAbajo.padre:=PosAux;

NodoAbajo.hi:=PosNodo;

NodoAbajo.hd:=PosMedio;

If NodoAbajo.padre <> Lib\_Auxiliar.PosNula then

begin

Seek (Me.D,NodoAbajo.padre); Read (Me.D,NodoAnterior);

If NodoAnterior.hd = PosNodo then

NodoAnterior.hd :=PosAbajo

Else

NodoAnterior.hi :=PosAbajo;

Seek (Me.D,NodoAbajo.padre); Write (Me.D,NodoAnterior);

end;

{Cambio el campo nivel}

NodoAbajo.nivel:=NodoAbajo.nivel-2;

NodoArriba.nivel:= NodoArriba.nivel+1;

{Verifico que el del drama no haya sido la raiz}

If PosNodo=RC.Raiz then

RC.raiz := NodoArriba.padre;

Seek (Me.D,PosNodo); write (Me.D,NodoArriba);

Seek (Me.D,PosMedio); write (Me.D,NodoMedio);

Seek (Me.D,PosAbajo); write (Me.D,NodoAbajo);

Seek (Me.C,0);Write (Me.C,RC);

{Llamo a Actualizar Nivel para actualizar todos los nodos por debajo del padre del nodo que estaba desbalanceado

porque ese nodo sigue teniendo el nivel original}

ActualizarNiveles\_Arbol(ME,PosNodo);

end;

{-----------------------------------------------------------------------------------}

Function ProfundidadNodo\_Arbol(me: tArbol; raiz: tPos): integer;

var

Profundidad:tProfundidad;

Nodo:tNodoIndice;

Begin

//Dada una posición, devuelve la profundidad (nivel) en el que se encuentra

Profundidad := 0;

CapturarNodo\_Arbol(me, nodo, raiz);

ProfundidadTotal\_Arbol(me, raiz, Profundidad);

result := (Profundidad - nodo.nivel) + 1;

End;

{-----------------------------------------------------------------------------------}

procedure ProfundidadTotal\_Arbol(var me:tArbol; Raiz:tPos; var Profundidad:tProfundidad);

var

Nodo: tNodoIndice;

begin

If Raiz <> Lib\_Auxiliar.PosNula then

begin

// Primero recursivo tendiendo a la Izquierda

ProfundidadTotal\_Arbol(me, HijoIzq\_Arbol(me, raiz), Profundidad);

// Recursividad tendiendo a la Derecha.

ProfundidadTotal\_Arbol(me, HijoDer\_Arbol(me, raiz), Profundidad);

// Guardo en Nodo el nodo indice.

CapturarNodo\_Arbol(me, nodo, raiz);

If Nodo.nivel > Profundidad then

Profundidad := Nodo.nivel;

end;

end;

{-----------------------------------------------------------------------------------}

Procedure CasoDesequilibrio\_Arbol(me: tArbol; PosNodo: tPos);

var

Nodo: tNodoIndice;

begin

CapturarNodo\_Arbol(me, Nodo, PosNodo);

//Si el factor de equilibrio es positivo, esta cargado a la izquierda..., reviso con hijo izquierdo

If FactorEquilibrio\_Arbol(me, PosNodo) >= 0 then

begin

If FactorEquilibrio\_Arbol(me, Nodo.hi) >= 0 then //Si el factor de equilibrio del hijo izquierdo es positivo, hago una RotacionSimpleDerecha

RotacionSimpleDerecha(me, PosNodo)

else //...si es negativo, hago una RotacionCompuestaDerecha

RotacionCompuestaDerecha(me, PosNodo);

end

else //Si el factor de equilibrio es negativo, esta cargado a la derecha..., reviso con hijo derecho

begin

If FactorEquilibrio\_Arbol(me, Nodo.hd) >= 0 then //Si el factor de equilibrio del hijo derecho es positivo, hago una RotacionCompuestaIzquierda

RotacionCompuestaIzquierda(me, PosNodo)

else //...si es negativo, hago una RotacionSimpleIzquierda

RotacionSimpleIzquierda(me, PosNodo);

end;

end;

{-----------------------------------------------------------------------------------}

Function FactorEquilibrio\_Arbol(me: tArbol; PosNodo: tPos): integer;

var

Nodo: tNodoIndice;

Ti, Td: integer;

begin

If PosNodo = Lib\_Auxiliar.PosNula then

FactorEquilibrio\_Arbol := 0

else

begin

//Guardo en Nodo el nodo indice.

CapturarNodo\_Arbol(me, Nodo, PosNodo);

// Calculo la profundidad de ambos hijos.

If Nodo.hi <> Lib\_Auxiliar.PosNula then

Ti := ProfundidadNodo\_Arbol(me, Nodo.hi)

else

Ti := 0;

if Nodo.hd <> Lib\_Auxiliar.PosNula then

Td := ProfundidadNodo\_Arbol(me, Nodo.hd)

else

Td := 0;

//cada nodo del arbol equilibrado (balanceado) tiene balance igual a 1, -1 o 0

FactorEquilibrio\_Arbol := Ti - Td;

end;

end;

{-----------------------------------------------------------------------------------}

Procedure ActualizarNiveles\_Arbol(var me: tArbol; Pos:TPos);

var

Nodo,NodoPadre: tNodoIndice;

PosPadre:TPos;

Begin

//Me paro en la posición para obtener la posición de su padre.

seek(me.D, Pos);

Read(me.D, Nodo);

PosPadre:=Nodo.Padre;

If PosPadre<>Lib\_Auxiliar.Posnula Then

Begin

//Me paro en el padre y lo leo para saber su altura, la cual es correcta.

Seek(me.D,PosPadre);

Read(me.D,NodoPadre);

//Al nodo actual le modifico la altura dandole la del padre más uno.

Nodo.nivel:=NodoPadre.nivel+1;

End

ELse

Nodo.nivel:=1;

seek(me.D, Pos);

write(me.D, Nodo);

//El nodo ya tiene su altura correcta y llamo al procedimiento con la posición de sus hijos.

If Nodo.hi<>Lib\_Auxiliar.PosNula Then

ActualizarNiveles\_Arbol(me, Nodo.hi);

If Nodo.hd<>Lib\_Auxiliar.PosNula Then

ActualizarNiveles\_Arbol(me, Nodo.hd);

end;

{-----------------------------------------------------------------------------------}

{

Balancear Arbol:

- Una vez insertado el nodo, pregunto por el balanceo del árbol.

- Se pregunta por el balanceo de cada nodo desde el nodo padre del recién insertado hasta la raíz

- Un nodo esta balanceado si su factor de equilibrio es -1, 1 o 0.

- El ‘factor de equilibrio’ verifica que el nodo este equilibrado. Para ello la función captura el nodo y calcula la profundidad de ambos hijos

El nodo esta equilibrado si la altura del subárbol izquierdo menos la altura del subárbol derecho difiera a lo sumo en 1.

- Si un nodo esta desequilibrado, se invoca a ‘Caso Desquilibrio’ donde verifica que tipo de rotación se llevara a cabo para balancear dicho nodo

}

procedure Balancear\_Arbol(var me: tArbol; Pos:tpos);

var

Nodo: tNodoIndice;

result:Integer;

begin

CapturarNodo\_Arbol(me,Nodo,pos);

pos:= Nodo.Padre;

while (pos<>Lib\_Auxiliar.PosNula) do

begin

Result := FactorEquilibrio\_Arbol(me, Pos);

//si result es negativo, entonces nodo cargado a la izquierda

If Result < 0 then

Result := Result \* (-1);

//El nodo esta balanceado cuando las alturas de sus dos subarboles (izquierdo y derecho) difiera¡en a lo sumo en 1

//Si la diferencia de la altura de los hijo izquierdo y derecho es mayor a 1, el arbol esta desbalanceado. Invoco a caso de desequilibrio

If (Result > 1) then

CasoDesequilibrio\_Arbol(me,pos);

CapturarNodo\_Arbol(me,Nodo,pos);

pos:= Nodo.Padre;

end;

end;

{-----------------------------------------------------------------------------------}

function UltFicticio\_Arbol(var Me: tArbol): tCantidad;

var

Rc: tControlArbol;

begin

Seek(Me.c, 0); Read(Me.c, Rc);

UltFicticio\_Arbol:= Rc.ficticio;

end;

{-----------------------------------------------------------------------------------}

function CantidadNodos\_Arbol(var Me: tArbol): tCantidad;

var

Rc: tControlArbol;

begin

Seek(Me.c, 0); Read(Me.c, Rc);

CantidadNodos\_Arbol:= Rc.Cantidad;

end;

{-----------------------------------------------------------------------------------}

function ObtenerNivel\_Arbol(var Me:tArbol; clave:tClave):tCantidad;

var

pos:tPos;

Nodo: tNodoIndice;

begin

if BuscarNodo\_Arbol(Me, clave, pos) then

begin

seek(me.D,pos); read(me.D,Nodo);

ObtenerNivel\_Arbol:=Nodo.Nivel;

end;

end;

END.

### Árbol Trinario AVL

unit LO\_ArbolTrinario;

{

Con Hijo Medio se enganchan todas las jugadas (ganadoras) en las que intervino el jugador, por ello hijo medio es un registro de control que apunta hacia una lista parcial doble enlace donde se guardan estas jugadas ganadas por el jugador ganador.

Si el jugador ganador no esta en el arbol, se agrega como nodo en el arbol y se insertar en la lista. Si ya se encuentra en el arbol se inserta solo en la lista y se actualiza el registro de control

}

interface

Uses

LO\_ListaParcialDobleEnlace, Lib\_Auxiliar;

TYPE

tNodoIndice = record

clave:tClave;

padre, hi, hd:tPos;

//Control de la lista parcial

HijoMedio: LO\_ListaParcialDobleEnlace.TipoRegControl;

nivel: integer;

end;

tControlArbol = record

Raiz,Borrados:TPos;

Cantidad, NivelMax: TCantidad;

end;

tArchivoNodos=File of tNodoIndice;

tArchivoControlArbol=File Of tControlArbol;

tArbolTrinario = record

D:tArchivoNodos;

C:tArchivoControlArbol;

end;

procedure CrearMe\_ArbolTri (var Me:tArbolTrinario;

\_NombreArchivoControl,\_NombreArchivoIndice:String;

\_Directorio:String);

Procedure AbrirMe\_ArbolTri(var Me:tArbolTrinario);

Procedure CerrarMe\_ArbolTri(var Me:tArbolTrinario);

Procedure DestruirMe\_ArbolTri(var Me:tArbolTrinario);

Function Arbol\_Vacio(var Me: tArbolTrinario):Boolean;

Procedure InsertarNodo\_ArbolTri(var Me:tArbolTrinario; var nodo:tNodoIndice;pos:tPos);

Procedure EliminarNodo\_ArbolTri(var Me:tArbolTrinario; pos:tPos);

Procedure ModificarNodo\_ArbolTri(var Me: tArbolTrinario; RegInd:tNodoIndice;Pos:tPos);

Function BuscarNodo\_ArbolTri (var Me:tArbolTrinario; clave:tClave;var pos:tPos):boolean;

Procedure CapturarNodo\_ArbolTri(var Me: tArbolTrinario; var RegInd:tNodoIndice; Pos:tPos);

function PosNula\_ArbolTri(Var Me:tArbolTrinario):tPos;

function Raiz\_ArbolTri(var Me: tArbolTrinario): tPos;

Function Padre\_ArbolTri(var Me: tArbolTrinario; var Reg: tNodoIndice): tPos;

function HijoDer\_ArbolTri(var Me: tArbolTrinario; pos: tPos): tPos;

function HijoIzq\_ArbolTri(var Me: tArbolTrinario; pos: tPos): tPos;

{---------------------------(BALANCEO)----------------------------------------}

Procedure RotacionSimpleIzquierda(var me: tArbolTrinario; PosNodo: tPos);

Procedure RotacionSimpleDerecha(var me: tArbolTrinario; PosNodo: tPos);

Procedure RotacionCompuestaDerecha(var me: tArbolTrinario; PosNodo: tPos);

Procedure RotacionCompuestaIzquierda(var me: tArbolTrinario; PosNodo: tPos);

Function ProfundidadNodo\_ArbolTri(me: tArbolTrinario; raiz: tPos): integer;

procedure ProfundidadTotal\_ArbolTri(var me:tArbolTrinario; Raiz:tPos; var Profundidad:tProfundidad);

Procedure CasoDesequilibrio\_ArbolTri(me: tArbolTrinario; PosNodo: tPos);

Function FactorEquilibrio\_ArbolTri(me: tArbolTrinario; PosNodo: tPos): integer;

Procedure ActualizarNiveles\_ArbolTri(var me: tArbolTrinario; Pos:TPos);

procedure Balancear\_ArbolTri(var me: tArbolTrinario; Pos:tpos);

{-----------------------------------------------------------------------------}

function CantidadNodos\_Arbol(var Me: tArbolTrinario): tCantidad;

function ObtenerNivel\_ArbolTri(var Me: tArbolTrinario; clave:tClave):tCantidad;

VAR

Me: tArbolTrinario;

implementation

{-----------------------------------------------------------------------------------}

procedure CrearMe\_ArbolTri (var Me:tArbolTrinario;

\_NombreArchivoControl,\_NombreArchivoIndice:String;

\_Directorio:String);

Var

ioC,ioI:integer;

Rc: tControlArbol;

begin

assignFile (Me.C, \_Directorio + \_NombreArchivoControl);

assignFile (Me.D, \_Directorio + \_NombreArchivoIndice);

{$i-}

reset(Me.C); ioC:=IoResult; //Si IoResult es cero, la operacion es exitosa

reset(Me.D); ioI:=IoResult;

if (IoC<>0) or (ioI<>0) then

begin

Rewrite(Me.C);

RC.Raiz:=Lib\_Auxiliar.PosNula;

RC.Borrados:=Lib\_Auxiliar.PosNula;

Rc.Cantidad:= 0;

Write(Me.C,RC);

ReWrite(Me.D);

end;

Close(Me.D);

Close(Me.C);

{$I+}

end;

{-----------------------------------------------------------------------------------}

procedure AbrirMe\_ArbolTri(var Me:tArbolTrinario);

begin

Reset(Me.C);

Reset(Me.D);

end;

{-----------------------------------------------------------------------------------}

procedure CerrarMe\_ArbolTri(var Me:tArbolTrinario);

begin

Close(Me.D);

Close(Me.C);

end;

{-----------------------------------------------------------------------------------}

procedure DestruirMe\_ArbolTri(var Me:tArbolTrinario);

begin

Erase(Me.D);

Erase(Me.C);

end;

{-----------------------------------------------------------------------------------}

Function Arbol\_Vacio(var Me: tArbolTrinario):Boolean;

var

Rc:tControlArbol;

begin

Seek(Me.C,0); Read(Me.c, Rc);

Arbol\_Vacio:= (Rc.Raiz = Lib\_Auxiliar.PosNula) ;

end;

{-----------------------------------------------------------------------------------}

Procedure InsertarNodo\_ArbolTri(var Me:tArbolTrinario; var nodo:tNodoIndice;pos:tPos);//pos, posicion fisica de su padre

var

posnueva:tPos;

reg,rd:tNodoIndice;

RC:tControlArbol;

NodoMedio: boolean;

begin

NodoMedio:= false;

seek(Me.C,0); read(Me.C,RC);

if RC.borrados=Lib\_Auxiliar.PosNula then //No hay borrados

posnueva:=filesize(Me.D)

else //Hay borrados

begin

Posnueva:=RC.Borrados;

seek(Me.D,posnueva); read(Me.D,rD);

RC.Borrados:=rD.hi;

end;

if (RC.Raiz=Lib\_Auxiliar.PosNula) then//Inserto En árbol vacio

begin

nodo.padre:=Lib\_Auxiliar.PosNula;

nodo.hi:=Lib\_Auxiliar.PosNula;

nodo.hd:=Lib\_Auxiliar.PosNula;

nodo.nivel:=1;

RC.Raiz:=posnueva;

end

//Grabamos al final

else //Inserto como hoja

begin

seek(Me.D,pos); read(Me.D,reg); //leo el registro del padre

nodo.padre:= pos;

nodo.hd:= Lib\_Auxiliar.PosNula;

nodo.hi:= Lib\_Auxiliar.PosNula;

nodo.nivel:=Reg.Nivel+1;

//Si el nodo a insertar es la misma que la de su padre, entonces significa que voy a insertar al medio

if nodo.Clave = reg.Clave then

NodoMedio:= true;

if (nodo.clave<=reg.clave) then

reg.hi:=posnueva

else

reg.hd:=posnueva;

seek(Me.d,pos); write(Me.d,reg);

end;

if NodoMedio then //actualizo las poosiciones del nodo que se va a insertar al medio

begin

posNueva:= pos;

nodo.Hi:= reg.Hi;

nodo.Hd:= reg.Hd;

nodo.Padre:= reg.Padre;

nodo.Nivel:= reg.Nivel;

rc.Cantidad:= rc.Cantidad-1;

end

else //Si no inserto al medio, entonces inicializo todas las posiciones

begin

LO\_ListaParcialDobleEnlace.InicializarCabecera(nodo.HijoMedio);

nodo.Hi:=Lib\_Auxiliar.PosNula;

nodo.Hd:=Lib\_Auxiliar.PosNula;

end;

//Si la profuncididad del nodo es mayor a la prfundidad en el control actualizo.

If nodo.nivel > RC.NivelMax then

RC.NivelMax := nodo.nivel;

rc.cantidad:=rc.cantidad+1;

seek(Me.d,posnueva); write(Me.d,nodo); //se escribe el elemento en el archivo

seek(Me.c,0); write(Me.c,rc); // se actualiza el registro control

//pregunto por el balanceo del árbol

Balancear\_ArbolTri(me, PosNueva);

end;

{-----------------------------------------------------------------------------------}

Procedure EliminarNodo\_ArbolTri(var Me:tArbolTrinario; pos:tPos);

var

RD:tNodoIndice;

RC:tControlArbol;

rAux,RHD,RDP,RHI:tNodoIndice;

posAux:tPos;

begin

seek(Me.c,0); read(Me.c,rc);

seek(Me.d,pos); read(Me.d,rd); //Capturo el registro a eliminar

//no tiene padre, no tiene hijos

if (pos=rc.raiz) and (rd.hi=Lib\_Auxiliar.PosNula) and (rd.hd=Lib\_Auxiliar.PosNula) then

Rc.Raiz:=Lib\_Auxiliar.PosNula

else

begin

if (rd.hi=Lib\_Auxiliar.PosNula) and (rd.hd=Lib\_Auxiliar.PosNula) then

//Tiene padre, no tiene hijos. Eliminar de una hoja

begin

Seek(Me.D,rd.padre);read(Me.d,rdp);

If rdp.hi=pos then

rdp.hi:=Lib\_Auxiliar.PosNula

else

rdp.hd:=Lib\_Auxiliar.PosNula;

Seek(Me.d,rd.padre);write(Me.d,RDP);

end

else

begin

if (rd.hi <> Lib\_Auxiliar.PosNula) and (rd.hd <> Lib\_Auxiliar.PosNula) then //Caso general, tiene padre y tiene hijos...

begin

If pos<>RC.raiz then //...no es raiz

begin

Seek(Me.d,rd.padre);read(Me.d,rdp);

If rdp.hi=pos then

rdp.hi:=rd.hd

else

rdp.hd:=rd.hd;

Seek(Me.d,RD.padre);Write(Me.d,rdp);

end

else

RC.raiz:=rd.hd;

Seek(Me.d,rd.hd);read(Me.d,RHD);

RHD.padre:=RD.padre;

posAux:=RD.hd;

rAux:=RHD;

While rAux.hi<>Lib\_Auxiliar.PosNula do

begin

Seek(Me.d,posAux);read(Me.d,RAux);

If raux.hi<>Lib\_Auxiliar.PosNula then

posaux:=rAux.hi;

end;

Seek(Me.d,RD.hi);Read(Me.d,RHI);

RHI.padre:=posAux;

RAux.hi:=RD.hi;

Seek(Me.d,RD.hi);Write(Me.d,RHI);

Seek(Me.d,posAux);Write(Me.d,raux);

If posAux<>RD.hd then

begin

Seek(Me.d,RD.hd);Write(Me.d,RHD);

end;

end

else

begin

If pos<>RC.Raiz then

begin

Seek(Me.D,RD.padre);Read(Me.D,RDP);

If RDP.hi=pos then

begin

If RD.hi=Lib\_Auxiliar.PosNula then

begin

Rdp.hi:=RD.hd;

Seek(Me.D,RD.hd);read(Me.d,RHD);

RHD.padre:=RD.padre;

Seek(Me.d,RD.hd);write(Me.d,RHD);

end

else

begin

Rdp.hi:=RD.hi;

Seek(Me.D,RD.hi);read(Me.d,RHI);

RHI.padre:=RD.padre;

Seek(Me.d,RD.hi);write(Me.d,RHI);

end

end

else

If RD.hi=Lib\_Auxiliar.PosNula then

begin

Rdp.hd:=RD.hd;

Seek(Me.D,RD.hd);read(Me.d,RHD);

RHD.padre:=RD.padre;

Seek(Me.d,RD.hd);write(Me.d,RHD);

end

else

begin

Rdp.hd:=RD.hi;

Seek(Me.D,RD.hi);read(Me.d,RHI);

RHI.padre:=RD.padre;

Seek(Me.d,RD.hi);write(Me.d,RHI);

end;

Seek(Me.d,rd.padre);write(Me.d,rdp);

end

else

If RD.hi=Lib\_Auxiliar.PosNula then

RC.Raiz:=RD.hd

else

RC.raiz:=RD.hi;

end;

end;

end;

RD.hi:=RC.borrados;

RD.hd:=RC.borrados;

RC.borrados:=pos;

RC.Cantidad:=RC.Cantidad-1;

Seek(Me.c,0);write(Me.c,rc); //Grabo la cabecera de control

end;

{-----------------------------------------------------------------------------------}

Procedure ModificarNodo\_ArbolTri(var Me: tArbolTrinario; RegInd:tNodoIndice;Pos:tPos);

begin

seek(Me.D,pos); write(Me.D,RegInd);

end;

{-----------------------------------------------------------------------------------}

Function BuscarNodo\_ArbolTri (var Me:tArbolTrinario; clave:tClave;var pos:tPos):boolean; //Buscar iterativo

var

reg:tNodoIndice;

RC:tControlArbol;

encont:boolean;

posPadre:tpos;

begin

seek(Me.c,0); read(Me.c,rc);

pos:=rc.raiz; encont:=false;

posPadre:=Lib\_Auxiliar.PosNula;

while (not encont) and (pos<>Lib\_Auxiliar.PosNula) do

begin

seek(Me.d,pos); read(Me.d,reg);

if (clave=reg.clave) then

begin

encont:=true;

end

else

begin

if (reg.clave>clave) then

begin

posPadre:=pos;

pos:=reg.hi;

end

else

begin

posPadre:=pos;

pos:=reg.hd;

end;

end;

end;

if (not encont) then

begin

pos:=posPadre;

end;

BuscarNodo\_ArbolTri:=encont;

end;

{-----------------------------------------------------------------------------------}

Procedure CapturarNodo\_ArbolTri(var Me: tArbolTrinario; var RegInd:tNodoIndice; Pos:tPos);

begin

seek(Me.D,Pos); Read(Me.D,RegInd);

end;

{-----------------------------------------------------------------------------------}

function PosNula\_ArbolTri(Var Me:tArbolTrinario):tPos;

begin

PosNula\_ArbolTri:= Lib\_Auxiliar.PosNula;

end;

{-----------------------------------------------------------------------------------}

function Raiz\_ArbolTri(var Me: tArbolTrinario): tPos;

var

Rc: tControlArbol;

begin

Seek(Me.c, 0); Read(Me.c, Rc);

Raiz\_ArbolTri:= Rc.Raiz;

end;

{-----------------------------------------------------------------------------------}

Function Padre\_ArbolTri(var Me: tArbolTrinario; var Reg: tNodoIndice): tPos;

begin

Padre\_ArbolTri:= Reg.padre;

end;

{-----------------------------------------------------------------------------------}

function HijoDer\_ArbolTri(var Me: tArbolTrinario; pos: tPos): tPos;

var

reg: tNodoIndice;

begin

seek(me.D,pos); read(me.D,reg);

HijoDer\_ArbolTri:= reg.hd;

end;

{-----------------------------------------------------------------------------------}

function HijoIzq\_ArbolTri(var Me: tArbolTrinario; pos: tPos): tPos;

var

reg: tNodoIndice;

begin

seek(me.D,pos); read(me.D,reg);

HijoIzq\_ArbolTri:= reg.hi;

end;

{-----------------------------------------------------------------------------------}

//Rotacion en arbol tambien conocido como 'Rotacion Derecha-Derecha'

Procedure RotacionSimpleIzquierda (var me: tArbolTrinario; PosNodo: tPos);

Var

NodoArriba,NodoAbajo,NodoAnterior:tNodoIndice;

RC:tControlArbol;

PosAux:tPos;

begin

Seek (me.D,PosNodo); Read (me.D,NodoArriba);

Seek (me.D,NodoArriba.hd); Read (me.D,NodoAbajo);

Seek (me.C,0);Read(me.C,RC);

//Cambio enlaces

PosAux:=NodoAbajo.hi;

NodoAbajo.padre:=NodoArriba.padre;

NodoAbajo.hi:=PosNodo;

NodoArriba.padre:=NodoArriba.hd;

NodoArriba.hd := PosAux; ;

//Si el desbalanceado no era raíz

If NodoAbajo.padre <> Lib\_Auxiliar.PosNula then

begin

//Me paro en el padre del hijo desbalanceado, y le pongo como hijo al que era su nieto

Seek (Me.D,NodoAbajo.padre); Read (Me.D,NodoAnterior);

If NodoAnterior.hd = PosNodo then

NodoAnterior.hd :=NodoArriba.padre

else

NodoAnterior.hi :=NodoArriba.padre;

seek (Me.D,NodoAbajo.padre); Write (Me.D,NodoAnterior);

end;

//Cambio niveles

NodoAbajo.nivel:=NodoAbajo.nivel-1;

NodoArriba.nivel := NodoArriba.nivel+1;

//Verifico que el del problema no haya sido la raiz

//Anteriormeente a NodoArriba.padre le da NodoAbajo.hdecho. Por lo tanto, pone como nueva raiz

//al hijo derecho del que estaba desbalanceado.

If PosNodo=RC.Raiz then

RC.raiz := NodoArriba.padre;

Seek (Me.D,PosNodo); Write (Me.D,NodoArriba);

Seek (Me.D,NodoArriba.padre); Write (Me.D,NodoAbajo);

Seek (Me.C,0);Write(Me.C,RC);

{Llamo a Actualizar Nivel para actualizar todos los nodos por debajo del padre del nodo que estaba desbalanceado

porque ese nodo sigue teniendo el nivel original}

ActualizarNiveles\_ArbolTri(ME,PosNodo);

end;

{-----------------------------------------------------------------------------------}

//Rotacion en arbol tambien conocido como 'Rotacion Izquierda-Izquierda'

Procedure RotacionSimpleDerecha(var me: tArbolTrinario; PosNodo: tPos);

var

NodoArriba,NodoAbajo,NodoAnterior:tNodoIndice;

RC:tControlArbol;

PosAux:tPos;

begin

Seek (me.D,PosNodo); Read (me.D,NodoArriba);

Seek (me.D,NodoArriba.hi); Read (me.D,NodoAbajo);

Seek (me.C,0);Read(me.C,RC);

{Cambio enlaces}

PosAux:=NodoAbajo.hd;

NodoAbajo.padre:=NodoArriba.padre;

NodoAbajo.hd:=PosNodo;

NodoArriba.padre:=NodoArriba.hi;

NodoArriba.hi := PosAux; ;

If NodoAbajo.padre <> Lib\_Auxiliar.PosNula then

begin

Seek (Me.D,NodoAbajo.padre); Read (Me.D,NodoAnterior);

If NodoAnterior.hd = PosNodo then

NodoAnterior.hd :=NodoArriba.padre

else

NodoAnterior.hi :=NodoArriba.padre;

Seek (Me.D,NodoAbajo.padre); Write (Me.D,NodoAnterior);

end;

{Cambio el campo nivel}

NodoAbajo.nivel:= NodoAbajo.nivel-1;

NodoArriba.nivel := NodoArriba.nivel+1;

{Verifico que el del drama no haya sido la raiz}

If PosNodo=RC.Raiz then

RC.raiz := NodoArriba.padre;

Seek (Me.D,PosNodo); Write (Me.D,NodoArriba);

Seek (Me.D,NodoArriba.padre); Write (Me.D,NodoAbajo);

Seek (Me.C,0);Write(Me.C,RC);

{Llamo a Actualizar Nivel para actualizar todos los nodos por debajo del padre del nodo que estaba desbalanceado

porque ese nodo sigue teniendo el nivel original}

ActualizarNiveles\_ArbolTri(ME,PosNodo);

end;

{-----------------------------------------------------------------------------------}

//Rotacion en arbol tambien conocido como 'Rotacion Izquierda-Derecha'

Procedure RotacionCompuestaDerecha(var me: tArbolTrinario; PosNodo: tPos);

Var

NodoArriba,NodoMedio,NodoAbajo,NodoAnterior:tNodoIndice;

RC:tControlArbol;

PosAux,PosMedio,PosAbajo:tPos;

begin

Seek (Me.D,PosNodo); Read (Me.D,NodoArriba);

Seek (Me.D,NodoArriba.hi); Read (Me.D,NodoMedio);

Seek (Me.D,NodoMedio.hd); Read (Me.D,NodoAbajo);

Seek (Me.C,0);Read(Me.C,RC);

PosMedio:=NodoArriba.hi;

PosAbajo:=NodoMedio.hd;

PosAux:=NodoArriba.padre;

NodoArriba.padre:=NodoMedio.hd ;

NodoArriba.hi:=NodoAbajo.hd;

NodoMedio.padre:=NodoMedio.hd;

NodoMedio.hd:= NodoAbajo.hi;

NodoAbajo.padre:=PosAux;

NodoAbajo.hd:=PosNodo;

NodoAbajo.hi:=PosMedio;

If NodoAbajo.padre <> Lib\_Auxiliar.PosNula then

begin

Seek (Me.D,NodoAbajo.padre); Read (Me.D,NodoAnterior);

If NodoAnterior.hd = PosNodo then

NodoAnterior.hd :=PosAbajo

else

NodoAnterior.hi :=PosAbajo;

Seek(Me.D,NodoAbajo.padre); Write (Me.D,NodoAnterior);

end;

{Cambio el campo nivel}

NodoAbajo.nivel:=NodoAbajo.nivel-2;

NodoArriba.nivel:= NodoArriba.nivel+1;

{Verifico que el del drama no haya sido la raiz}

If PosNodo=RC.Raiz then

RC.raiz := NodoArriba.padre;

Seek (Me.D,PosNodo); write (Me.D,NodoArriba);

Seek (Me.D,PosMedio); write (Me.D,NodoMedio);

Seek (Me.D,PosAbajo); write (Me.D,NodoAbajo);

Seek (Me.C,0); Write(Me.C,RC);

{Llamo a Actualizar Nivel para actualizar todos los nodos por debajo del padre del nodo que estaba desbalanceado

porque ese nodo sigue teniendo el nivel original}

ActualizarNiveles\_ArbolTri(ME,PosNodo);

end;

{-----------------------------------------------------------------------------------}

//Rotacion en arbol tambien conocido como 'Rotacion Derecha-Izquierda'

Procedure RotacionCompuestaIzquierda(var me: tArbolTrinario; PosNodo: tPos);

Var

NodoArriba,NodoMedio,NodoAbajo,NodoAnterior:tNodoIndice;

RC:tControlArbol;

PosAux,PosMedio,PosAbajo:tPos;

begin

Seek (Me.D,PosNodo); Read (Me.D,NodoArriba);

Seek (Me.D,NodoArriba.hd); Read (Me.D,NodoMedio);

Seek (Me.D,NodoMedio.hi); Read (Me.D,NodoAbajo);

Seek (Me.C,0);Read(Me.C,RC);

PosMedio:=NodoArriba.hd;

PosAbajo:=NodoMedio.hi;

PosAux:=NodoArriba.padre;

NodoArriba.padre:=NodoMedio.hi;

NodoArriba.hd:=NodoAbajo.hi;

NodoMedio.padre:=NodoMedio.hi;

NodoMedio.hi:= NodoAbajo.hd;

NodoAbajo.padre:=PosAux;

NodoAbajo.hi:=PosNodo;

NodoAbajo.hd:=PosMedio;

If NodoAbajo.padre <> Lib\_Auxiliar.PosNula then

begin

Seek (Me.D,NodoAbajo.padre); Read (Me.D,NodoAnterior);

If NodoAnterior.hd = PosNodo then

NodoAnterior.hd :=PosAbajo

else

NodoAnterior.hi :=PosAbajo;

Seek (Me.D,NodoAbajo.padre); Write (Me.D,NodoAnterior);

end;

{Cambio el campo nivel}

NodoAbajo.nivel:=NodoAbajo.nivel-2;

NodoArriba.nivel:= NodoArriba.nivel+1;

{Verifico que el del drama no haya sido la raiz}

If PosNodo=RC.Raiz then

RC.raiz := NodoArriba.padre;

Seek (Me.D,PosNodo); write (Me.D,NodoArriba);

Seek (Me.D,PosMedio); write (Me.D,NodoMedio);

Seek (Me.D,PosAbajo); write (Me.D,NodoAbajo);

Seek (Me.C,0);Write (Me.C,RC);

{Llamo a Actualizar Nivel para actualizar todos los nodos por debajo del padre del nodo que estaba desbalanceado

porque ese nodo sigue teniendo el nivel original}

ActualizarNiveles\_ArbolTri(ME,PosNodo);

end;

{-----------------------------------------------------------------------------------}

Function ProfundidadNodo\_ArbolTri(me: tArbolTrinario; raiz: tPos): integer;

var

Profundidad:tProfundidad;

Nodo:tNodoIndice;

Begin

//Dada una posición, devuelve la profundidad (nivel) en el que se encuentra}

Profundidad := 0;

CapturarNodo\_ArbolTri(me, nodo, raiz);

ProfundidadTotal\_ArbolTri(me, raiz, Profundidad);

result:= (Profundidad - nodo.nivel) + 1;

End;

{-----------------------------------------------------------------------------------}

procedure ProfundidadTotal\_ArbolTri(var me:tArbolTrinario; Raiz:tPos;

var Profundidad:tProfundidad);

var

Nodo: tNodoIndice;

begin

If Raiz <> Lib\_Auxiliar.PosNula then

begin

// Primero recursivo tendiendo a la Izquierda

ProfundidadTotal\_ArbolTri(me, HijoIzq\_ArbolTri(me, raiz), Profundidad);

// Recursividad tendiendo a la Derecha.

ProfundidadTotal\_ArbolTri(me, HijoDer\_ArbolTri(me, raiz), Profundidad);

// Guardo en Nodo el nodo indice.

CapturarNodo\_ArbolTri(me, nodo, raiz);

If Nodo.nivel > Profundidad then

Profundidad := Nodo.nivel;

end;

end;

{-----------------------------------------------------------------------------------}

Procedure CasoDesequilibrio\_ArbolTri(me: tArbolTrinario; PosNodo: tPos);

var

Nodo: tNodoIndice;

begin

CapturarNodo\_ArbolTri(me, Nodo, PosNodo);

If FactorEquilibrio\_ArbolTri(me, PosNodo) >= 0 then

//Si el factor de equilibrio es positivo, esta cargado a la izquierda..., reviso con hijo izquierda

begin

If FactorEquilibrio\_ArbolTri(me, Nodo.hi) >= 0 then //Si el factor de equilibrio del hijo izq es positivo, hago una RotacionSimpleDerecha

RotacionSimpleDerecha(me, PosNodo)

else //...si es negativo, hago una RotacionCompuestaDerecha

RotacionCompuestaDerecha(me, PosNodo);

end

else

begin

If FactorEquilibrio\_ArbolTri(me, Nodo.hd) >= 0 then //Si el factor de equilibrio del hijo der es positivo, hago una RotacionCompuestaIzquierda

RotacionCompuestaIzquierda(me, PosNodo)

else //...si es negativo, hago una RotacionSimpleIzquierda

RotacionSimpleIzquierda(me, PosNodo);

end;

end;

{-----------------------------------------------------------------------------------}

Function FactorEquilibrio\_ArbolTri(me: tArbolTrinario; PosNodo: tPos): integer;

var

Nodo: tNodoIndice;

Ti, Td: integer;

begin

If PosNodo = Lib\_Auxiliar.PosNula then

FactorEquilibrio\_ArbolTri := 0

Else

begin

// Guardo en Nodo el nodo indice.

CapturarNodo\_ArbolTri(me, Nodo, PosNodo);

// Calculo la profundidad de ambos hijos.

If Nodo.hi <> Lib\_Auxiliar.PosNula then

Ti := ProfundidadNodo\_ArbolTri(me, Nodo.hi)

Else

Ti := 0;

If Nodo.hd <> Lib\_Auxiliar.PosNula then

Td := ProfundidadNodo\_ArbolTri(me, Nodo.hd)

Else

Td := 0;

{Factor\_Equilibrio = La altura del subarbol derecho menos la altura del subarbol izquierdo para ese nodo

dabe ser <=1 para estar equilibrado}

FactorEquilibrio\_ArbolTri := Ti - Td;

end;

end;

{-----------------------------------------------------------------------------------}

Procedure ActualizarNiveles\_ArbolTri(var me: tArbolTrinario; Pos:TPos);

var

Nodo,NodoPadre: tNodoIndice;

PosPadre:TPos;

Begin

//Me paro en la posición para obtener la posición de su padre.

seek(me.D, Pos); Read(me.D, Nodo);

PosPadre:=Nodo.Padre;

If PosPadre<>Lib\_Auxiliar.Posnula Then

begin

//Me paro en el padre y lo leo para saber su altura, la cual es correcta.

Seek(me.D,PosPadre); Read(me.D,NodoPadre);

//Al nodo actual le modifico la altura dandole la del padre más uno.

//RI.Altura:=RIPadre.Altura+1;

Nodo.nivel:=NodoPadre.nivel+1;

end

eLse

Nodo.nivel:=1;

seek(me.D, Pos); write(me.D, Nodo);

//El nodo ya tiene su altura correcta y llamo al procedimiento con la posición de sus hijos.

If Nodo.hi<>Lib\_Auxiliar.PosNula Then

ActualizarNiveles\_ArbolTri(me, Nodo.hi);

If Nodo.hd<>Lib\_Auxiliar.PosNula Then

ActualizarNiveles\_ArbolTri(me, Nodo.hd);

End;

{-----------------------------------------------------------------------------------}

procedure Balancear\_ArbolTri(var me: tArbolTrinario; Pos:tpos);

var

Nodo: tNodoIndice;

result:Integer;

begin

CapturarNodo\_ArbolTri(me,Nodo,pos);

pos:=Nodo.Padre;

while (pos<>Lib\_Auxiliar.PosNula) do

Begin

Result := FactorEquilibrio\_ArbolTri(me, Pos);

If Result < 0 then

Result := Result \* (-1);

If (Result > 1) then

CasoDesequilibrio\_ArbolTri(me,pos);

CapturarNodo\_ArbolTri(me,Nodo,pos);

pos:=Nodo.Padre;

End;

end;

{-----------------------------------------------------------------------------------}

function CantidadNodos\_Arbol(var Me: tArbolTrinario): tCantidad;

var

Rc: tControlArbol;

begin

Seek(Me.c, 0); Read(Me.c, Rc);

CantidadNodos\_Arbol:= Rc.Cantidad;

end;

{-----------------------------------------------------------------------------------}

function ObtenerNivel\_ArbolTri(var Me: tArbolTrinario; clave:tClave):tCantidad;

var

pos:tPos;

Nodo: tNodoIndice;

begin

if BuscarNodo\_ArbolTri(Me, clave, pos) then

begin

seek(me.D,pos); read(me.D,Nodo);

ObtenerNivel\_ArbolTri:= Nodo.Nivel;

end;

end;

END.

### Colas

unit LO\_Colas;

{

Estructura lineal de tipo FIFO (First In First Out), es decir,

el primero en entrar a la cola, también el primero en salir.

Este TDA utiliza la libreria Libreria Operacional de "Simple Enlace", donde:

- Se inserta siempre al final: Por eso Encolar le parametriza a la lista simple el ultimo elemento

- Se elimina siempre el primer elemento

- Se captura siempre al frente: Por eso Frente captura de la lista el primer elemento

}

interface

uses

LO\_ListaSimpleEnlace, Lib\_Auxiliar;

type

TipoRegDatos = LO\_ListaSimpleEnlace.TipoRegDatos;

TipoRegControl = LO\_ListaSimpleEnlace.TipoRegControl;

tCola = LO\_ListaSimpleEnlace.tLsEnlace;

procedure CrearCola(var Me:tCola;NombreControl:string;NombreDatos:string; Ruta:String);

procedure AbrirCola(var Me:tCola);

procedure CerrarCola(var Me:tCola);

procedure DestruirCola(var ME:tCola);

function ColaVacia(var Me:tCola):boolean;

procedure Frente(var Me:tCola; var reg:tiporegdatos);

procedure Decolar(var me:tCola);

procedure Encolar(var me:tCola; reg:tiporegdatos);

procedure InicializarRegBandera(var Reg:tiporegDatos);

function ClaveNula(var me:tCola): TClave;

function Buscar(var Me:tCola; Clave:TClave):boolean;

implementation

{-----------------------------------------------------------------------------------}

procedure CrearCola(var Me:tCola;NombreControl:string;NombreDatos:string; Ruta:String);

begin

//Creo estructura cola con el metodo de crear de la lista simple

LO\_ListaSimpleEnlace.CrearListaSimple(me,NombreControl,NombreDatos,ruta);

end;

{-----------------------------------------------------------------------------------}

procedure AbrirCola(var Me:tCola);

begin

//Abro la cola invocando al procedimiento abrir de la lista simple

LO\_ListaSimpleEnlace.AbrirListaSimple(me);

end;

{-----------------------------------------------------------------------------------}

procedure CerrarCola(var Me:tCola);

begin

//Cierro la cola invocando al procedimiento cerrar de la lista simple

LO\_ListaSimpleEnlace.CerrarListaSimple(me);

end;

{-----------------------------------------------------------------------------------}

procedure DestruirCola(var ME:tCola);

begin

//Destruyo la cola invocando al procedimiento destruir de la lista simple

LO\_ListaSimpleEnlace.DestruirListaSimple(me);

end;

{-----------------------------------------------------------------------------------}

function ColaVacia(var Me:tCola):boolean;

begin

//Consulto si la cola esta vacia. Devuelve verdadero si la cola esta vacia

ColaVacia:= LO\_ListaSimpleEnlace.ListaSimpleVacia(me);

end;

{-----------------------------------------------------------------------------------}

procedure Frente(var Me:tCola; var reg:tiporegdatos);

begin

//Con Frente capturo siempre el primer elemento de la cola, por eso le paso la posicion del primer elemento de la lista

LO\_ListaSimpleEnlace.CapturarInfo\_LSimple(me,LO\_ListaSimpleEnlace.Primero(me),reg);

end;

{-----------------------------------------------------------------------------------}

procedure Decolar(var me:tCola);

begin

//Al decolar, le paso por parametrizo la posicion nula de la lista (-1) para que elimine el primer elemento

LO\_ListaSimpleEnlace.EliminarInfo\_LSimple(me,LO\_ListaSimpleEnlace.Posnula(me));

end;

{-----------------------------------------------------------------------------------}

procedure Encolar(var me:tCola; reg:tiporegdatos);

begin

//Al encolar se inserta siempre al final, por eso le paso la ultima posicion de la lista

LO\_ListaSimpleEnlace.InsertarInfo\_LSimple(me,reg,LO\_ListaSimpleEnlace.Ultimo(me));

end;

{-----------------------------------------------------------------------------------}

procedure InicializarRegBandera(var Reg:tiporegDatos);

begin

//Creo un registro nulo que servira de 'Bandera' para poder recorrer la cola

Reg.Clave:=Lib\_Auxiliar.ClaveNula;

Reg.Siguiente:=Lib\_Auxiliar.PosNula;

Reg.PosEnDatos:= Lib\_Auxiliar.PosNula;

end;

{-----------------------------------------------------------------------------------}

function ClaveNula(var me:tCola): TClave;

begin

//Devuelve la clave nula de la estructura de lista simple

claveNula:= LO\_ListaSimpleEnlace.ClaveNula(ME);

end;

{-----------------------------------------------------------------------------------}

function Buscar(var Me:tCola; Clave:TClave):boolean;

var

regBandera,reg:tiporegdatos;

corte,enc:boolean;

begin

//Se encola un registro bandera para cortar recorrido cuando en el ciclo se topa con el mismo. Se encontro clave si enc = true

corte:=false;

enc:=false;

//Inicializo registro bandera

InicializarRegBandera(regBandera);

//Encolo el registro bandera

Encolar(me,regBandera);

while not corte do

begin

Frente(me,reg);

Decolar(me);

if clave = reg.Clave then

enc:=true;

if reg.Clave <> Lib\_Auxiliar.ClaveNula then

Encolar(me,reg)

else

corte:=true;

end;

Buscar:= enc;

end;

END.

### Hash

unit LO\_Hash;

{

'Tabla hash' contiene un archivo secuencial y fijo, donde se guardara en cada registro un registro control que apuntara a la lista de Cuentas Corrientes.

'Tabla hash' guarda en cada posicion el registro control de todas las cuentas corrientes de aquellos jugadores que coinciden en la misma posicion de la tabla hash, posicion dada por la funcion Hash

La funcion hash toma los 6 primeros caracteres del nick y los pasa a mayuscula.

Se lleva a cabo la conversion de cada caracter a su ordinal.

Obtiene la suma de los tres primeros ordinales para luego restarlos con los proximos tres ordinales

El resultado de la funcion hash va a ir de un rango de minima posicion y maxima posicion declaradas como constantes

Minima posicion = 0

Maxima posicion = 126

}

interface

uses

Type\_Hash, LO\_ListaParcialDobleEnlace, Lib\_Auxiliar, SysUtils,Math;

type

tHash = file of LO\_ListaParcialDobleEnlace.TipoRegControl; //Primero, Ultimo, Borrado

Procedure Crear (var Me:tHash;Archivo: string; ruta:string);

procedure Abrir (var Me: tHash);

procedure Cerrar (var Me: tHash);

procedure Destruir (var Me: tHash); procedure insertar(var me:tHash; reg:TipoRegControl;pos:tpos);

procedure modificar(var me:tHash; reg:TipoRegControl;pos:tpos);

procedure Capturar(var Me:tHash;pos:tPos; var Reg:TipoRegControl);

function FuncionHash(Clave:tClaveHash): tPos;

implementation

{------------------------------------------------------------------------------}

//Creo estructura Hash

Procedure Crear (var Me:tHash;Archivo: string; ruta:string);

var

RegHash:TipoRegControl;

FaltaDatos:Boolean;

i:tPos;

begin

AssignFile (me,ruta+Archivo);

{$I-}

Reset(me); FaltaDatos:=(ioResult<>0);

If (FaltaDatos) then

begin

Rewrite(me);

//Inicializo registro Hash para insertar en la estructura secuencial y fija

//desde la mínima posicion hasta la máxima

LO\_ListaParcialDobleEnlace.InicializarCabecera(RegHash);

For i:=\_MinPosHash to \_MaxPosHash do

Begin

seek(me,i);

write(me,RegHash);

end;

Close(me);

end;

{$i+}

end;

{------------------------------------------------------------------------------}

function FuncionHash(Clave:tClaveHash): tPos;

var

ValorCaracter, Resultado,i:integer;

ClaveMayuscula: string;

Begin

//nick pasado a mayuscula

ClaveMayuscula:= UpperCase(clave);

Resultado:=0;

//recorrido 6 veces. Desde el primer caracter hasta el sexto

for i:=1 to 6 do

begin

{excepcion: al ordinal de la letra Ñ se altera su valor para que no sea 241 y sea 91. De esta manera esta dentro de los rangos de minima y maxima posicion en tabla hash previamente declaradas como constantes}

if (ord(ClaveMayuscula[i])=241) then

ValorCaracter:=91

else

ValorCaracter:=ord(ClaveMayuscula[i]);

{suma de los tres primeros ordinales para luego restarlos con los proximos tres ordinales}

if (i<=3) then

Resultado:=Resultado + ValorCaracter

else

Resultado:=Resultado - ValorCaracter;

end;

if Resultado < 0 then //Si el resultado es negativo lo paso a positivo

Resultado:= Resultado \* (-1);

FuncionHash:= Resultado;

End;

{------------------------------------------------------------------------------}

procedure Abrir (var Me: tHash);

begin

reset(me);

end;

{------------------------------------------------------------------------------}

procedure Cerrar (var Me: tHash);

begin

close(me);

end;

{------------------------------------------------------------------------------}

procedure Destruir (var Me: tHash);

begin

erase(me);

end;

{------------------------------------------------------------------------------} procedure insertar(var me:tHash; reg:TipoRegControl;pos:tpos);

begin

Seek(me,pos); write(me,reg);

end;

{------------------------------------------------------------------------------}

procedure Capturar(var Me:tHash; pos:tPos; var Reg:TipoRegControl);

begin

Seek(me,pos); read(me,Reg);

end;

{------------------------------------------------------------------------------}

procedure modificar(var me:tHash; reg:TipoRegControl; pos:tpos);

begin

Seek(Me,Pos); Write(Me,Reg);

end;

END.

#### Type Hash

unit Type\_Hash;

Interface

Uses

Lib\_Auxiliar;

const

\_MinPosHash = 0;

\_MaxPosHash = 126; //Segun la funcion hash a utilizar que da maximo 129

type

tClaveHash = String [6];

tPosHash= \_MinPosHash..\_MaxPosHash;

implementation

END.

### Lista Simple Enlace

unit LO\_ListaSimpleEnlace;

interface

USES

Lib\_Auxiliar;

TYPE

TipoRegDatos = record

Clave:Tclave;

Siguiente:TPos;

PosEnDatos: TPos;

end;

TipoRegControl = record

Primero,Ultimo,Borrado :TPos;

end;

TipoArchDatos = file of TipoRegDatos;

TipoArchControl = file of TiporegControl;

tLsEnlace = record

D: tipoArchDatos;

C: tipoArchControl;

end;

procedure CrearListaSimple(var Me:tLsEnlace;NombreControl:String;NombreDatos:String; Ruta:String);

procedure AbrirListaSimple(var Me:tLsEnlace);

procedure CerrarListaSimple(var Me:tLsEnlace);

procedure DestruirListaSimple(var Me:tLsEnlace);

procedure InsertarInfo\_LSimple(var Me:tLsEnlace; reg:TipoRegDatos; Pos:TPos);

procedure EliminarInfo\_LSimple(var Me:tLsEnlace; pos:TPos);

procedure ModificarInfo\_LSimple(var Me:tLsEnlace; pos:TPos; reg:tiporegdatos);

function BuscarInfo\_LdSimple(var Me:tLsEnlace; Clave:Tclave; var Pos :TPos):boolean;

procedure CapturarInfo\_LSimple(var ME:tLsEnlace; pos:TPos; var Reg:TipoRegDatos);

function Primero(var Me:tLsEnlace):TPos;

function Ultimo(var Me:tLsEnlace):TPos;

function Proximo(var Me:tLsEnlace; pos:TPos):TPos;

function ListaSimpleVacia(me:tLsEnlace):boolean;

function Posnula(var Me:tLsEnlace):TPos;

function ClaveNula(var Me:tLsEnlace):tClave;

implementation

{------------------------------------------------------------------------------}

procedure CrearListaSimple(var Me:tLsEnlace;NombreControl:String;NombreDatos:String; Ruta:String);

Var

ioD, ioC:integer;

rc:TipoRegControl;

begin

//Vinculacion de las variable ME con los archivos fisicos

assign(Me.C, Ruta + NombreControl);

Assign(Me.D, Ruta + NombreDatos);

{$i-}

//Si IoResult es cero, la operacion es exitosa

Reset(Me.C); ioC:=IoResult;

Reset(Me.D); ioD:=IoResult;

if ((ioD<>0) or (ioC<>0)) then

begin

Rewrite(Me.D);

Rewrite(Me.C);

//Inicializo variables de control

Rc.Primero:=Lib\_Auxiliar.PosNula;

Rc.Ultimo:=Lib\_Auxiliar.PosNula;

Rc.Borrado:=Lib\_Auxiliar.Posnula;

Write(Me.C,Rc);

Close(Me.D);

Close(Me.C);

end;

{$i+}

End;

{------------------------------------------------------------------------------}

procedure AbrirListaSimple(var Me:tLsEnlace);

begin

reset(me.D);

reset(me.C);

end;

{------------------------------------------------------------------------------}

procedure CerrarListaSimple(var Me:tLsEnlace);

begin

close(me.D);

close(me.C);

end;

{------------------------------------------------------------------------------}

procedure DestruirListaSimple(var Me:tLsEnlace);

begin

erase(me.D);

erase(me.C);

end;

{------------------------------------------------------------------------------}

procedure InsertarInfo\_LSimple(var Me:tLsEnlace; reg:TipoRegDatos; Pos:tpos);

var

rc:tiporegcontrol;

regAnt:tiporegdatos;

posNueva:tpos;

begin

seek(me.C,0); read(me.C,rc); //Leo cabecera de control

if rc.Borrado = Lib\_Auxiliar.PosNula then //No hay borrados

posnueva:= filesize(me.D)

else //hay borrados

begin

seek(me.D,rc.Borrado); read(me.D,regant);

posnueva:= rc.Borrado; //PosNueva sera el primer elemento de la lista de borrados

rc.Borrado:= regant.Siguiente;

end;

if rc.Primero = Lib\_Auxiliar.PosNula then //Lista vacia

begin

rc.Primero:=posnueva;

rc.Ultimo:= posnueva;

reg.Siguiente:= Lib\_Auxiliar.PosNula;

end

else

if pos = Lib\_Auxiliar.PosNula then //Inserto al principio

begin

reg.Siguiente:= rc.Primero;

rc.Primero:= posnueva;

end

else

if pos = rc.Ultimo then //Inserto al final

begin

seek(me.D,pos); read(me.D,regant);

regant.Siguiente:= posnueva;

reg.Siguiente:= Lib\_Auxiliar.PosNula;

rc.Ultimo:= posnueva;

seek(me.D,pos); write(me.D,regant);

end

else //Inserto al medio

begin

seek(me.D,pos); read(me.D,regant);

reg.Siguiente:= regant.Siguiente;

regant.Siguiente:= posnueva;

seek(me.D,pos); write(me.D,regant);

end;

//Grabo registro en la posNueva y cabecera de control actualizado

seek(me.D,posnueva); write(me.D,reg);

seek(me.C,0); write(me.C,rc);

end;

{------------------------------------------------------------------------------}

procedure EliminarInfo\_LSimple(var Me:tLsEnlace; pos:tpos);

var

rc:tiporegcontrol;

RegAnt,regaelim:tiporegdatos;

posaux:tpos;

begin

seek(me.C,0); read(me.C,rc);

if pos = Lib\_Auxiliar.PosNula then //Elimino el primer elemento

begin

seek(me.D,rc.Primero); read(me.D,regaelim);

posaux:= rc.Primero;

rc.Primero:= regaelim.Siguiente;

if rc.Primero = Lib\_Auxiliar.PosNula then //Ahora la lista esta vacia

rc.Ultimo:= Lib\_Auxiliar.PosNula;

end

else

begin

seek(me.D,pos); read(me.D,RegAnt);

posaux:= RegAnt.Siguiente;

seek(me.D,posaux); read(me.D,regaelim);

if rc.Ultimo = RegAnt.Siguiente then

rc.Ultimo := pos;

RegAnt.Siguiente:= regaelim.Siguiente;

regaelim.Siguiente:=Lib\_Auxiliar.PosNula;

seek(me.D,pos); write(me.D,RegAnt);

end;

regaelim.Siguiente:= rc.Borrado;

rc.Borrado:=posaux;

seek(me.D,posaux); write(me.D,regaelim);

seek(me.C,0); write(me.C,rc);

end;

{------------------------------------------------------------------------------}

procedure ModificarInfo\_LSimple(var Me:tLsEnlace; pos:tpos; reg:tiporegdatos);

begin

seek(Me.D,pos); write(Me.D,reg);

end;

{------------------------------------------------------------------------------}

function BuscarInfo\_LdSimple(var Me:tLsEnlace; Clave:Tclave; var Pos :tPos):boolean;

var

posaux:tPos;

corte,encontrado:boolean;

reg:tipoRegDatos;

rc:tiporegControl;

begin

//Busco clave en lista, si no lo encuentra devuelve la posicion donde deberia estar

corte:=false;

encontrado:=false;

seek(me.C,0); read(me.C,rc);

posaux:= rc.Primero;

pos:= Lib\_Auxiliar.PosNula;

while (posaux <> Lib\_Auxiliar.PosNula) and not (Corte) and not (encontrado) do

begin

seek(me.D,posaux); read(me.D,reg);

if reg.Clave = clave then

encontrado:= true

else

if reg.Clave > Clave then

corte:=true

else

begin

pos:= posaux;

posaux:= reg.Siguiente;

end;

end;

BuscarInfo\_LdSimple:= encontrado;

end;

{------------------------------------------------------------------------------}

procedure CapturarInfo\_LSimple(var ME:tLsEnlace; pos:tPos; var Reg:TipoRegDatos);

begin

seek(me.D,pos); read(me.D,reg);

end;

{------------------------------------------------------------------------------}

function Primero(var Me:tLsEnlace):tPos;

var

rc:tiporegcontrol;

begin

seek(me.C,0); read(me.C,rc);

Primero:= rc.Primero; //Posicion del primer elemento en la lista

end;

{------------------------------------------------------------------------------}

function Ultimo(var Me:tLsEnlace):tPos;

var

rc:tiporegcontrol;

begin

seek(me.C,0); read(me.C,rc);

Ultimo:= rc.Ultimo; //Posicion del ultimo elemento en la lista

end;

{------------------------------------------------------------------------------}

function Proximo(var Me:tLsEnlace; pos:tPos):tPos;

var reg:tipoRegDatos;

begin

seek(me.D,pos); read(me.D,reg);

Proximo:= reg.Siguiente; //Posicion del proximo elemento en pos parametrizado

end;

{------------------------------------------------------------------------------}

function ListaSimpleVacia(me:tLsEnlace):boolean;

var

rc:tiporegcontrol;

begin

seek(me.C,0); read(me.C,rc);

ListaSimpleVacia := (rc.Primero = Lib\_Auxiliar.PosNula);

end;

{------------------------------------------------------------------------------}

function Posnula(var Me:tLsEnlace):TPos;

begin

Posnula:= Lib\_Auxiliar.PosNula;

end;

{------------------------------------------------------------------------------}

function ClaveNula(var Me:tLsEnlace):tClave;

begin

ClaveNula:= Lib\_Auxiliar.ClaveNula;

end;

END.

### Lista Doble Enlace

unit LO\_ListaDobleEnlace;

interface

uses

Lib\_Auxiliar;

Type

TipoRegDatos = Record

Ant,Sig:TPos;

Clave: TClave;

PosenDatos: TPos;

end;

TipoRegControl= Record

Primero:TPos;

Ultimo:TPos;

Borrado: TPos;

end;

TipoArchivoDatos = File Of TipoRegDatos;

TipoArchivoControl = File of TipoRegControl;

tListaDoble = Record

C:TipoArchivoControl;

D:TipoArchivoDatos;

end;

Procedure CrearLd (Var Me:tListaDoble; Nombre:String; Ruta:string);

procedure AbrirLd (var Me: tListaDoble);

procedure CerrarLd (var Me: tListaDoble);

procedure DestruirLista (var Me: tListaDoble);

function ListaVacia(me:tListaDoble):boolean;

Procedure Insertar (var Me:tListaDoble; Reg :TipoRegDatos;Pos:TPos);

procedure Eliminar(var Me:tListaDoble; Pos:TPos);

procedure Capturar(var Me:tListaDoble; Pos:TPos; var Reg:TipoRegDatos);

procedure Modificar(var Me:tListaDoble; Pos:TPos; Reg:TipoRegDatos);

function Primero(var Me:tListaDoble): TPos;

function Ultimo(var Me:tListaDoble): TPos;

function PosNula(var Me:tListaDoble): TPos;

function ClaveNula(var Me:tListaDoble): TClave;

Function Anterior(var Me:tListaDoble; Pos:TPos):TPos;

function Proximo(var Me:tListaDoble; Pos:TPos):TPos;

Function Buscar(var Me:tListaDoble;Clave:TClave;Var pos:TPos):Boolean;

Implementation

uses Math,SysUtils,Forms;

Var

Me:tListaDoble;

{--------------------------------------------------------------------------------}

Procedure CrearLd (Var Me:tListaDoble; Nombre:String; Ruta:string);

Var

FaltaControl:Boolean;

FaltaDatos:Boolean;

rc:TipoRegControl;

Begin

//Vinculacion de la variable ME con los archivos fisicos

AssignFile(Me.C,ruta+Nombre+'.CON');

AssignFile(Me.D,ruta+Nombre+'.DAT');

{$i-}

Reset(Me.C);

FaltaControl:=(Ioresult<>0);

Reset(Me.D);

FaltaDatos:=(ioResult<>0);

//Si alguno no existe, los creo nuevamente vacios

If ((faltaControl) or (FaltaDatos)) then

begin

Rewrite(Me.C);

Rc.Primero:=Lib\_Auxiliar.PosNula;

Rc.Ultimo:=Lib\_Auxiliar.PosNula;

Rc.Borrado:=Lib\_Auxiliar.Posnula;

Write(Me.C,Rc);

Rewrite(Me.D);

Close(Me.D);

Close(Me.C);

end;

{$i+}

end;

{--------------------------------------------------------------------------------}

procedure AbrirLD (var Me: tListaDoble);

begin

//Abro los archivos

reset (Me.D);

reset (Me.C);

end;

{--------------------------------------------------------------------------------}

procedure CerrarLD (var Me: tListaDoble);

begin

//Cierro los archivos

Close (Me.D);

Close (Me.C);

end;

{--------------------------------------------------------------------------------}

procedure DestruirLista (var Me: tListaDoble);

begin

//Destruyo los archivos

Erase (Me.D);

Erase (Me.C);

end;

{--------------------------------------------------------------------------------}

function ListaVacia(me:tListaDoble):boolean;

var

rc:tiporegcontrol;

begin

//Funcion que devuleve verdadero si es que que la lista esta vacia y falso si no es asi

seek(me.C,0); read(me.C,rc);

ListaVacia := (rc.Primero = Lib\_Auxiliar.posnula);

end;

{--------------------------------------------------------------------------------}

Procedure Insertar(var Me: tListaDoble; Reg: TipoRegDatos; Pos: TPos);

// Se inserta a lo ultimo

var

rc: TipoRegControl;

Raux, RauxAnt, RegBorr: TipoRegDatos;

PosNueva, posant: TPos;

begin

{ Dada una posición inserta un nuevo registro en la lista }

seek(Me.C, 0); read(Me.C, rc);

if rc.Borrado = Lib\_Auxiliar.PosNula then // No hay registros borrados

PosNueva:= FileSize(Me.D) // Uso el fin de archivo

else

begin

// capturo el primer borrado

seek(Me.D, rc.Borrado); Read(Me.D, RegBorr);

PosNueva:= rc.Borrado;

// el primer elemento de la lista de borrados pasa a ser siguiente

rc.Borrado:= RegBorr.Sig;

end;

If (rc.Primero = Lib\_Auxiliar.PosNula) Then // La lista esta vacia

Begin

rc.Primero:= PosNueva;

rc.Ultimo:= PosNueva;

Reg.Sig:= Lib\_Auxiliar.PosNula;

Reg.Ant:= Lib\_Auxiliar.PosNula;

end

else If rc.Primero = Pos then // Inserto al principio

Begin

Reg.Sig:= rc.Primero;

Reg.Ant:= Lib\_Auxiliar.PosNula;

seek(Me.D, rc.Primero); Read(Me.D, Raux);

Raux.Ant:= PosNueva;

seek(Me.D, rc.Primero); Write(Me.D, Raux);

rc.Primero:= PosNueva;

end

else If (Pos = Lib\_Auxiliar.PosNula) then // Inserto al final

Begin

seek(Me.D, rc.Ultimo); Read(Me.D, Raux);

Raux.Sig:= PosNueva;

Reg.Sig:= Lib\_Auxiliar.PosNula;

Reg.Ant:= rc.Ultimo;

rc.Ultimo:= PosNueva;

seek(Me.D, Reg.Ant); Write(Me.D, Raux);

End

Else // Inserto al medio

Begin

// Leo el registro que se encuentra en Pos

seek(Me.D, Pos); Read(Me.D, Raux);

// El nuevo registro tendra como siguiente al actual en "Pos"

Reg.Sig:= Pos;

// Capturo la posicion del anterior al registro de Raux

posant:= Raux.Ant;

//El nuevo registro (Reg) tendra como anterior al anterior del registro que //"desplaza"

Reg.Ant:= posant;

// Leo el anterior a la posicion parametrizada "Pos" y capturo ese registro

seek(Me.D, posant); Read(Me.D, RauxAnt);

// Actualizo los enlances para que apunten al nuevo registro

RauxAnt.Sig:= PosNueva;

Raux.Ant:= PosNueva;

//Actualizo el anterior con el nuevo enlace siguiente a Posnueva

seek(Me.D, posant); Write(Me.D, RauxAnt);

// Actualizo Raux con el nuevo enlaces anterior a PosNueva

seek(Me.D, Pos); Write(Me.D, Raux);

end;

// Gravo los registros de control y el registro parametrizado con los enlaces //actualizados

seek(Me.D, PosNueva); Write(Me.D, Reg);

seek(Me.C, 0); Write(Me.C, rc);

end;

{--------------------------------------------------------------------------------}

procedure Eliminar(var Me: tListaDoble; Pos: TPos);

var

rc: TipoRegControl;

Raux, RauxAdy, RauxAdy2: TipoRegDatos;

PosNueva, posant: TPos;

begin

seek(Me.C, 0); read(Me.C, rc);

seek(Me.D, Pos); Read(Me.D, Raux);

If (rc.Primero = Pos) and (rc.Ultimo = Pos) Then // Esta vacio1

Begin

rc.Primero:= Lib\_Auxiliar.PosNula;

rc.Ultimo:= Lib\_Auxiliar.PosNula;

end

else If rc.Primero = Pos then // Elimino al principio1

Begin

seek(Me.D, rc.Primero); Read(Me.D, RauxAdy);

RauxAdy.Ant:= Lib\_Auxiliar.PosNula;

seek(Me.D, rc.Primero); Write(Me.D, RauxAdy);

rc.Primero:= Raux.Sig;

end

else

If (Pos = rc.Ultimo) then // Elimino al final1

Begin

seek(Me.D, Raux.Ant); Read(Me.D, RauxAdy);

RauxAdy.Sig:= Lib\_Auxiliar.PosNula;

rc.Ultimo:= Raux.Ant;

seek(Me.D, Raux.Ant); Write(Me.D, RauxAdy);

End

Else // Elimino al medio

Begin

seek(Me.D, Raux.Sig); Read(Me.D, RauxAdy);

seek(Me.D, Raux.Ant); Read(Me.D, RauxAdy2);

RauxAdy2.Sig:= Raux.Sig;

RauxAdy.Ant:= Raux.Ant;

seek(Me.D, Raux.Sig); Write(Me.D, RauxAdy);

seek(Me.D, Raux.Ant); Write(Me.D, RauxAdy2);

end;

Raux.Ant:= -1;

// Grabo el archivo de control

seek(Me.C, 0); Write(Me.C, rc);

end;

{--------------------------------------------------------------------------------}

procedure Capturar (var Me:tListaDoble; Pos:TPos; var Reg:TipoRegDatos);

begin

Seek(Me.D,Pos); Read(Me.D,reg);

end;

{--------------------------------------------------------------------------------}

procedure Modificar(var Me:tListaDoble; Pos:TPos; Reg:TipoRegDatos);

var

Rd:TipoRegDatos;

begin

//Dada una posicion cambio los enlaces hacia el nuevo registro y el viejo lo dejo 'suelto'

Seek(Me.D,Pos); Read(Me.D,Rd);

Reg.Ant:=Rd.Ant;

Reg.Sig:=Rd.Sig;

Seek(Me.D,Pos); Write(Me.D,Reg);

end;

{--------------------------------------------------------------------------------}

function Primero(var Me:tListaDoble): TPos;

var

Rc:TipoRegControl;

begin

//Devuelve la posicion directa del primer elemento de la lista

seek(Me.C,0); read(Me.C,Rc);

Primero:=Rc.Primero;

end;

{--------------------------------------------------------------------------------}

function Ultimo(var Me:tListaDoble): TPos;

var

Rc:TipoRegControl;

begin

//Devuelve la posicion de la ultima posicion

seek(Me.C,0); read(Me.C,Rc);

Ultimo:=Rc.Ultimo;

end;

{--------------------------------------------------------------------------------}

function PosNula(var Me:tListaDoble): TPos;

begin

//Posicion nula de la lista

posNula:=Lib\_Auxiliar.posnula;

end;

{--------------------------------------------------------------------------------}

function ClaveNula(var Me:tListaDoble): TClave;

begin

//Clave nula del TDA

ClaveNula:=Lib\_Auxiliar.ClaveNula;

end;

{--------------------------------------------------------------------------------}

function Proximo(var Me:tListaDoble; Pos:TPos):TPos;

var

Reg: TipoRegDatos;

begin

//Dada una posicion devuelve la posicion siguiente a ella

Seek(Me.D,Pos); Read(Me.D,Reg);

Proximo:=Reg.Sig;

end;

{--------------------------------------------------------------------------------}

Function Anterior(var Me:tListaDoble; Pos:TPos):TPos;

var

Reg: TipoRegDatos;

begin

//Dada una posicion devuelve la posicion anterior a ella

Seek(Me.D,Pos); Read(Me.D,Reg);

Anterior:=Reg.Ant;

end;

{--------------------------------------------------------------------------------}

Function Buscar(var Me:tListaDoble;Clave:TClave;Var pos:TPos):Boolean;

Var

Encontrado:Boolean;

Reg:TipoRegDatos;

rc:TipoRegControl;

begin

//Recorre la lista y dad una clave devulve si la encontro o no. Si no la encuentra devuelve la posicion donde deveria estar

Seek(Me.C,0); Read(Me.C,Rc);

Pos:=Rc.Primero;

Encontrado:=False;

While ((Not Encontrado) And (Pos<>Lib\_Auxiliar.posNula)) Do

Begin

Seek(Me.D,Pos); Read(Me.D,Reg);

IF ((reg.Clave=Clave)) Then

Encontrado:=true

Else

Begin

Pos:=Reg.Sig;

end;

end;

Buscar:=Encontrado;

end;

END.

### Lista Parcial Doble Enlace

unit LO\_ListaParcialDobleEnlace;

{

Esta Libreria Operacional cuenta solo con un archivo de datos,

donde el control (primero, ultimo borrado) de dicho archivo es parametrizado

}

interface

USES

Lib\_Auxiliar, SysUtils;

TYPE

TipoRegControl= Record

Primero:TPos;

Ultimo:TPos;

Borrado: TPos;

end;

TipoRegDatos = Record

Ant,Sig:TPos;

Clave: TClave;

PosEnDatos:tPos;

end;

TipoArchivoDatos = File Of TipoRegDatos;

TipoArchivoControl = File Of TipoRegControl;

//'Lista doble parcial' cuenta solo con un archivo de datos

tLdParcial = Record

D:TipoArchivoDatos;

End;

Procedure CrearLdParcial (var Me:tLdParcial; Nombre:String; Ruta:String);

procedure AbrirLdParcial (var Me: tLdParcial);

procedure CerrarLdParcial (var Me: tLdParcial);

procedure DestruirLdParcial (var Me: tLdParcial);

Procedure InsertarInfo\_LdParcial (var Me:tLdParcial; Reg :TipoRegDatos; Pos:TPos; var RegControl:tipoRegControl);

procedure EliminarInfo\_LdParcial (var Me:tLdParcial; Pos:TPos; var RegControl:TipoRegControl);

procedure ModificarInfo\_LdParcial (var Me:tLdParcial; Pos:TPos; Reg:TipoRegDatos);

Function BuscarInfo\_LdParcial(var Me:tLdParcial;Clave:TClave;Var pos:TPos;var RegControl:TipoRegControl):Boolean;

procedure CapturarInfo\_LdParcial (var Me:tLdParcial; Pos:TPos; var Reg:TipoRegDatos);

function Primero (var Me:tLdParcial; RegControl:TipoRegControl): TPos;

function Ultimo (var Me:tLdParcial; RegControl:TipoRegControl): TPos;

Function Anterior (var Me:tLdParcial; Pos:TPos):TPos;

function Proximo (var Me:tLdParcial; Pos:TPos):TPos;

function LdParcialVacia (var Me:tLdParcial; RegControl:TipoRegControl):boolean;

function PosNula (var Me:tLdParcial): TPos;

function ClaveNula (var Me:tLdParcial): TClave;

procedure InicializarCabecera (var regControl: TipoRegControl);

Implementation

{--------------------------------------------------------------------------------}

Procedure CrearLdParcial (var Me:tLdParcial; Nombre:String; Ruta:String);

Var

FaltaDatos:Boolean;

Begin

AssignFile(Me.D,Ruta+Nombre);

AssignFile(Me.D,Ruta+Nombre);

{$i-}

Reset(Me.D); FaltaDatos:=(ioResult<>0);

if (FaltaDatos) then

begin

Rewrite(Me.D);

Close(Me.D);

end;

{$i+}

end;

{--------------------------------------------------------------------------------}

Procedure AbrirLdParcial (var Me: tLdParcial);

begin

reset (Me.D);

end;

{--------------------------------------------------------------------------------}

Procedure CerrarLDParcial (var Me: tLdParcial);

begin

Close (Me.D);

end;

{--------------------------------------------------------------------------------}

Procedure DestruirLdParcial (var Me: tLdParcial);

begin

Erase (Me.D);

end;

{--------------------------------------------------------------------------------}

Procedure InsertarInfo\_LdParcial (var Me:tLdParcial; Reg :TipoRegDatos; Pos:TPos; var RegControl:tipoRegControl); //Se inserta a lo ultimo

var

RegAux,RegAntAux,RegBorrado:TipoRegDatos;

PosNueva,posAnt:TPos;

begin

if RegControl.Borrado = Lib\_Auxiliar.Posnula then //No hay borrados

PosNueva:=FileSize(Me.D)

else

begin //si hay borrados

seek(Me.D,RegControl.Borrado); Read(Me.D,RegBorrado);

//PosNueva sera el primer elemento de la lista de borrados

PosNueva:= RegControl.Borrado;

RegControl.Borrado:= RegBorrado.Sig;

end;

If (RegControl.Primero=Lib\_Auxiliar.posnula) Then //Lista vacia

begin

RegControl.Primero:=PosNueva;

RegControl.Ultimo:=PosNueva;

Reg.Sig:=Lib\_Auxiliar.PosNula;

Reg.Ant:=Lib\_Auxiliar.PosNula;

end

else

If RegControl.Primero=pos then //Inserto al principio

Begin

Reg.Sig:=RegControl.Primero;

Reg.Ant:=Lib\_Auxiliar.PosNula;

seek(Me.D,RegControl.Primero); read(Me.D,RegAux);

RegAux.Ant:=PosNueva;

seek(Me.D,RegControl.Primero); write(Me.D,RegAux);

RegControl.Primero:=PosNueva;

end

else

If(Pos=Lib\_Auxiliar.posnula) then //Inserto al final

Begin

seek(Me.D,RegControl.Ultimo); read(Me.D,RegAux);

RegAux.Sig:=PosNueva;

Reg.Sig:=Lib\_Auxiliar.PosNula;

Reg.Ant:=RegControl.Ultimo;

RegControl.Ultimo:=PosNueva;

seek(Me.D,Reg.Ant); write(Me.D,RegAux);

end

else //Inserto al medio

begin

Seek(Me.D,Pos); Read(Me.D,RegAux); //Leo reg en Pos

//El nuevo registro tendra como siguiente al actual en que se encuentra en Pos

Reg.Sig:=pos;

PosAnt:=RegAux.ant;

Reg.Ant:=PosAnt;

//Leo el anterior a reg que se encuentra en Pos

Seek(Me.D, PosAnt); Read(Me.D, RegAntAux);

//Actualizo los enlances para que apunten al nuevo registro

RegAntAux.Sig:=PosNueva;

RegAux.Ant:=PosNueva;

//Actualizo el anterior con el nuevo enlace siguiente a Posnueva

Seek(Me.D,PosAnt); Write(Me.D,RegAntAux);

//Actualizo Raux con el nuevo enlaces anterior a PosNueva

Seek(Me.D,Pos); Write(Me.D,RegAux);

end;

//Grabo registro en la posNueva

Seek(Me.D,Posnueva); Write(Me.D,Reg);

end;

{-----------------------------------------------------------------------------------}

Procedure EliminarInfo\_LdParcial (var Me:tLdParcial; Pos:TPos; var RegControl:TipoRegControl);

var

RegAux,RegAuxSig,RegAuxAnt:TipoRegDatos;

PosNueva,posant:TPos;

begin

seek(Me.D,Pos); read(Me.D,RegAux);

If (RegControl.Primero=pos) and (RegControl.Ultimo=Pos) Then //Lista vacia

Begin

RegControl.Primero:=Lib\_Auxiliar.Posnula;

RegControl.Ultimo:=Lib\_Auxiliar.Posnula;

end

else

If RegControl.Primero=pos then //Se elimina al principio

Begin

seek(Me.D,RegAux.sig); read(Me.D,RegAuxSig);

RegAuxSig.Ant:=Lib\_Auxiliar.Posnula;

seek(Me.D,RegAuxSig.sig); write(Me.D,RegAuxSig);

RegControl.Primero:=RegAux.Sig;

end

else

If(Pos=RegControl.Ultimo) then //Se elimina al final

Begin

seek(Me.D,RegAux.Ant); read(Me.D,RegAuxAnt);

RegAuxAnt.Sig:=Lib\_Auxiliar.Posnula;

RegControl.Ultimo:=RegAux.Ant;

seek(Me.D,RegAux.Ant); write(Me.D,RegAuxAnt);

end

else //Se elimina al medio

Begin

seek(Me.D,RegAux.Sig); read(Me.D,RegAuxSig);

seek(Me.D,RegAux.Ant); read(Me.D,RegAuxAnt);

RegAuxAnt.Sig:=RegAux.Sig;

RegAuxSig.Ant:=RegAux.Ant;

seek(Me.D,RegAux.Sig); write(Me.D,RegAuxSig);

seek(Me.D,RegAux.Ant); write(Me.D,RegAuxAnt);

end;

RegAux.Ant:=-1;

//Pila de borrados

RegAux.Ant:= Lib\_Auxiliar.Posnula;

RegAux.Sig:= RegControl.Borrado;

RegControl.Borrado:= Pos;

//Grabo registro borrado

seek(Me.D,pos); write(Me.D,RegAux);

end;

{-----------------------------------------------------------------------------------}

Procedure ModificarInfo\_LdParcial (var Me:tLdParcial; Pos:TPos; Reg:TipoRegDatos);

begin

seek(Me.D,Pos); write(Me.D,Reg);

end;

{-----------------------------------------------------------------------------------}

Function BuscarInfo\_LdParcial (var Me:tLdParcial;Clave:TClave;Var pos:TPos;var RegControl:TipoRegControl):Boolean;

Var

Enc, corte:Boolean;

Reg:TipoRegDatos;

rc:TipoRegControl;

Begin

Pos:=RegControl.Primero;

Enc:=False;

corte:=false;

//Busco clave en lista, si no lo encuentra devuelve la posicion donde deberia estar

While ((Not Enc) And (Pos<>Lib\_Auxiliar.posNula) and not corte) Do

Begin

seek(Me.D,Pos); read(Me.D,Reg);

if (reg.Clave=Clave) Then

Enc:=true

else

begin

if Clave < reg.Clave then

corte:=true

else

Pos:=Reg.Sig;

end;

end;

BuscarInfo\_LdParcial:=Enc;

end;

{-----------------------------------------------------------------------------------}

Procedure CapturarInfo\_LdParcial (var Me:tLdParcial; Pos:TPos; var Reg:TipoRegDatos);

begin

seek(Me.D,Pos); read(Me.D,reg);

end;

{-----------------------------------------------------------------------------------}

Function Primero (var Me:tLdParcial; RegControl:TipoRegControl): TPos;

begin

Primero:=RegControl.Primero;

end;

{-----------------------------------------------------------------------------------}

Function Ultimo (var Me:tLdParcial; RegControl:TipoRegControl): TPos;

begin

Ultimo:=RegControl.Ultimo;

end;

{-----------------------------------------------------------------------------------}

Function Anterior (var Me:tLdParcial; Pos:TPos):TPos;

var

Reg: TipoRegDatos;

begin

seek(Me.D,Pos); read(Me.D,Reg);

Anterior:=Reg.Ant;

end;

{-----------------------------------------------------------------------------------}

Function Proximo (var Me:tLdParcial; Pos:TPos):TPos;

var

Reg: TipoRegDatos;

begin

seek(Me.D,Pos); read(Me.D,Reg);

Proximo:=Reg.Sig;

end;

{-----------------------------------------------------------------------------------}

Function LdParcialVacia (var Me:tLdParcial; RegControl:TipoRegControl):boolean;

begin

LdParcialVacia := (RegControl.Primero = Lib\_Auxiliar.Posnula);

end;

{-----------------------------------------------------------------------------------}

Function PosNula (var Me:tLdParcial): TPos;

begin

posNula:=Lib\_Auxiliar.posnula;

end;

{-----------------------------------------------------------------------------------}

Function ClaveNula (var Me:tLdParcial): TClave;

begin

ClaveNula:=Lib\_Auxiliar.ClaveNula;

end;

{-----------------------------------------------------------------------------------}

Procedure InicializarCabecera (var regControl: TipoRegControl);

begin

regcontrol.Primero:= -1;

regcontrol.Ultimo:= -1;

regcontrol.Borrado:= -1;

end;

END.

## ME

### Jugadores

unit ME\_JUGADORES;

{

Consiga:

Tiene por objeto registrar a todos los jugadores de RuleTrucha

El ME\_JUGADORES contendrá los siguientes archivos:

1a) El archivo JUGADORES.DAT, que se encuentran organizados a través de un índice de Árbol AVL,

que contiene los datos del jugador

2a) El archivo JUGADORES.CON, cabecera de Indice

3a) El archivo JUGADORES.NTX, Indice

}

interface

uses

Lib\_Auxiliar, LO\_ArbolAVL, Type\_JUGADOR,

System.Math,System.SysUtils,Forms,Dialogs;

CONST

\_Directorio = 'Archivos\';

\_NombreArchivoDatos = 'JUGADORES.DAT';

\_NombreArchivoControl = 'JUGADORES.CON';

\_NombreArchivoIndice = 'JUGADORES.NTX';

TYPE

ME\_JUGADOR = Record

D: Type\_JUGADOR.tArchDatos; //Archivo datos

I: LO\_ArbolAVL.tArbol; //Archivo indice de datos

End;

procedure CrearME\_Jugadores (var Me: ME\_JUGADOR);

Procedure AbrirME\_Jugadores(var Me:ME\_JUGADOR);

Procedure CerrarME\_Jugadores(var Me:ME\_JUGADOR);

procedure DestruirME\_Jugadores(var Me:ME\_JUGADOR);

Function MeVacio\_Jugadores(var Me: ME\_JUGADOR):Boolean;

Procedure InsertarInfoME\_Jugadores(var Me: ME\_JUGADOR;Reg:tRegDatos;Pos:tPos);

Procedure EliminarInfoME\_Jugadores(var Me: ME\_JUGADOR; Pos:tPos);

Procedure ModificarInfoME\_Jugadores(var Me: ME\_JUGADOR; Reg:tRegDatos;Pos:tPos);

Procedure CapturarInfoME\_Jugadores(var Me: ME\_JUGADOR; var Reg:tRegDatos; Pos:tPos);

Function BuscarInfoME\_Jugadores(Var Me:ME\_JUGADOR; Var Pos:tPos; Clave:TClave):Boolean;

function PosNula\_Jugadores(Var Me:ME\_JUGADOR):tPos;

function ClaveNula\_Jugadores(Var Me:ME\_JUGADOR):tClave;

function Raiz(var Me: ME\_JUGADOR): tPos;

function ProximoIzq(var Me: ME\_JUGADOR; pos: tPos): tPos;

function ProximoDer(var Me: ME\_JUGADOR; pos: tPos): tPos;

function UltFicticio (var Me: ME\_JUGADOR): tCantidad;

function Cantidad\_Jugadores(var Me: ME\_JUGADOR): tCantidad;

function ObtenerNivel\_enArbol(var Me:ME\_JUGADOR; clave:tClave): tCantidad;

procedure Asignar(var Me:ME\_JUGADOR);

implementation

{-----------------------------------------------------------------------------------}

procedure CrearME\_Jugadores (var Me: ME\_JUGADOR);

Var

ioD:integer;

begin

assignFile (Me.D, \_Directorio + \_NombreArchivoDatos);

{$i-}

reset(Me.D); ioD:=IoResult; //Si IoResult es cero, la operacion es exitosa

if (IoD<>0)then

Rewrite(Me.D); Close(Me.D);

{$I+}

LO\_ArbolAVL.CrearMe\_Arbol(me.I,\_NombreArchivoControl,\_NombreArchivoIndice,\_Directorio);

end;

{-----------------------------------------------------------------------------------}

Procedure AbrirME\_Jugadores(var Me:ME\_JUGADOR);

begin

Asignar(Me);

Reset(Me.D);

LO\_ArbolAVL.AbrirMe\_Arbol(me.I);

end;

{-----------------------------------------------------------------------------------}

Procedure CerrarME\_Jugadores(var Me:ME\_JUGADOR);

begin

Close(Me.D);

LO\_ArbolAVL.CerrarMe\_Arbol(me.i);

end;

{-----------------------------------------------------------------------------------}

procedure DestruirME\_Jugadores(var Me:ME\_JUGADOR);

begin

Erase(Me.D);

LO\_ArbolAVL.DestruirMe\_Arbol(me.i);

end;

{-----------------------------------------------------------------------------------}

Function MeVacio\_Jugadores(var Me: ME\_JUGADOR):Boolean;

begin

MeVacio\_Jugadores:= LO\_ArbolAVL.Arbol\_Vacio(me.i);

end;

{-----------------------------------------------------------------------------------}

Procedure InsertarInfoME\_Jugadores(var Me: ME\_JUGADOR;Reg:tRegDatos;Pos:tPos);

Var

Aux,RegPadre:tRegDatos;

Rc:tControlArbol;

PosenDatos:tPos;

RegME: LO\_ArbolAVL.tNodoIndice;

Begin

PosenDatos:= filesize(Me.D);

reg.Nick:= UpperCase(Reg.Nick);

RegME.PosenDatos:= PosenDatos;

RegME.Clave:= Reg.Nick;

LO\_ArbolAVL.InsertarNodo\_Arbol(Me.I,RegME,Pos, reg.TipoJugador);

Seek (Me.D,RegME.PosEnDatos); Write(Me.D,Reg);

end;

{-----------------------------------------------------------------------------------}

Procedure EliminarInfoME\_Jugadores(var Me: ME\_JUGADOR; Pos:tPos);

begin

//Elimina de la estructura el registro que esta en la posicion parametrizada

LO\_ArbolAVL.EliminarNodo\_Arbol(me.I,pos);

end;

{-----------------------------------------------------------------------------------}

Procedure ModificarInfoME\_Jugadores(var Me: ME\_JUGADOR; Reg:tRegDatos; Pos:tPos);

var

RE: LO\_ArbolAVL.tNodoIndice;

regAlm:tRegDatos;

begin

//Se le pasa la posicion de la estructrura, trae el registro y sobreescribe el //registro en el almacen de datos

Seek(Me.I.D,pos); Read(Me.I.D,RE);

Seek(Me.D,RE.PosEnDatos); Write(Me.D, Reg);

end;

{-----------------------------------------------------------------------------------}

Procedure CapturarInfoME\_Jugadores(var Me: ME\_JUGADOR; var Reg:tRegDatos; Pos:tPos);

var

regIndice: LO\_ArbolAVL.tNodoIndice;

begin

//Se le pasa la posicion en arbol, trae el registro y con posEnDatos trae ese registro

LO\_ArbolAVL.CapturarNodo\_Arbol(ME.I, regIndice, pos);

Seek(Me.D, regIndice.posendatos);Read(Me.D, Reg);

end;

{-----------------------------------------------------------------------------------}

Function BuscarInfoME\_Jugadores(Var Me:ME\_JUGADOR; Var Pos:tPos; Clave:TClave):Boolean;

Begin

//Busco en la arbol, si la clave esta devuelve la posicion donde se encuentra, sino //devuelve la posicion donde deberia estar

BuscarInfoME\_Jugadores:= LO\_ArbolAVL.BuscarNodo\_Arbol(ME.i,clave, pos)

End;

{-----------------------------------------------------------------------------------}

function PosNula\_Jugadores(Var Me:ME\_JUGADOR):tPos;

begin

PosNula\_Jugadores:= LO\_ArbolAVL.PosNula\_Arbol(me.I);

end;

{-----------------------------------------------------------------------------------}

function ClaveNula\_Jugadores(Var Me:ME\_JUGADOR):tClave;

begin

ClaveNula\_Jugadores:= \_clave\_nula\_archivo;

end;

{-----------------------------------------------------------------------------------}

function Raiz(var Me: ME\_JUGADOR): tPos;

begin

Raiz:= LO\_ArbolAVL.Raiz\_Arbol(me.I);

end;

{-----------------------------------------------------------------------------------}

function ProximoIzq(var Me: ME\_JUGADOR; pos: tPos): tPos;

begin

ProximoIzq:= LO\_ArbolAVL.HijoIzq\_Arbol(ME.I,pos);

end;

{-----------------------------------------------------------------------------------}

function ProximoDer(var Me: ME\_JUGADOR; pos: tPos): tPos;

begin

ProximoDer:= LO\_ArbolAVL.HijoDer\_Arbol(ME.I,pos);

end;

{-----------------------------------------------------------------------------------}

function UltFicticio (var Me: ME\_JUGADOR): tCantidad;

begin

UltFicticio:= LO\_ArbolAVL.UltFicticio\_Arbol(me.I);

end;

{-----------------------------------------------------------------------------------}

function Cantidad\_Jugadores(var Me: ME\_JUGADOR): tCantidad;

begin

Cantidad\_Jugadores:= LO\_ArbolAVL.CantidadNodos\_Arbol(me.I);

end;

{-----------------------------------------------------------------------------------}

function ObtenerNivel\_enArbol(var Me:ME\_JUGADOR; clave:tClave): tCantidad;

begin

ObtenerNivel\_enArbol:= LO\_ArbolAVL.ObtenerNivel\_Arbol(Me.i, clave);

end;

{-----------------------------------------------------------------------------------}

procedure Asignar(var Me:ME\_JUGADOR);

begin

AssignFile(me.D, \_Directorio + \_NombreArchivoDatos);

AssignFile(me.I.C, \_Directorio + \_NombreArchivoControl);

AssignFile(me.I.D, \_Directorio + \_NombreArchivoIndice);

end;

END.

#### Type Jugador

unit Type\_JUGADOR;

interface

USES

Sysutils, dateutils;

Const

\_clave\_nula\_archivo='00000000';

TYPE

tClave = string[15];

tString = string [40];

tContrasenia = string[15];

tFecha = TDateTime;

// true = jugador real ... false = jugador ficticio

tTipoJugador = boolean;

// true = jugador conectado ... false = jugador no conectado

tEstadoJugador = boolean;

tBloqueado = boolean; //true = bloqueado ... false = no bloqueado

tRegDatos = Record

Nick: tclave;

Contrasenia: tContrasenia;

Nombre, Apellido: tString;

Alta: tFecha;

UltimaConexion: tFecha;

TipoJugador: tTipoJugador;

Estado: tEstadoJugador;

Bloqueado: tbloqueado;

end;

tArchDatos = File Of tRegDatos;

implementation

END.

### Cuenta Corriente

unit ME\_CTACTE;

{

ME Cuenta Corriente utiliza:

(1) Un archivo de datos donde se almacenan todos los movimientos de cada cuenta corriente

(2) Una lista doble parcial que hará de indice sobre archivo almacen

(3) Una tabla secuencial dinámica que hará de control sobre la lista parcial anterior mencionada

(4) Una lista doble parcial de cuentas corrientes, que almacena todas las cuentas corrientes existentes y que apunta a la tabla secuencial anterior mencionada.

Las cuentas corrientes son de los jugadores cuyos nicks coinciden con el mismo hash

(5) Un archivo fijo (tabla hash) que tiene control sobre la lista cuentas corrientes anterior mencionada. En este archivo se guardara en cada posicion un registro control que apuntara a la lista de Cuentas Corrientes.

Cada posicion del archivo contiene un registro control que corresponde a una lista de cuentas corrientes de aquellos jugadores que coinciden en la misma posicion de la tabla hash, posicion dada por la funcion Hash

}

Interface

uses

SysUtils,

Type\_Hash, Type\_ALMACEN, LO\_Hash, LO\_ListaParcialDobleEnlace, Lib\_Auxiliar;

Const

\_Directorio = 'Archivos\';

\_NombreArch\_TablaHash = 'CUENTA.DAT';

\_NombreArch\_LDCtaCte = 'LISTACUENTAS.CTE';

\_NombreArch\_TablaIndMov = 'CONTROLINDICE.COX';

\_NombreArch\_LDIndMov = 'INDICEALMACEN.NTX';

\_NombreArch\_Almacen = 'ALMACEN.DAT';

type

tipoEstructura = record

tablaHash: LO\_Hash.tHash; //(5)

ListaCtasCtes: LO\_ListaParcialDobleEnlace.tLdParcial; //(4)

end;

tipoDatos = record

tablaControlIndice:LO\_ListaParcialDobleEnlace.TipoArchivoControl; //(3)

ListaIndiceAlmacen:LO\_ListaParcialDobleEnlace.tLdParcial; //(2)

Almacen:Type\_ALMACEN.tipoArchAlmacen; //(1)

end;

ME\_CUENTA = record

Estructura: tipoEstructura;

Datos: tipoDatos;

end;

Procedure CrearMe\_CtaCte (Var Me:ME\_CUENTA);

procedure AbrirMe\_CtaCte (var Me: ME\_CUENTA);

procedure CerrarMe\_CtaCte (var Me: ME\_CUENTA);

procedure DestruirMe\_CtaCte (var Me: ME\_CUENTA);

Procedure InsertarCtaCte (var Me:ME\_CUENTA; Clave:tClaveHash; var PosEnDatos:tPos);

Function SaldoAcumulado (var me:ME\_CUENTA; Clave: tclave): timporte;

Procedure CapturarCtaCte(var Me:ME\_CUENTA; Clave:tClaveHash; var Reg:tipoRegDatos);

procedure CapturarAlmacen (var Me:ME\_CUENTA; Pos:TPos; var Reg:TipoRegAlmacen);

Procedure InsertarIndiceAlmacen(var Me:ME\_CUENTA; Clave:tClaveHash; PosAlmacen:tPos);

procedure CapturarTablaIndiceMov (var Me:ME\_CUENTA; Pos:TPos; var Reg:TipoRegControl);

procedure GrabarTablaIndiceMov (var Me:ME\_CUENTA; Pos:TPos; Reg:TipoRegControl);

Procedure InsertarAlmacen (var Me:ME\_CUENTA; var RegAlmacen:tipoRegAlmacen);

function PosNula (var Me:ME\_CUENTA): TPos;

function ProximoEnLdParcial (var Me:ME\_CUENTA; Pos:TPos):TPos;

Function PrimeroLdParcial(var Me:ME\_CUENTA; Clave:tClaveHash):tPos;

procedure CapturarEnAlmacen (var Me:ME\_CUENTA; Pos:TPos; var Reg:TipoRegAlmacen);

procedure Asignar(var Me:ME\_CUENTA);

implementation

{------------------------------------------------------------------------------}

Procedure CrearMe\_CtaCte (Var Me:ME\_CUENTA);

Var

FaltaAlmacen, FaltaIndice:Boolean;

Begin

LO\_Hash.Crear(Me.Estructura.tablaHash,\_NombreArch\_TablaHash,\_Directorio);

LO\_ListaParcialDobleEnlace.CrearLdParcial

(me.Estructura.ListaCtasCtes,\_NombreArch\_LDCtaCte,\_Directorio);

AssignFile(Me.Datos.tablaControlIndice,\_Directorio +

\_NombreArch\_TablaIndMov);//control indiceAlmacen

{$i-}

Reset(Me.Datos.tablaControlIndice);

FaltaAlmacen:=(Ioresult<>0);

If (FaltaAlmacen) then

begin

Rewrite(Me.Datos.tablaControlIndice);

close(me.Datos.tablaControlIndice);

end;

LO\_ListaParcialDobleEnlace.CrearLdParcial (me.Datos.ListaIndiceAlmacen,\_NombreArch\_LDIndMov,\_Directorio);

AssignFile(Me.Datos.Almacen,\_Directorio + \_NombreArch\_Almacen);//datos almacen

{$i-}

Reset(Me.Datos.Almacen);

FaltaAlmacen:=(Ioresult<>0);

If (FaltaAlmacen) then

begin

Rewrite(Me.Datos.Almacen);

close(me.Datos.Almacen);

end;

end;

{------------------------------------------------------------------------------}

procedure AbrirMe\_CtaCte (var Me: ME\_CUENTA);

begin

Asignar(Me);

LO\_Hash.Abrir(Me.Estructura.tablaHash);

LO\_ListaParcialDobleEnlace.AbrirLDParcial(Me.Estructura.ListaCtasCtes);

reset(me.Datos.tablaControlIndice);

LO\_ListaParcialDobleEnlace.AbrirLDParcial(me.Datos.ListaIndiceAlmacen);

reset(me.Datos.Almacen);

end;

{------------------------------------------------------------------------------}

procedure CerrarMe\_CtaCte (var Me: ME\_CUENTA);

begin

LO\_Hash.Cerrar(Me.Estructura.tablaHash);

LO\_ListaParcialDobleEnlace.CerrarLDParcial(Me.Estructura.ListaCtasCtes);

close(me.Datos.tablaControlIndice);

LO\_ListaParcialDobleEnlace.CerrarLDParcial(me.Datos.ListaIndiceAlmacen);

close(me.Datos.Almacen);

end;

{------------------------------------------------------------------------------}

procedure DestruirMe\_CtaCte (var Me: ME\_CUENTA);

begin

LO\_Hash.Destruir(Me.Estructura.tablaHash);

LO\_ListaParcialDobleEnlace.DestruirLdParcial(Me.Estructura.ListaCtasCtes);

erase(me.Datos.tablaControlIndice);

LO\_ListaParcialDobleEnlace.DestruirLdParcial(me.Datos.ListaIndiceAlmacen);

erase(me.Datos.Almacen);

end;

{------------------------------------------------------------------------------}

Procedure InsertarCtaCte (var Me:ME\_CUENTA; Clave:tClaveHash; var PosEnDatos:tPos);

var

RegHashControl:LO\_ListaParcialDobleEnlace.TipoRegControl;

RegCtaCte:LO\_ListaParcialDobleEnlace.TipoRegDatos;

posEnCtaCte, posClaveHash, pos, posLista, posUltimo:TPos;

buscar:boolean;

RegControlIndice:tipoRegControl;

begin

RegCtaCte.Clave:=Clave;

posClaveHash:=LO\_Hash.FuncionHash(Clave);//Otengo posicion de la tabla Hash

//Obtengo contenido de la tabla Hash con la posicion parametrizada

LO\_Hash.Capturar(me.Estructura.tablaHash,posClaveHash,RegHashControl);

pos:=RegHashControl.primero;

if (pos = Lib\_Auxiliar.PosNula) then //Si no existe el equipo

begin

LO\_ListaParcialDobleEnlace.BuscarInfo\_LdParcial

(me.Estructura.ListaCtasCtes,Clave,PosLista, RegHashControl);

{Antes de insertar RegCtaCte, voy a buscar su posEnTablaControlIndice (vengo de

insertar un movimiento}

posUltimo:=Filesize(me.Datos.tablaControlIndice);

RegCtaCte.PosEnDatos:=posUltimo;

PosEnDatos:=posUltimo;

LO\_ListaParcialDobleEnlace.InsertarInfo\_LdParcial

(me.Estructura.ListaCtasCtes,RegCtaCte,PosLista,RegHashControl);

LO\_Hash.modificar(me.Estructura.tablaHash, RegHashControl,

posClaveHash);//actualizo cabecera tabla hash

LO\_ListaParcialDobleEnlace.InicializarCabecera(regControlIndice);

GrabarTablaIndiceMov(me,posEnDatos,regControlIndice);

end

else //si no existe la CuentaCorriente...

begin

if not (LO\_ListaParcialDobleEnlace.BuscarInfo\_LdParcial

(me.Estructura.ListaCtasCtes,Clave,PosLista, RegHashControl)) then

begin

buscar:=LO\_ListaParcialDobleEnlace.BuscarInfo\_LdParcial

(me.Estructura.ListaCtasCtes,Clave,PosLista, RegHashControl);

//Antes de insertar RegCtaCte ...

posUltimo:=Filesize(me.Datos.tablaControlIndice);

RegCtaCte.PosEnDatos:=posUltimo;

PosEnDatos:=posUltimo;

//...grabo registro en tabla indice de movimientos

LO\_ListaParcialDobleEnlace.InicializarCabecera(regControlIndice);

GrabarTablaIndiceMov(me,posEnDatos,regControlIndice);

LO\_ListaParcialDobleEnlace.InsertarInfo\_LdParcial

(me.Estructura.ListaCtasCtes,RegCtaCte,PosLista,RegHashControl);

LO\_Hash.modificar(me.Estructura.tablaHash, RegHashControl, posClaveHash);

//ACTUALIZO CABECERA TABLA HASH

end

else

begin

LO\_ListaParcialDobleEnlace.CapturarInfo\_LdParcial

(me.Estructura.ListaCtasCtes,PosLista,RegCtaCte);

PosEnDatos:=RegCtaCte.posEnDatos;

end;

end;

end;

{------------------------------------------------------------------------------}

Procedure InsertarAlmacen (var Me:ME\_CUENTA; var RegAlmacen:tipoRegAlmacen);

var

posAlmacen:tPos;

begin

posAlmacen:=fileSize(me.Datos.Almacen);

Seek(me.Datos.Almacen, PosAlmacen);

write(me.Datos.Almacen, RegAlmacen);

InsertarIndiceAlmacen(me, RegAlmacen.Nick, PosAlmacen);

end;

{------------------------------------------------------------------------------}

Procedure InsertarIndiceAlmacen(var Me:ME\_CUENTA; Clave:tClaveHash; PosAlmacen:tPos);

var

posEnDatos:tPos;

PosLista:tPos;

regCI, RegControlIndice:tipoRegControl;

regIndiceAlmacen:tipoRegDatos;

begin

//Invoco InsertarCtaCte por si no existe la cuenta. De todas formar me trae en PosEnDatos que le paso por referencia la posicion que apunta a la 'tabla indice movimientos'

InsertarCtaCte(me, clave, posEnDatos);

//Me paro en la tabla tablaControlIndice, y capturo ese registroControl

CapturarTablaIndiceMov(me,posEnDatos,regControlIndice);

//Preparo registro indice almacen antes de insertar

regIndiceAlmacen.posEnDatos:=PosAlmacen;

regIndiceAlmacen.Clave:=Clave;

//Parametrizo en Buscar el registro control indice que esta en la tabla Indice

//Movimientos en esa posicion

LO\_ListaParcialDobleEnlace.BuscarInfo\_LdParcial

(me.Datos.ListaIndiceAlmacen,Clave,PosLista, RegControlIndice);

LO\_ListaParcialDobleEnlace.InsertarInfo\_LdParcial

(me.Datos.ListaIndiceAlmacen,regIndiceAlmacen,PosLista,RegControlIndice);

//Actualizo cabecera tablba control indice

GrabarTablaIndiceMov(me,posEnDatos,regControlIndice);

end;

{------------------------------------------------------------------------------}

Procedure CapturarCtaCte(var Me:ME\_CUENTA; Clave:tClaveHash; var Reg:tipoRegDatos);

var

RegHashControl:LO\_ListaParcialDobleEnlace.TipoRegControl;

posClaveHash, PosLista:TPos;

begin

posClaveHash:=LO\_Hash.FuncionHash(Clave);//Otengo posicion de la tabla Hash

//capturo el registro control de la tabla hash

LO\_Hash.Capturar(me.Estructura.tablaHash,posClaveHash,RegHashControl);

//Busco en 'lista doble parcial' parametrizando el registro control y capturo

//registro

LO\_ListaParcialDobleEnlace.BuscarInfo\_LdParcial

(me.Estructura.ListaCtasCtes,Clave,PosLista, RegHashControl);

LO\_ListaParcialDobleEnlace.CapturarInfo\_LdParcial

(me.Estructura.ListaCtasCtes,PosLista,Reg);

end;

{------------------------------------------------------------------------------}

procedure CapturarAlmacen (var Me:ME\_CUENTA; Pos:TPos; var Reg:TipoRegAlmacen);

begin

seek(me.Datos.Almacen,Pos);

read(me.Datos.Almacen,Reg);

end;

{------------------------------------------------------------------------------}

//capturar registro en tabla indice de movimientos

procedure CapturarTablaIndiceMov (var Me:ME\_CUENTA; Pos:TPos; var Reg:TipoRegControl);

begin

seek(me.Datos.tablaControlIndice,pos);

read(me.Datos.tablaControlIndice,Reg);

end;

{------------------------------------------------------------------------------}

//Grabo registro en tabla indice de movimientos

procedure GrabarTablaIndiceMov (var Me:ME\_CUENTA; Pos:TPos; Reg:TipoRegControl);

var

RegIndMov:tipoRegControl;

begin

seek(me.Datos.tablaControlIndice,pos);

write(me.Datos.tablaControlIndice,Reg);

end;

{------------------------------------------------------------------------------}

Function SaldoAcumulado (var me:ME\_CUENTA; Clave: tclave): timporte;

var

debeacumulado, haberacumulado: timporte;

regCtaCte,RegIndMovimiento:tipoRegDatos;

pos,PosLista,i:Tpos;

ClaveMayuscula:tClaveHash;

regTablaIndice:tipoRegControl;

RegAlm:TipoRegAlmacen;

begin

ClaveMayuscula:= UpperCase(clave);

CapturarCtaCte(me,ClaveMayuscula,regCtaCte);

CapturarTablaIndiceMov(me,regCtaCte.PosEnDatos,regTablaIndice);

i:=LO\_ListaParcialDobleEnlace.Primero(me.Datos.ListaIndiceAlmacen, regTablaIndice);

debeAcumulado:= 0;

haberAcumulado:= 0;

while (i<>Lib\_Auxiliar.PosNula) do

begin

LO\_ListaParcialDobleEnlace.CapturarInfo\_LdParcial

(me.Datos.ListaIndiceAlmacen,i,RegIndMovimiento);

CapturarAlmacen(me,RegIndMovimiento.PosEnDatos,RegAlm);

debeAcumulado:= debeAcumulado + RegAlm.Debe;

haberAcumulado:= haberAcumulado + RegAlm.haber;

i:=LO\_ListaParcialDobleEnlace.Proximo(me.Datos.ListaIndiceAlmacen,i);

end;

SaldoAcumulado:= debeAcumulado-haberAcumulado;

end;

{------------------------------------------------------------------------------}

Procedure Capturar\_InfoAlmacen(var Me:ME\_CUENTA; Clave:tClaveHash; var RegAlm:TipoRegAlmacen);

var

regCtaCte,RegIndMovimiento:tipoRegDatos;

ClaveMayuscula:tClaveHash;

regTablaIndice:tipoRegControl;

posLista:tPos;

begin

ClaveMayuscula:= UpperCase(clave);

CapturarCtaCte(me,ClaveMayuscula,regCtaCte);

CapturarTablaIndiceMov(me,regCtaCte.PosEnDatos,regTablaIndice);

LO\_ListaParcialDobleEnlace.CapturarInfo\_LdParcial

(me.Datos.ListaIndiceAlmacen,posLista,RegIndMovimiento);

CapturarAlmacen(me,RegIndMovimiento.PosEnDatos,RegAlm);

end;

{------------------------------------------------------------------------------}

Function PrimeroLdParcial(var Me:ME\_CUENTA; Clave:tClaveHash):tPos;

var

regCtaCte,RegIndMovimiento:tipoRegDatos;

pos,PosLista,i:Tpos;

ClaveMayuscula:tClaveHash;

regTablaIndice:tipoRegControl;

RegAlm:TipoRegAlmacen;

begin

ClaveMayuscula:= UpperCase(clave);

CapturarCtaCte(me,ClaveMayuscula,regCtaCte);

CapturarTablaIndiceMov(me,regCtaCte.PosEnDatos,regTablaIndice);

PrimeroLdParcial:=LO\_ListaParcialDobleEnlace.Primero (me.Datos.ListaIndiceAlmacen,

regTablaIndice);

end;

{------------------------------------------------------------------------------}

procedure CapturarEnAlmacen (var Me:ME\_CUENTA; Pos:TPos; var Reg:TipoRegAlmacen);

var

RegIndMovimiento:tipoRegDatos;

begin

LO\_ListaParcialDobleEnlace.CapturarInfo\_LdParcial (me.Datos.ListaIndiceAlmacen,pos,RegIndMovimiento);

seek(me.Datos.Almacen,RegIndMovimiento.PosEnDatos);

read(me.Datos.Almacen,Reg);

end;

{------------------------------------------------------------------------------}

function ProximoEnLdParcial (var Me:ME\_CUENTA; Pos:TPos):TPos;

begin

ProximoEnLdParcial:= LO\_ListaParcialDobleEnlace.Proximo

(me.Datos.ListaIndiceAlmacen,pos);

end;

{------------------------------------------------------------------------------}

function PosNula (var Me:ME\_CUENTA): TPos;

begin

PosNula:= Lib\_Auxiliar.PosNula;

end;

{------------------------------------------------------------------------------}

procedure Asignar(var Me:ME\_CUENTA);

begin

AssignFile(me.Estructura.tablaHash, \_Directorio + \_NombreArch\_TablaHash);

AssignFile(me.Estructura.ListaCtasCtes.D, \_Directorio + \_NombreArch\_LDCtaCte);

AssignFile(Me.Datos.tablaControlIndice, \_Directorio + \_NombreArch\_TablaIndMov);

AssignFile(me.Datos.ListaIndiceAlmacen.D, \_Directorio + \_NombreArch\_LDIndMov);

AssignFile(Me.Datos.Almacen, \_Directorio + \_NombreArch\_Almacen);

end;

END.

#### Type Almacén

unit Type\_ALMACEN;

interface

USES

Sysutils, dateutils;

Const

//movimientos ctacte

\_creacionCtaCte = 'Creacion de cuenta';

\_bajaCtaCte = 'Baja de la cuenta';

\_CompraFicha = 'Compra de fichas';

\_PremioJuego = 'Premio por juego nro. ';

\_RegaloCredito = 'Regalo de credito';

\_Apuesta = 'Apuesta juego Nº ';

type

tClave = string[15];

tFecha = TDateTime;

tConcepto = string[50];

tImporte = 0..maxint;

tipoRegAlmacen = record

Nick: Type\_ALMACEN.tClave;

FechaHora: Type\_ALMACEN.tFecha;

Concepto: Type\_ALMACEN.tConcepto;

Debe, haber, saldo: Type\_ALMACEN.tImporte;

end;

//Archivo donde se almacenan los datos de la cuenta corriente

tipoArchAlmacen = file of TipoRegAlmacen;

//Tipo de conceptos

tipoConceptos = record

const Creacion\_Cuenta = \_creacionCtaCte;

const Baja\_cuenta = \_bajaCtaCte;

const Compra\_Fichas = \_CompraFicha;

const Premio\_Juego = \_PremioJuego;

const Credito\_Regalado = \_RegaloCredito;

const Numero\_Apuesta = \_Apuesta;

end;

implementation

END.

### Juego

unit ME\_JUEGO;

{

Este ME consta con un archivo de datos denominado RULETA.DAT organizado a través de un

TDA Cola, donde se almacena los datos de cada jugada (NroJugada, FechaHora, Estado, Bolilla)

También dispone de un archivo de control denominado RULETA.CON, donde se tiene control sobre el juego (maxima apuesta y minima apuesta)

Este ME al utilizar un TDA de colas, se inserta siempre al final, se elimina el primer elemento, y se captura al frente

Ya sea para UltimaJugada (que cicla hasta encontrar la ultima jugada) o ModificoUltimo(que modifica el ultimo registro con los datos actuales de la jugada parametrizada) se insertar un registroBandera que permite ciclar y funcionar de corte sin cambiar el orden de la cola

}

interface

Uses

LO\_Colas, LO\_ListaSimpleEnlace, Type\_JUEGO, Lib\_Auxiliar, Dialogs, SysUtils;

Const

\_Directorio = 'Archivos\';

\_NombreArchDatos = 'RULETA.DAT';

\_NombreArchControl = 'RULETA.CON';

\_NombreArchColasDatos = 'COLASRULETA.DAT';

\_NombreArchColasControl = 'COLASRULETA.CON';

Type

ME\_RULETA = Record

E: LO\_Colas.tCola;

D: Type\_JUEGO.TipoArchivoDatos;

C: Type\_JUEGO.tipoArchControlJuego; //Control sobre el juego

End;

Procedure CrearME (Var Me:ME\_RULETA);

procedure AbrirME (var Me:ME\_RULETA);

procedure CerrarME (var Me: ME\_RULETA);

procedure DestruirME (var Me: ME\_RULETA);

function MEVacio(var Me: ME\_RULETA): Boolean;

Procedure Insertar (Var Me: ME\_RULETA; Reg: TipoRegDatos);

Procedure Frente(Var Me: ME\_RULETA; var regDat: TipoRegDatos );

Procedure Decolar(Var Me: ME\_RULETA );

function Presente(var ME: ME\_RULETA; reg: TipoRegDatos): Boolean ;

procedure RegistroNulo(var Nulo:TipoRegDatos);

function ClaveNula(var Me: ME\_RULETA):TClave;

function ApuestaMaxima(var Me: ME\_RULETA):tImporte;

function ApuestaMinima(var Me: ME\_RULETA):tImporte;

Procedure UltimaJugada(var Me:ME\_RULETA; var reg:type\_JUEGO.TipoRegDatos);

procedure ModificoUltimo(Me: ME\_RULETA; reg:type\_JUEGO.TipoRegDatos);

procedure Asignar(var Me:ME\_RULETA);

implementation

{------------------------------------------------------------------------------}

Procedure CrearME (Var Me:ME\_RULETA);

Var

FaltaControl:Boolean;

FaltaDatos:Boolean;

rc:TipoRegControlJuego;

Begin

AssignFile(Me.D,\_Directorio+\_NombreArchDatos);

{$i-}

Reset(Me.D);

FaltaDatos:=(Ioresult<>0);

If (FaltaDatos) then

begin

Rewrite(Me.D); Close(Me.D);

end;

AssignFile(Me.C,\_Directorio+\_NombreArchControl);

{$i-}

Reset(Me.C);

FaltaControl:=(Ioresult<>0);

If (FaltaControl) then

begin

Rewrite(Me.C);

//Se incializan las variables de control

Rc.MaximaApuesta:= Type\_JUEGO.MaximaApuesta;

Rc.MinimaApuesta := Type\_JUEGO.MinimaApuesta;

seek(Me.C,0); Write(Me.C,rc);

Close(Me.C);

Close(Me.D);

end;

LO\_Colas.CrearCola(Me.E, \_NombreArchColasControl, \_NombreArchColasDatos, \_Directorio);

End;

{------------------------------------------------------------------------------}

procedure AbrirME (var Me:ME\_RULETA);

var

rc: TipoRegControlJuego;

begin

Asignar(Me);

reset(Me.D);

reset(Me.C);

LO\_Colas.AbrirCola(Me.E);

end;

{------------------------------------------------------------------------------}

procedure CerrarME (var Me:ME\_RULETA);

begin

Close(Me.D);

Close(Me.C);

LO\_Colas.CerrarCola(Me.E);

end;

{------------------------------------------------------------------------------}

procedure DestruirME (var Me:ME\_RULETA);

begin

Erase(Me.D);

Erase(Me.C);

LO\_Colas.DestruirCola(Me.E);

end;

{------------------------------------------------------------------------------}

function Presente(var ME: ME\_RULETA; reg: TipoRegDatos): Boolean ;

begin

//Busca si la jugada pasada en el reg esta en la cola

presente:= LO\_Colas.Buscar(me.E, reg.NroJugada);

end;

{------------------------------------------------------------------------------}

function MEVacio(var Me:ME\_RULETA): Boolean;

begin

MEVacio:= LO\_Colas.ColaVacia(Me.E);

end;

{------------------------------------------------------------------------------}

Procedure Insertar (Var Me: ME\_RULETA; Reg: TipoRegDatos);

var

RegET: LO\_Colas.TipoRegDatos;

RegControl: TipoRegControl;

begin

//Se inserta al final del archivo

RegET.PosEnDatos:= FileSize(Me.D);

RegET.Clave:= (Reg.NroJugada);

LO\_Colas.Encolar(Me.E,RegET);

Seek (Me.D,RegET.PosEnDatos); Write(Me.D,Reg);

end;

{------------------------------------------------------------------------------}

Procedure Frente(Var Me:ME\_RULETA; var regDat: TipoRegDatos );

var

RegAux: LO\_Colas.TipoRegDatos;

Begin

//Tomo el primer valor de la estructura

LO\_Colas.Frente(me.E,RegAux);

seek(me.D,RegAux.PosEnDatos);

read(me.D,regDat);

End;

{------------------------------------------------------------------------------}

Procedure Decolar(Var Me: ME\_RULETA );

Begin

//Elimino el primer valor de la estructura

LO\_Colas.Decolar(me.E);

End;

{------------------------------------------------------------------------------}

//Inicializo el registro nulo que lo utilizo de bandera en la busqueda

procedure RegistroNulo(var Nulo:TipoRegDatos);

begin

nulo.NroJugada:= Lib\_Auxiliar.clavenula;

nulo.Estado:= -1;

nulo.bolilla:= -1;

end;

{------------------------------------------------------------------------------}

function ClaveNula(var Me:ME\_RULETA):TClave;

begin

//Devuelve la clave nula

ClaveNula:=LO\_Colas.ClaveNula(Me.E);

end;

{------------------------------------------------------------------------------}

function ApuestaMaxima(var Me:ME\_RULETA):tImporte;

var

RegControl: tiporegcontrolJuego;

begin

//Devuelve el valor del importe de la apuesta maxima

seek(me.C, 0); Read(me.C, RegControl);

ApuestaMaxima:= regcontrol.MaximaApuesta;

end;

{------------------------------------------------------------------------------}

function ApuestaMinima(var Me:ME\_RULETA):tImporte;

var

RegControl: tiporegcontroljuego;

begin

//Devuelve el valor del importe de la apuesta minima

seek(Me.C,0); Read(Me.C, RegControl);

ApuestaMinima:= regcontrol.MinimaApuesta;

end;

{------------------------------------------------------------------------------}

//Devuelve el reg de la ultima jugada

Procedure UltimaJugada(var Me:ME\_RULETA; var reg:type\_JUEGO.TipoRegDatos);

var

regAux: TipoRegDatos;

regnulo: TipoRegDatos;

begin

ME\_JUEGO.RegistroNulo(regnulo);

ME\_JUEGO.Insertar(me, regnulo);

ME\_JUEGO.Frente(me,regAux);

ME\_JUEGO.Decolar(me);

//Mientras no me encuentre con el RegBandera, que tiene como clave -1,

//voy sobreescribiendo reg que pase por referencia hasta que el ciclo se encuentra de frente con el RegBandera y lo decole

//Es asi que los valores que se guardaran en reg seran del ultimo registro de la cola

while regAux.NroJugada<>ME\_JUEGO.ClaveNula(me) do

begin

reg.NroJugada:= regAux.NroJugada;

reg.FechaHora:= regAux.FechaHora;

reg.Estado:= regAux.Estado;

reg.Bolilla:= regAux.Bolilla;

ME\_JUEGO.Insertar(me, regAux);

ME\_JUEGO.Frente(me,regAux);

ME\_JUEGO.Decolar(me);

end;

end;

{------------------------------------------------------------------------------}

//Modifico el ultimo registro con los datos actuales de la jugada que le parametrizo en reg

procedure ModificoUltimo(Me: ME\_RULETA; reg:type\_JUEGO.TipoRegDatos);

var

regAux, regauxUltimo: TipoRegDatos;

regMod:TipoRegDatos;

regNulo: TipoRegDatos;

begin

if (Presente(me,reg)) then//el registro que le paso esta en la cola

begin

RegistroNulo(regnulo);

Insertar(me, regnulo);

Frente(me,regaux);

Decolar(me);

while regaux.NroJugada<>ClaveNula(me) do//me encontre el registro bandera

begin

if (regaux.NroJugada = reg.NroJugada) then

begin

regaux.FechaHora:= reg.FechaHora;

regaux.Estado:= reg.Estado;

regaux.Bolilla:= reg.Bolilla;

Insertar(me,regaux);

Frente(me,regaux);

Decolar(me);

end

else

begin //no es el registro que busco

Insertar(me,regaux);

Frente(me,regaux);

Decolar(me);

end;

end;

end;

end;

{------------------------------------------------------------------------------}

procedure Asignar(var Me:ME\_RULETA);

begin

AssignFile(Me.D, \_Directorio + \_NombreArchDatos);

AssignFile(Me.C, \_Directorio + \_NombreArchControl);

AssignFile(Me.E.D, \_Directorio + \_NombreArchColasDatos);

AssignFile(Me.E.C, \_Directorio + \_NombreArchColasControl);

end;

END.

#### Type Juego

unit Type\_JUEGO;

interface

Const

MaximaApuesta = 1000; //es el importe máximo de una apuesta. Por defecto 1000 pesos.

MinimaApuesta = 50; //es el importe mínimo de una apuesta. Por defecto 50 pesos.

Type

tImporte = 0..maxint;

tClave = string[6];

tFecha = TDateTime;

tEstados = -1..4; {Juego no creado(-1)/ Juego creado(0)/ Hagan sus apuestas(1)/ no va mas(2)/ bolilla tirada (3)/ premios repartidos y fin de juego(4)};

tBolilla = -1..36; //identifica a la bolilla que ha salido. Se inicializa en -1 (no ha salido la bolilla)

TipoRegControlJuego = record

MaximaApuesta, MinimaApuesta: tImporte;

end;

tipoArchControlJuego = File of TipoRegControlJuego;

TipoRegDatos = Record

NroJugada: TClave;

FechaHora: tFecha;

Estado: tEstados;

Bolilla: tBolilla;

end;

TipoArchivoDatos = File Of TipoRegDatos;

implementation

END.

### Apuestas

unit ME\_APUESTAS;

{El ME APUESTAS guarda todas las apuestas efectuadas en el juego, tanto de jugadores reales como ficticios.

Utiliza un TDA de lista doble}

interface

Uses

Type\_APUESTA, LO\_ListaDobleEnlace, Lib\_Auxiliar, Dialogs;

Const

\_Directorio = 'Archivos\';

\_NombreArchDatos = 'APUESTAS.DAT';

\_NombreArchApuestasLDDatos = 'APUESTASLD.DAT';

\_NombreArchApuestasLDControl = 'APUESTASLD.CON';

Type

ME\_APUESTA = Record

E:LO\_ListaDobleEnlace.tListaDoble;

D:Type\_APUESTA.TipoArchDatos;

End;

Procedure CrearME (Var Me:ME\_APUESTA);

procedure AbrirME\_Apuestas (var Me: ME\_APUESTA);

procedure CerrarME\_Apuestas (var Me: ME\_APUESTA);

procedure DestruirME\_Apuestas (var Me: ME\_APUESTA);

function MEVacio\_Apuestas(me:ME\_APUESTA):boolean;

Procedure InsertarInfoME\_Apuestas (var Me:ME\_APUESTA; Reg:Type\_APUESTA.TipoRegDatos;Pos:tPos);

procedure EliminarInfoME\_Apuestas (var Me:ME\_APUESTA; Pos:tPos);

procedure CapturarInfoME\_Apuestas (var Me:ME\_APUESTA; Pos:tPos; var Reg:Type\_APUESTA.TipoRegDatos);

procedure ModificarInfoME\_Apuestas (var Me:ME\_APUESTA; Pos:tPos; Reg:Type\_APUESTA.TipoRegDatos);

function PosNula\_Apuestas (var Me:ME\_APUESTA): tPos;

function ClaveNula\_Apuestas(var Me:ME\_APUESTA): tClave;

Function BuscarInfoME\_Apuestas(var Me:ME\_APUESTA;Clave:tClave;Var pos:tPos):Boolean;

function Primero (var Me:ME\_APUESTA): tPos;

function Ultimo (var Me:ME\_APUESTA): tPos;

Function Anterior(var Me:ME\_APUESTA; Pos:tPos):tPos;

function Proximo (var Me:ME\_APUESTA; Pos:tPos):tPos;

function SaldoApuestasPartida (var Me:ME\_APUESTA; Clave: tclave; Jugada: tClave): tImporte;

procedure Asignar(var Me:ME\_APUESTA);

Implementation

Procedure CrearME (Var Me:ME\_APUESTA);

Var

FaltaControl:Boolean;

FaltaDatos:Boolean;

rc:TipoRegControl;

Begin

AssignFile(Me.D,\_Directorio+'APUESTAS.DAT');

{$i-}

Reset(Me.D);

FaltaDatos:=(ioResult<>0);

If (FaltaDatos) then

begin

Rewrite(Me.D); Close(Me.D);

end;

LO\_ListaDobleEnlace.CrearLd(Me.E, 'APUESTASLD',\_Directorio);

End;

{--------------------------------------------------------------------------------}

procedure AbrirME\_Apuestas (var Me: ME\_APUESTA);

begin

Asignar(Me);

reset (Me.D);

LO\_ListaDobleEnlace.AbrirLd(me.E);

end;

{--------------------------------------------------------------------------------}

procedure CerrarME\_Apuestas (var Me: ME\_APUESTA);

begin

Close (Me.D);

LO\_ListaDobleEnlace.CerrarLd(me.E)

end;

{--------------------------------------------------------------------------------}

procedure DestruirME\_Apuestas (var Me: ME\_APUESTA);

begin

Erase (Me.D);

LO\_ListaDobleEnlace.DestruirLista(me.E);

end;

{--------------------------------------------------------------------------------}

function MEVacio\_Apuestas(me:ME\_APUESTA):boolean;

begin

MEVacio\_Apuestas:= LO\_ListaDobleEnlace.ListaVacia(me.E);

end;

{--------------------------------------------------------------------------------}

Procedure InsertarInfoME\_Apuestas (var Me:ME\_APUESTA; Reg:Type\_APUESTA.TipoRegDatos;Pos:tPos);

var

RegEst: LO\_ListaDobleEnlace.TipoRegDatos;

Raux,RauxAnt,RauxSig:TipoRegDatos;

PosenDatos:TPos;

begin

//obtengo final del archivo almacen donde se guardarà la apuesta

PosenDatos:= filesize(Me.D);

//se completa registro para insertar en la lista doble con pos apuntando al registro

//en el almacen

RegEst.PosenDatos:= PosenDatos;

RegEst.Clave:= Reg.NroJugada;

//inserto en lista doble

LO\_ListaDobleEnlace.Insertar(Me.E,RegEst,Pos);

//grabo en almacen

Seek (Me.D,RegEst.PosEnDatos); Write(Me.D,Reg);

end;

{--------------------------------------------------------------------------------}

procedure EliminarInfoME\_Apuestas (var Me:ME\_APUESTA; Pos:tPos);

begin

LO\_ListaDobleEnlace.Eliminar(me.E,pos);

end;

{--------------------------------------------------------------------------------}

procedure CapturarInfoME\_Apuestas(var Me:ME\_APUESTA; Pos:tPos; var Reg:Type\_APUESTA.TipoRegDatos);

var

RE: LO\_ListaDobleEnlace.TipoRegDatos;

begin

//Accediendo a la lista en la posicion parametrizada, obtengo la posicion en datos

//(posicion que corresponde al archivo almacen)

LO\_ListaDobleEnlace.Capturar(ME.E,Pos,RE);

Seek(Me.D,RE.PosEnDatos); Read(Me.D,Reg);

end;

{--------------------------------------------------------------------------------}

procedure ModificarInfoME\_Apuestas(var Me:ME\_APUESTA; Pos:tPos; Reg:Type\_APUESTA.TipoRegDatos);

var

RE:LO\_ListaDobleEnlace.TipoRegDatos;

begin

{Cuando se quiere modificar se pide la posicion de la estructura, se lee el registro de la estructura que trae la posicion en datos del alamcen con la informacion y ese registro sera modificado}

Seek(Me.E.D,pos);

Read(Me.E.D,RE);

Seek(Me.D,RE.PosEnDatos);

Write(Me.D,Reg);

end;

{--------------------------------------------------------------------------------}

function Primero(var Me:ME\_APUESTA):TPos;

var

Rc:TipoRegControl;

begin

//Obtengo la posicion del primero que se encuentra en la estructura

Primero:=LO\_ListaDobleEnlace.primero(me.e)

end;

{--------------------------------------------------------------------------------}

function Ultimo (var Me:ME\_APUESTA):TPos;

var

Rc:TipoRegControl;

begin

//Obtengo la posicion del ultimo que se encuentra en la estructura

Ultimo:=LO\_ListaDobleEnlace.Ultimo(me.e);

end;

{--------------------------------------------------------------------------------}

function PosNula\_Apuestas (var Me:ME\_APUESTA): tPos;

begin

//Obtengo la posicion nula de la estructura

PosNula\_Apuestas:=LO\_ListaDobleEnlace.PosNula(me.e)

end;

{--------------------------------------------------------------------------------}

function ClaveNula\_Apuestas(var Me:ME\_APUESTA): tClave;

begin //Obtengo la clave nula de la estructura

ClaveNula\_Apuestas:=LO\_ListaDobleEnlace.ClaveNula(me.e);

end;

{--------------------------------------------------------------------------------}

function Proximo(var Me:ME\_APUESTA;Pos:TPos):TPos;

begin

//Dada una posicion, obtengo la posicion del proximo que se encuentra en la estructura

Proximo:=LO\_ListaDobleEnlace.Proximo(me.e, Pos);

end;

{--------------------------------------------------------------------------------}

Function Anterior(var Me:ME\_APUESTA;Pos:TPos):TPos;

begin

//Dada una posicion, obtengo la posicion del anterior que se encuentra en la estructura

Anterior:=LO\_ListaDobleEnlace.Anterior(me.e, Pos);

end;

{--------------------------------------------------------------------------------}

Function BuscarInfoME\_Apuestas(var Me:ME\_APUESTA;Clave:tClave;Var pos:tPos):Boolean;

{Con la clave parametrizada se busca en la lista doble, si se encuentra devuelve la posicion sino la encuentra devuelve la posicion donde deberia estar esa clave}

Begin

BuscarInfoME\_Apuestas:= LO\_ListaDobleEnlace.Buscar(me.E,clave,pos);

end;

{--------------------------------------------------------------------------------}

//Dada una clave y un numero de jugada, obtengo el saldo total apostado por ese jugador //en esa jugada

function SaldoApuestasPartida (var Me:ME\_APUESTA; Clave: tclave; Jugada: tClave): tImporte;

var

i: tpos;

Reg: Type\_APUESTA.TipoRegDatos;

Importe: tImporte;

begin

Importe:= 0;

i:= Primero(me);

while i<>PosNula\_Apuestas(me) do

begin

CapturarInfoME\_Apuestas(me,i,Reg);

if (reg.NroJugada= Jugada) and (reg.Nick=Clave) then

begin

Importe:=Importe+reg.Importe;

end;

i:= Proximo(me, i);

end;

SaldoApuestasPartida:= Importe;

end;

{--------------------------------------------------------------------------------}

procedure Asignar(var Me:ME\_APUESTA);

begin

AssignFile(Me.D,\_Directorio+\_NombreArchDatos);

AssignFile(Me.E.D,\_Directorio+ \_NombreArchApuestasLDDatos);

AssignFile(Me.E.C,\_Directorio+ \_NombreArchApuestasLDControl);

end;

END.

#### Type Apuesta

unit Type\_APUESTA;

interface

USES

Sysutils;

CONST

//Nomencladores

Nom\_Rojonegro = 'RN';

Nom\_ParImpar= 'PI';

Nom\_PasaFalta= 'PF';

Nom\_Docena= 'DO';

Nom\_Columna= 'CO';

Nom\_Pleno= 'XX';

TYPE

tClaveApuesta = string[15];

tNomenclador = string[2];

tValor = string[2];

tImporte = 0..maxint;

TipoRegDatos = Record

NroJugada:tClaveApuesta;

Nick:tClaveApuesta;

Nomenclador: tNomenclador;

Valor: tValor;

Importe: tImporte;

end;

TipoArchDatos = File Of TipoRegDatos;

implementation

END.

### Ganadores

unit ME\_GANADORES;

{

En este método se recolectan los ganadores de cada juego y las jugadas ganadas de cada jugador

Cada nuevo ganador se inserta en el arbol

Con Hijo Medio se enganchan todas las jugadas (ganadoras) en las que intervino el jugador, por ello hijo medio es un registro de control que apunta hacia una lista parcial doble enlace donde se guardan estas jugadas ganadas por el jugador ganador.

Si el jugador ganador no esta en el arbol, se agrega como nodo en el arbol y se insertar en la lista

Si ya se encuentra en el arbol se inserta solo en la lista y se actualiza el registro de control

}

interface

Uses

Lib\_Auxiliar, LO\_ArbolTrinario, LO\_ListaParcialDobleEnlace, Type\_GANADOR;

CONST

\_Directorio = 'Archivos\';

\_NombreArchivoDatos = 'GANADORES.DAT';

\_NombreArchControl\_ArbolTrinario = 'GANADORES.CON';

\_NombreArchIndice\_ArbolTrinario = 'GANADORES.NTX';

\_NombreListaIndiceDatos = 'INDICEGANADORES.IND';

TYPE

regNodo = LO\_ArbolTrinario.tNodoIndice;

ME\_GANADOR = Record

Almacen: Type\_GANADOR.TipoArchivoDatos; //datos

//lista indice sobre archivo almacen de datos

ListaIndiceAlmacen: LO\_ListaParcialDobleEnlace.tLdParcial;

ArbolTri: LO\_ArbolTrinario.tArbolTrinario; //estructura arbol

end;

procedure CrearME\_Ganadores (var Me: ME\_GANADOR);

Procedure AbrirME\_Ganadores(var Me:ME\_GANADOR);

Procedure CerrarME\_Ganadores(var Me:ME\_GANADOR);

Procedure DestruirME\_Ganadores(var Me:ME\_GANADOR);

Function MeVacio\_Ganadores(var Me: ME\_GANADOR):Boolean;

Procedure InsertarInfoME\_Ganadores(var Me: ME\_GANADOR; RegAlm:Type\_GANADOR.TipoRegDatos; Clave:tClave);

Procedure CapturarInfoME\_Ganadores(var Me: ME\_GANADOR; var nodo:LO\_ArbolTrinario.tNodoIndice; Pos:TPos);

Function BuscarInfoME\_Ganadores(Var Me:ME\_GANADOR; Var Pos:TPos; Clave:TClave):Boolean;

function PosNula\_Ganadores(Var Me:ME\_GANADOR):TPos;

function ProximoIzq (var Me:ME\_GANADOR; pos: TPos): TPos;

function ProximoDer (var Me:ME\_GANADOR; pos: TPos): TPos;

function Raiz(var Me: ME\_GANADOR): TPos;

function PrimerJugada\_Ganador(var Me:ME\_GANADOR; nodo:LO\_ArbolTrinario.tNodoIndice):TPos;

function SiguienteJugada\_Ganador(var Me:ME\_GANADOR; pos:TPos): TPos;

procedure CapturarJugada\_Ganador(var Me:ME\_GANADOR; pos:TPos; var reg:Type\_GANADOR.TipoRegDatos);

function Cantidad\_Ganadores(var Me: ME\_GANADOR): tCantidad;

function ObtenerNivel\_enArbolTri(var Me:ME\_GANADOR; clave:tClave): tCantidad;

procedure Asignar(var Me:ME\_GANADOR);

IMPLEMENTATION

{-----------------------------------------------------------------------------------}

procedure CrearME\_Ganadores (var Me: ME\_GANADOR);

var

ioD:integer;

Begin

assignFile (Me.Almacen, \_Directorio + \_NombreArchivoDatos);

{$i-}

reset(Me.Almacen); ioD:=IoResult; //Si IoResult es cero, la operacion es exitosa

if (IoD<>0)then

Rewrite(Me.Almacen);

Close(Me.Almacen);

{$I+}

LO\_ListaParcialDobleEnlace.CrearLdParcial(Me.ListaIndiceAlmacen,\_NombreListaIndiceDatos,\_Directorio);

LO\_ArbolTrinario.CrearMe\_ArbolTri(Me.ArbolTri,\_NombreArchControl\_ArbolTrinario,\_NombreArchIndice\_ArbolTrinario,\_Directorio);

end;

{-----------------------------------------------------------------------------------}

Procedure AbrirME\_Ganadores(var Me:ME\_GANADOR);

Begin

Asignar(Me);

reset(Me.Almacen);

LO\_ListaParcialDobleEnlace.AbrirLdParcial(Me.ListaIndiceAlmacen);

LO\_ArbolTrinario.AbrirMe\_ArbolTri(Me.ArbolTri);

end;

{-----------------------------------------------------------------------------------}

Procedure CerrarME\_Ganadores(var Me:ME\_GANADOR);

Begin

close(Me.Almacen);

LO\_ListaParcialDobleEnlace.CerrarLdParcial(Me.ListaIndiceAlmacen);

LO\_ArbolTrinario.CerrarMe\_ArbolTri(Me.ArbolTri);

end;

{-----------------------------------------------------------------------------------}

Procedure DestruirME\_Ganadores(var Me:ME\_GANADOR);

Begin

erase(Me.Almacen);

LO\_ListaParcialDobleEnlace.DestruirLdParcial(Me.ListaIndiceAlmacen);

LO\_ArbolTrinario.DestruirMe\_ArbolTri(Me.ArbolTri);

end;

{-----------------------------------------------------------------------------------}

Function MeVacio\_Ganadores(var Me: ME\_GANADOR):Boolean;

begin

MeVacio\_Ganadores:= LO\_ArbolTrinario.Arbol\_Vacio(me.arbolTri);

end;

{-----------------------------------------------------------------------------------}

Procedure InsertarInfoME\_Ganadores(var Me: ME\_GANADOR; RegAlm:Type\_GANADOR.TipoRegDatos; Clave:tClave);

var

PosNodo, PosLista, PosFinalAlm: TPos;

nodoAux: LO\_ArbolTrinario.tNodoIndice;

regIndAlm: LO\_ListaParcialDobleEnlace.TipoRegDatos;

RegDatos:Type\_GANADOR.TipoRegDatos;

nodo:LO\_ArbolTrinario.tNodoIndice;

Begin

//Busco en arbol trinario

if (LO\_ArbolTrinario.BuscarNodo\_ArbolTri(Me.ArbolTri,clave,PosNodo)) then //Si existe //el nodo entonces...

begin

//Caputuro nodo en arbol

LO\_ArbolTrinario.CapturarNodo\_ArbolTri(Me.ArbolTri,nodoAux,PosNodo);

//A PosFinalAlm le doy el final del archivo de almacen de datos

PosFinalAlm:=FileSize(Me.Almacen);

regIndAlm.PosEnDatos:= PosFinalAlm;

regIndAlm.Clave:=clave;

//Busco la posicion en la lista indice datos e inserto en la lista Indice datos //pasandole por parametro el regControl que es el campo HijoMedio

LO\_ListaParcialDobleEnlace.BuscarInfo\_LdParcial(Me.ListaIndiceAlmacen,Clave, PosLista, nodoAux.HijoMedio);

LO\_ListaParcialDobleEnlace.InsertarInfo\_LdParcial(Me.ListaIndiceAlmacen, regIndAlm, PosLista, nodoAux.HijoMedio);

//Inserto al final del archivo almacen.

Seek (me.Almacen,PosFinalAlm);

Write(me.Almacen,RegAlm);

//Busco la posicion nuevamente del arbol y lo modifico con campo HijoMedio //actualizado

LO\_ArbolTrinario.BuscarNodo\_ArbolTri(Me.ArbolTri,nodoAux.clave,PosNodo);

LO\_ArbolTrinario.ModificarNodo\_ArbolTri(Me.ArbolTri,nodoAux,PosNodo);

end

else //No existe el nodo, entonces...

begin

nodo.clave:=Clave;

//Inicializo la cabecera del hijo del medio

LO\_ListaParcialDobleEnlace.InicializarCabecera(nodo.HijoMedio);

//No existe en el árbol el nodo lo inserto

LO\_ArbolTrinario.InsertarNodo\_ArbolTri(Me.ArbolTri,nodo,PosNodo);

//A PosFinalAlm le doy el final del archivo de almacen de datos

PosFinalAlm:=FileSize(Me.Almacen);

regIndAlm.PosEnDatos:= PosFinalAlm;

regIndAlm.Clave:=clave;

//Busco la posicion en la lista indice datos e inserto en la lista Indice //datos pasandole por parametro el regControl que es el campo HijoMedio

LO\_ListaParcialDobleEnlace.BuscarInfo\_LdParcial

(Me.ListaIndiceAlmacen,Clave, PosLista, nodo.HijoMedio);

LO\_ListaParcialDobleEnlace.InsertarInfo\_LdParcial (Me.ListaIndiceAlmacen,

regIndAlm, PosLista, nodo.HijoMedio);

//Inserto al final del archivo almacen.

Seek (me.Almacen,PosFinalAlm);

Write(me.Almacen,RegAlm);

//Busco la posicion nuevamente del arbol y lo modifico con campo HijoMedio //actualizado

LO\_ArbolTrinario.BuscarNodo\_ArbolTri(Me.ArbolTri,nodo.clave,PosNodo);

LO\_ArbolTrinario.ModificarNodo\_ArbolTri(Me.ArbolTri,nodo,PosNodo);

end;

end;

{-----------------------------------------------------------------------------------}

Procedure CapturarInfoME\_Ganadores(var Me: ME\_GANADOR; var nodo:LO\_ArbolTrinario.tNodoIndice; Pos:TPos);

begin

LO\_ArbolTrinario.CapturarNodo\_ArbolTri(me.ArbolTri, nodo, pos);

end;

{-----------------------------------------------------------------------------------}

Function BuscarInfoME\_Ganadores(Var Me:ME\_GANADOR; Var Pos:TPos; Clave:TClave):Boolean;

begin

BuscarInfoME\_Ganadores:=LO\_ArbolTrinario.BuscarNodo\_ArbolTri(me.ArbolTri,clave,pos);

end;

{-----------------------------------------------------------------------------------}

function PosNula\_Ganadores(var Me:ME\_GANADOR):TPos;

begin

PosNula\_Ganadores:= LO\_ArbolTrinario.PosNula\_ArbolTri(me.arbolTri);

end;

{-----------------------------------------------------------------------------------}

function Raiz (var Me: ME\_GANADOR): TPos;

begin

Raiz:= LO\_ArbolTrinario.Raiz\_ArbolTri(me.arbolTri);

end;

{-----------------------------------------------------------------------------------}

function PrimerJugada\_Ganador(var Me:ME\_GANADOR; nodo:LO\_ArbolTrinario.tNodoIndice):TPos;

var

pos: tpos;

begin

LO\_ArbolTrinario.BuscarNodo\_ArbolTri(ME.ArbolTri, nodo.clave, pos);

LO\_ArbolTrinario.CapturarNodo\_ArbolTri(ME.ArbolTri, nodo, pos);

PrimerJugada\_Ganador:= LO\_ListaParcialDobleEnlace.Primero(ME.ListaIndiceAlmacen, nodo.HijoMedio);

end;

{-----------------------------------------------------------------------------------}

function SiguienteJugada\_Ganador(var Me:ME\_GANADOR; pos:TPos): TPos;

begin

SiguienteJugada\_Ganador:=LO\_ListaParcialDobleEnlace.Proximo(ME.ListaIndiceAlmacen,pos);

end;

{-----------------------------------------------------------------------------------}

procedure CapturarJugada\_Ganador(var Me:ME\_GANADOR; pos:TPos; var reg:Type\_GANADOR.TipoRegDatos);

begin

Seek(Me.Almacen, pos);

Read(Me.Almacen, reg);

end;

{-----------------------------------------------------------------------------------}

function ProximoIzq (var Me:ME\_GANADOR; pos: TPos): TPos;

begin

ProximoIzq:= LO\_ArbolTrinario.HijoIzq\_ArbolTri(Me.ArbolTri, pos);

end;

{-----------------------------------------------------------------------------------}

function ProximoDer (var Me:ME\_GANADOR; pos: TPos): TPos;

begin

ProximoDer:= LO\_ArbolTrinario.HijoDer\_ArbolTri(Me.ArbolTri, pos);

end;

{-----------------------------------------------------------------------------------}

function Cantidad\_Ganadores(var Me: ME\_GANADOR): tCantidad;

begin

Cantidad\_Ganadores:= LO\_ArbolTrinario.CantidadNodos\_Arbol(me.ArbolTri);

end;

{-----------------------------------------------------------------------------------}

function ObtenerNivel\_enArbolTri(var Me:ME\_GANADOR; clave:tClave): tCantidad;

begin

ObtenerNivel\_enArbolTri:= LO\_ArbolTrinario.ObtenerNivel\_ArbolTri(Me.ArbolTri, clave);

end;

{-----------------------------------------------------------------------------------}

procedure Asignar(var Me:ME\_GANADOR);

begin

AssignFile(Me.Almacen, \_Directorio + \_NombreArchivoDatos);

AssignFile(me.ListaIndiceAlmacen.D, \_Directorio + \_NombreListaIndiceDatos);

AssignFile(Me.ArbolTri.D, \_Directorio + \_NombreArchIndice\_ArbolTrinario);

AssignFile(Me.ArbolTri.C, \_Directorio + \_NombreArchControl\_ArbolTrinario);

end;

END.

#### Type Ganador

unit Type\_GANADOR;

interface

USES

Sysutils;

CONST

//Valor del premio segun el nomenclador

Prem\_Cero = 0.5;

Prem\_Rojonegro = 1;

Prem\_ParImpar= 1;

Prem\_PasaFalta= 1;

Prem\_Docena= 2;

Prem\_Columna= 2;

Prem\_Pleno= 35;

TYPE

tNomenclador = string[2];

tNumero = -1..36;

tImporte = 0..maxint;

tNumeroJugada = 0.. maxint;

TipoRegDatos = Record

Nomenclador:tNomenclador;

Numero: tNumero; //Solo si el nomenclador es pleno (XX) el resto -1

Importe: tImporte;

NroJugada: tNumeroJugada;

end;

TipoArchivoDatos = File Of TipoRegDatos;

implementation

END.

# Proyecto

## Librería Auxiliar Juego

unit Lib\_AuxJuego;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

SysUtils, Vcl.Dialogs, Winapi.Windows, Winapi.Messages, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.ExtCtrls, Vcl.StdCtrls, Vcl.Buttons, Vcl.Grids, Math;

//Consultas sobre rango de valores a la que puede llegar a pertenecer la bolilla ganadora. Si se encuentra devuelve true

function IsIn\_PrimeraColumna(BolillaGanadora: tBolilla): boolean;

function IsIn\_SegundaColumna(BolillaGanadora: tBolilla): boolean;

function IsIn\_TerceraColumna(BolillaGanadora: tBolilla): boolean;

function IsIn\_PrimerDocena(BolillaGanadora: tbolilla): boolean;

function IsIn\_SegundaDocena(BolillaGanadora: tbolilla): boolean;

function IsIn\_TercerDocena(BolillaGanadora: tbolilla): boolean;

function IsIn\_PASA(BolillaGanadora: tbolilla): boolean;

function IsIn\_FALTA(BolillaGanadora: tbolilla): boolean;

function IsIn\_PAR(BolillaGanadora: tbolilla): boolean;

function IsIn\_IMPAR(BolillaGanadora: tbolilla): boolean;

function IsIn\_ROJOS(BolillaGanadora: tbolilla): boolean;

function IsIn\_NEGROS(BolillaGanadora: tbolilla): boolean;

//Procedimientos sobre ficticios

Procedure Insertar\_JugadoresFicticios(var Me: ME\_JUGADORES.ME\_JUGADOR; cant:tCantidad);

procedure Activar\_JugadoresFicticios(var Me: ME\_JUGADORES.ME\_JUGADOR; cantFict\_Activar:tCantidad);

procedure Desactivar\_JugadoresFicticios(RaizJugadores: Lib\_Auxiliar.tPos);

procedure Cant\_FicticiosActivos(RaizJugadores: Lib\_Auxiliar.tPos; var cant:tCantidad);

procedure Apuestas\_JugadoresFicticios(RaizJugadores: Lib\_Auxiliar.tPos; var cantFict\_Activos:tCantidad);

//Generales

procedure Ini\_Sesion (var Me: ME\_JUGADORES.ME\_JUGADOR; var regJugador: Type\_JUGADOR.tRegDatos);

procedure Fin\_Sesion (var Me: ME\_JUGADORES.ME\_JUGADOR; var regJugador: Type\_JUGADOR.tRegDatos);

procedure repartirPremio\_xJugada (RaizJugadores: Lib\_Auxiliar.tPos; nroJugada: Type\_JUEGO.tClave);

Function ExistenApuestas\_enEsteJuego(Num\_Jugada: Type\_JUEGO.tClave):boolean;

Procedure CantGanadores\_EnUnaJugada(RaizGanadores: Lib\_Auxiliar.tPos; jugada: Type\_JUEGO.TClave; var cantGanadores: tCantidad);

Procedure CantGanadores\_xNomenclador(RaizGanadores: Lib\_Auxiliar.tPos; nomenclador: tNomenclador; var cantGanadores: tCantidad);

Procedure MontoEnApuestaXPartida (NroJugada: Type\_JUEGO.TClave; var montoTotalApuestas: tImporte);

Procedure MontoPremiosXPartida(RaizGanadores: Lib\_Auxiliar.tPos; NroJugada: Type\_JUEGO.TClave; var montoTotalPremios: tImporte);

Procedure Cant\_RealesYFicticios\_xPartida(RaizJugadores: Lib\_Auxiliar.tPos; NroJugada: Type\_JUEGO.TClave; var cantReales: tCantidad; var cantFicticios: tCantidad);

procedure ReordenarMasPremiados (var grid: TStringGrid; num:integer; cantFilas: integer);

Procedure MovimientoHaberApostadores();

procedure recuperarRegPartida (Nro\_Jugada: Type\_JUEGO.tClave; var regBuscado: Type\_JUEGO.TipoRegDatos);

Procedure InsertoPremioUnGanador(regGanador: Type\_GANADOR.TipoRegDatos; var regApuesta: Type\_APUESTA.TipoRegDatos; tipoGanador: tNomenclador; Importe: tImporte);

Procedure MovimientoApuestaUnJugador(nick: tClave; regApuesta: Type\_APUESTA.TipoRegDatos; Importe: tImporte);

procedure ReordenarGridPorFecha (var grid: TStringGrid; cantFilas: integer);

procedure Cant\_JugadoresEnLinea(RaizJugadores: Lib\_Auxiliar.tPos; var cant:tCantidad);

Procedure CantidadJugadoresNuncaGanaron(RaizJugadores: Lib\_Auxiliar.tPos; var cant: tCantidad);

Procedure Premios\_AcumuladosXJugador(RaizGanadores: Lib\_Auxiliar.tPos; Nick:Type\_JUGADOR.tClave; var cantPremios: tCantidad; var Monto: tImporte);

Procedure Premios\_AcumuladosXJugadorUnaJugada(RaizGanadores: Lib\_Auxiliar.tPos; Nick:Type\_JUGADOR.tClave; jugada: Type\_JUEGO.TClave; var Monto: tImporte);

Function JugadorTieneApuestasEnPartida (NroJugada: Type\_JUEGO.TClave; nick: Type\_JUGADOR.tClave): boolean;

Procedure JugadorEsUnGanandor(pos:tPos; Nick: tClave; var esGanador: boolean);

Procedure EliminarApuestasUnJugador(Nick: tClave);

Procedure Cant\_JugadoresDesbloqueados(RaizJugadores: Lib\_Auxiliar.tPos; var cantDesbloq: Lib\_Auxiliar.tCantidad);

Procedure Cant\_JugadoresActivos(RaizJugadores: Lib\_Auxiliar.tPos; var cantActivos: Lib\_Auxiliar.tCantidad);

//Listados

Procedure ListadoGeneral\_JugadoresEnGrid(RaizJugadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

Procedure Listado\_JugadoresActivos(RaizJugadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

Procedure ListadoArbolGanadores(RaizGanadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

Procedure ListadoArbolJugadores(RaizJugadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

procedure ListadoPartidas (var grid: TStringGrid);

Procedure ListadoJugadoresNuncaGanaron(RaizJugadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

procedure ListadoCtaCte\_deUnJugador(posListaParcial: tPos; Nick: tclave; var grid: TStringGrid);

Procedure ListadoPremiados\_Nomenclador(RaizGanadores: Lib\_Auxiliar.tPos; nomenclador: tNomenclador; var grid: TStringGrid);

Procedure ListadoMasPremiados(RaizGanadores: Lib\_Auxiliar.tPos; var importe:tImporte; var grid: TStringGrid; var cantFilas: integer);

Procedure Listado\_GeneralApuestas(var grid: TStringGrid; Nro\_Jugada: Type\_JUEGO.tClave);

Procedure Listado\_ApuestasUnJugador(var grid: TStringGrid; Nro\_Jugada: Type\_JUEGO.tClave; nick: Type\_JUGADOR.TClave);

Procedure Listado\_Premios(RaizGanadores: Lib\_Auxiliar.tPos; jugada: Type\_JUEGO.TClave; var grid: TStringGrid);

//grid

function Mensaje\_Confirmacion(Msje: String): boolean;

procedure limpiarGrid(grid: TStringGrid);

Procedure EncabezadoDetalladoJugadores (grid: TStringGrid);

Procedure EncabezadoArbolJugadores(grid: TStringGrid);

Procedure EncabezadoListadoPartidas(grid: TStringGrid);

Procedure EncabezadoJugadoresMasGanaron(grid: TStringGrid);

Procedure EncabezadoGanadoresXPremio(grid: TStringGrid);

Procedure EncabezadoFiltroCtaCte(grid: TStringGrid);

Procedure EncabezadosComunApuestas(grid: TStringGrid);

Procedure EncabezadosComunJugadores(grid: TStringGrid);

Procedure EncabezadoGridPremios(grid: TStringGrid);

Procedure EncabezadosComunFicticios(grid: TStringGrid);

IMPLEMENTATION

uses

Form\_Login;

{--------------------------------------------------------------------------------}

//Inserto cantidad parametrizada de jugadores ficticios

Procedure Insertar\_JugadoresFicticios(var Me: ME\_JUGADORES.ME\_JUGADOR; cant:tCantidad);

var

numFicticio: Lib\_Auxiliar.tCantidad;

posJugador: tpos;

regJugador: type\_JUGADOR.tRegDatos;

RegCtaCte: Type\_ALMACEN.tipoRegAlmacen;

i: Integer;

largo:integer;

cadena:string;

pad:char;

begin

pad:='0';

for I := 1 to cant do

begin

//Cuando el croupier genera un nuevo jugador ficticio, se les asigna el nick

//que surja de la combinación "X\_" + str (<ficticio>) completado con ceros la parte no significativa.

//Por ejemplo: X\_0001, X\_0002, etc.

numFicticio:= UltFicticio(Me) + 1;

regJugador.Nick:= Lib\_Auxiliar.Fict\_IniNick + intTostr(numFicticio).PadLeft(4,pad);

regJugador.Nombre:='N\_Ficticio';

regJugador.Apellido:='A\_Ficticio';

regJugador.Contrasenia:='';

regJugador.Alta:= Now();

regJugador.UltimaConexion:= Now();

regJugador.TipoJugador:= false;

regJugador.Estado:= false;

regJugador.Bloqueado:= false;

BuscarInfoME\_Jugadores(Me,posJugador,regJugador.Nick);

InsertarInfoME\_Jugadores(Me,regJugador,posJugador);

//Cargo datos de ctacte a ficticios

RegCtaCte.Nick:= regJugador.Nick;

RegCtaCte.FechaHora:= Now();

RegCtaCte.Concepto:= Type\_ALMACEN.tipoConceptos.Creacion\_Cuenta;

RegCtaCte.Debe:= ME\_JUEGO.ApuestaMinima(Form\_Login.ME\_JUEGORULETA)\*10;

RegCtaCte.haber:= 0;

RegCtaCte.saldo:= ME\_JUEGO.ApuestaMinima(Form\_Login.ME\_JUEGORULETA)\*10;

//inserto cuenta corriente de ficticios

ME\_CTACTE.InsertarAlmacen(Form\_Login.ME\_CUENTACORRIENTE,RegCtaCte);

end;

end;

{------------------------------------------------------------------------------}

procedure Activar\_JugadoresFicticios(var Me: ME\_JUGADORES.ME\_JUGADOR; cantFict\_Activar:tCantidad);

var

RD:Type\_JUGADOR.tRegDatos;

CantFictio, esteFicticio:tCantidad;

pad:char;

estado:string;

begin

Randomize;

if cantFict\_Activar>0 then

begin

//Nick random ficticio con el campo guardo en RC de ultimoFicticio

pad:='0';

CantFictio:= ME\_JUGADORES.UltFicticio(ME\_JUGADOR);

esteFicticio:= random(CantFictio) + 1;

RD.Nick:= Lib\_Auxiliar.Fict\_IniNick + intTostr(esteFicticio).PadLeft(4,pad);

ME\_JUGADORES.BuscarInfoME\_Jugadores(Form\_Login.ME\_JUGADOR,posJugador,RD.Nick);

ME\_JUGADORES.CapturarInfoME\_Jugadores(Form\_Login.ME\_JUGADOR,RD,posJugador);

if (not RD.Estado)then

begin

RD.Nick:= RD.Nick;

RD.Contrasenia:= RD.Contrasenia;

RD.Nombre:= RD.Nombre;

RD.Apellido:= RD.Apellido;

RD.Alta:= RD.Alta;

RD.UltimaConexion:= Now(); // Última conexión actual

RD.Bloqueado:= RD.Bloqueado;

RD.Estado:= true; // activo jugador ficticio

ME\_JUGADORES.ModificarInfoME\_Jugadores(Form\_Login.ME\_JUGADOR, RD, posJugador);

Activar\_JugadoresFicticios(Form\_Login.ME\_JUGADOR, cantFict\_Activar - 1);

end

else

Activar\_JugadoresFicticios(Form\_Login.ME\_JUGADOR, cantFict\_Activar);

end;

end;

{------------------------------------------------------------------------------}

procedure Desactivar\_JugadoresFicticios(RaizJugadores: Lib\_Auxiliar.tPos);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula)then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if ((not RD.TipoJugador) and (RD.Estado)) then //ficticio y activo

begin

RD.Nick:= RD.Nick;

RD.Contrasenia:= RD.Contrasenia;

RD.Nombre:= RD.Nombre;

RD.Apellido:= RD.Apellido;

RD.Alta:= RD.Alta;

RD.UltimaConexion:= Now(); // Última conexión actual

RD.Bloqueado:= RD.Bloqueado;

RD.Estado:= false; // desactivo jugador ficticio

ME\_JUGADORES.ModificarInfoME\_Jugadores(Form\_Login.ME\_JUGADOR, RD, RaizJugadores);

end;

Desactivar\_JugadoresFicticios(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores));

Desactivar\_JugadoresFicticios(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores));

end;

end;

{------------------------------------------------------------------------------}

procedure Cant\_FicticiosActivos(RaizJugadores: Lib\_Auxiliar.tPos; var cant:tCantidad);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula)then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if ((not RD.TipoJugador) and (RD.Estado)) then //ficticio y activo

cant:= cant + 1;

Cant\_FicticiosActivos(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), cant);

Cant\_FicticiosActivos(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), cant);

end;

end;

{------------------------------------------------------------------------------}

procedure Cant\_JugadoresEnLinea(RaizJugadores: Lib\_Auxiliar.tPos; var cant:tCantidad);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula)then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if ((RD.TipoJugador) and (RD.Estado) and (RD.Nick<>'ADMINISTRADOR')) then //real, activo. Administrador no suma

cant:= cant + 1;

Cant\_JugadoresEnLinea(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), cant);

Cant\_JugadoresEnLinea(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), cant);

end;

end;

{------------------------------------------------------------------------------}

procedure Apuestas\_JugadoresFicticios(RaizJugadores: Lib\_Auxiliar.tPos; var cantFict\_Activos:tCantidad);

var

regJF, RD:Type\_JUGADOR.tRegDatos; //registro Jugador Ficticio

regAF: Type\_APUESTA.TipoRegDatos; //registro Apuesta Ficticio

regJuego: Type\_JUEGO.TipoRegDatos;

rNomenclador, rValor: integer; //nomenclador random, valor random

posAlmacenFict: tPos;

regAlmacenFict: Type\_ALMACEN.tipoRegAlmacen;

saldoFict, importeFict: tImporte;

importeValido: boolean;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

if (cantFict\_Activos > 0) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if ((not RD.TipoJugador) and (RD.Estado) and (not RD.Bloqueado)) then //ficticio, activo, no bloqueado

begin

Randomize;

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuego);

regAF.NroJugada:= regJuego.NroJugada;

regAF.Nick:= RD.Nick;

rNomenclador:= random(6) + 1;

CASE (rNomenclador) of

1:begin // Pleno

Randomize;

regAF.Nomenclador:= type\_APUESTA.Nom\_Pleno;

regAF.Valor:= inttostr(random(36));

end;

2:begin // Columna

Randomize;

regAF.Nomenclador:= type\_APUESTA.Nom\_Columna;

rValor:= random(3) + 1;

case (rValor) of

1:regAF.Valor:= '1';

2:regAF.Valor:= '2';

3:regAF.Valor:= '3';

end; //case rValor

end;

3:begin // Docena

Randomize;

regAF.Nomenclador:= type\_APUESTA.Nom\_Docena;

rValor:= random(3) + 1;

case (rValor) of

1:regAF.Valor:= '1';

2:regAF.Valor:= '2';

3:regAF.Valor:= '3';

end; //case rValor

end;

4:begin // Pasa o Falta

Randomize;

regAF.Nomenclador:= type\_APUESTA.Nom\_PasaFalta;

rValor:= random(2) + 1;

if (rValor = 1) then

regAF.Valor:= 'P'

else

regAF.Valor:= 'F';

end;

5:begin // Par o Impar

Randomize;

regAF.Nomenclador:= type\_APUESTA.Nom\_ParImpar;

rValor:= random(2) + 1;

if (rValor = 1) then

regAF.Valor:= 'P'

else

regAF.Valor:= 'I';

end;

6:begin //Rojo o Negro

Randomize;

regAF.Nomenclador:= type\_APUESTA.Nom\_Rojonegro;

rValor:= random(2) + 1;

if (rValor = 1) then

regAF.Valor:= 'R'

else

regAF.Valor:= 'N';

end;

END;//case

saldoFict:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, RD.Nick);

//Si saldo de ficticio es mayor que la apuesta minina

if (saldoFict >= ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA)) then

begin

Randomize;

//busco un importe que sea un rango aleatorio que este entre la apuesta minima y la maxima permitida

importeFict := RANDOMRANGE(ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA),ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA));

importeValido:= false;

while (not importeValido) do

begin

//El IMPORTE A APOSTAR del ficticio es valida si es menor o igual a su saldo, mayor o igual a la apuesta minima, menor o igual a la apueta maxima.

if ((importeFict <= saldoFict) and (importeFict >= ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA)) and (importeFict <= ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA))) then

importeValido:= true

else

//si el importe a apostar no es valido, busco de nuevo un rango que este entre la apuesta minima y la maxima permitida

importeFict := RANDOMRANGE(ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA),ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA));

end;//while

regAF.Importe:= importeFict;

ME\_APUESTAS.InsertarInfoME\_Apuestas(form\_login.ME\_APUESTA, regAF, ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA));

cantFict\_Activos:= cantFict\_Activos-1;

end;//If

end;

end;

Apuestas\_JugadoresFicticios(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), cantFict\_Activos);

Apuestas\_JugadoresFicticios(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), cantFict\_Activos);

end;

end;

{------------------------------------------------------------------------------}

procedure repartirPremio\_xJugada (RaizJugadores: Lib\_Auxiliar.tPos; nroJugada: Type\_JUEGO.tClave);

var

i:tPos;

regJugador, RD:Type\_JUGADOR.tRegDatos;

regJuego: Type\_JUEGO.TipoRegDatos;

regApuesta: Type\_APUESTA.TipoRegDatos;

regGanador: Type\_GANADOR.TipoRegDatos;

regCtaCte: Type\_ALMACEN.tipoRegAlmacen;

importe: tImporte;

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuego);

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

//capturo jugador para verificar si tiene alguna apuesta

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, RaizJugadores);

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while (i<>ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA)) do

begin

//capturo apuesta del jugador

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA,i, regApuesta);

//si la apuesta apuesta es de la misma jugada y mismo jugador

if ((regApuesta.NroJugada = regJuego.NroJugada) and (regApuesta.Nick = regJugador.Nick)) then

begin //busco si coinciden sus apuestas con el resultado de la jugada

if (regApuesta.Nomenclador = Type\_APUESTA.Nom\_Pleno) then //PLENO

begin

if (regApuesta.Valor = intTostr(regJuego.Bolilla)) then

begin

//Calculo el importe que gano el apostador en dicha jugada

Importe:= regApuesta.Importe \* type\_GANADOR.Prem\_Pleno;

//Invoco a PremioAunGanador que completa el registro de ganador y lo inserta

InsertoPremioUnGanador(regGanador, regApuesta, Type\_APUESTA.Nom\_Pleno, Importe);

//Inserto movimiento de premio de un jugador en CtaCte

MovimientoApuestaUnJugador(regJugador.Nick, regApuesta, Importe);

end;

end;

if (regApuesta.Nomenclador = Type\_APUESTA.Nom\_Columna) then // COLUMNA

begin

if (((regApuesta.Valor = '1') and (IsIn\_PrimeraColumna(regJuego.Bolilla))) or ((regApuesta.Valor = '2') and (IsIn\_SegundaColumna(regJuego.Bolilla))) or ((regApuesta.Valor = '3') and (IsIn\_TerceraColumna(regJuego.Bolilla)))) then

begin

//Calculo el importe que gano el apostador en dicha jugada

Importe:= regApuesta.Importe \* type\_GANADOR.Prem\_Pleno;

//Invoco a PremioAunGanador que completa el registro de ganador y lo inserta

InsertoPremioUnGanador(regGanador, regApuesta, Type\_APUESTA.Nom\_Columna, Importe);

//Inserto movimiento de premio de un jugador en CtaCte

MovimientoApuestaUnJugador(regJugador.Nick, regApuesta, Importe);

end;

end;

if (regApuesta.Nomenclador = Type\_APUESTA.Nom\_Docena) then // DOCENA

begin

if (((regApuesta.Valor = '1') and (IsIn\_PrimerDocena(regJuego.Bolilla))) or ((regApuesta.Valor = '2') and (IsIn\_SegundaDocena(regJuego.Bolilla))) or ((regApuesta.Valor = '3') and (IsIn\_TercerDocena(regJuego.Bolilla)))) then

begin

//Calculo el importe que gano el apostador en dicha jugada

Importe:= regApuesta.Importe \* type\_GANADOR.Prem\_Pleno;

//Invoco a PremioAunGanador que completa el registro de ganador y lo inserta

InsertoPremioUnGanador(regGanador, regApuesta, Type\_APUESTA.Nom\_Docena, Importe);

//Inserto movimiento de premio de un jugador en CtaCte

MovimientoApuestaUnJugador(regJugador.Nick, regApuesta, Importe);

end;

end;

if (regApuesta.Nomenclador = Type\_APUESTA.Nom\_PasaFalta) then // PASA-FALTA

begin

if (((regApuesta.Valor = 'P') and (IsIn\_PASA(regJuego.Bolilla))) or ((regApuesta.Valor = 'F') and (IsIn\_FALTA(regJuego.Bolilla)))) then

begin

//Calculo el importe que gano el apostador en dicha jugada

Importe:= regApuesta.Importe \* type\_GANADOR.Prem\_Pleno;

//Invoco a PremioAunGanador que completa el registro de ganador y lo inserta

InsertoPremioUnGanador(regGanador, regApuesta, Type\_APUESTA.Nom\_PasaFalta, Importe);

//Inserto movimiento de premio de un jugador en CtaCte

MovimientoApuestaUnJugador(regJugador.Nick, regApuesta, Importe);

end;

end;

if (regApuesta.Nomenclador = Type\_APUESTA.Nom\_ParImpar) then // PAR-IMPAR

begin

if (((regApuesta.Valor = 'P') and (IsIn\_PAR(regJuego.Bolilla))) or ((regApuesta.Valor = 'I') and (IsIn\_IMPAR(regJuego.Bolilla)))) then

begin

//Calculo el importe que gano el apostador en dicha jugada

Importe:= regApuesta.Importe \* type\_GANADOR.Prem\_Pleno;

//Invoco a PremioAunGanador que completa el registro de ganador y lo inserta

InsertoPremioUnGanador(regGanador, regApuesta, Type\_APUESTA.Nom\_ParImpar, Importe);

//Inserto movimiento de premio de un jugador en CtaCte

MovimientoApuestaUnJugador(regJugador.Nick, regApuesta, Importe);

end;

end;

if (regApuesta.Nomenclador = Type\_APUESTA.Nom\_Rojonegro) then // ROJO-NEGRO

begin

if (((regApuesta.Valor = 'R') and (IsIn\_ROJOS(regJuego.Bolilla))) or ((regApuesta.Valor = 'N') and (IsIn\_NEGROS(regJuego.Bolilla)))) then

begin

//Calculo el importe que gano el apostador en dicha jugada

Importe:= regApuesta.Importe \* type\_GANADOR.Prem\_Pleno;

//Invoco a PremioAunGanador que completa el registro de ganador y lo inserta

InsertoPremioUnGanador(regGanador, regApuesta, Type\_APUESTA.Nom\_Rojonegro, Importe);

//Inserto movimiento de premio de un jugador en CtaCte

MovimientoApuestaUnJugador(regJugador.Nick, regApuesta, Importe);

end;

end;

//Si la bola cae sobre el cero los jugadores que hicieron apuestas sencillas (rojo/negro, par/impar, pasa/falta)

//recuperan la mitad de su apuesta y pierden la otra mitad.

if (regJuego.Bolilla = 0) and ((regApuesta.Nomenclador = Type\_APUESTA.Nom\_Columna) or

(regApuesta.Nomenclador = Type\_APUESTA.Nom\_Docena) or

(regApuesta.Nomenclador = Type\_APUESTA.Nom\_PasaFalta) or

(regApuesta.Nomenclador = Type\_APUESTA.Nom\_ParImpar) or

(regApuesta.Nomenclador = Type\_APUESTA.Nom\_Rojonegro)) then

begin

//completo registro de ganador

regGanador.Nomenclador:= regApuesta.Nomenclador;

regGanador.Numero:= strToint('0');

regGanador.NroJugada:= strToint(regApuesta.NroJugada);

regGanador.Importe:= round(regApuesta.Importe\*type\_GANADOR.Prem\_Cero);

//Inserto ganador

ME\_GANADORES.InsertarInfoME\_Ganadores(ME\_GANADOR,regGanador, regApuesta.Nick);

//Completo registro movimiento en CtaCte

regCtaCte.Nick:= regJugador.Nick;

regCtaCte.FechaHora:= Now();

regCtaCte.Concepto:= Type\_ALMACEN.\_PremioJuego + regApuesta.NroJugada;

regCtaCte.Debe:= regGanador.Importe;

regCtaCte.haber:= 0;

regCtaCte.saldo:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, regApuesta.Nick) + regGanador.Importe;

//Inserto movimiento en CtaCte

ME\_CTACTE.InsertarAlmacen(ME\_CUENTACORRIENTE, regCtaCte);

end;

end;

i:= ME\_APUESTAS.Proximo(ME\_APUESTA, i)

end;//while

repartirPremio\_xJugada(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR, RaizJugadores), nroJugada);

repartirPremio\_xJugada(ME\_JUGADORES.ProximoDer(ME\_JUGADOR, RaizJugadores), nroJugada);

end;

end;

{------------------------------------------------------------------------------}

function IsIn\_PrimeraColumna(BolillaGanadora: tBolilla): boolean;

begin

IsIn\_PrimeraColumna:= (BolillaGanadora = 1) or (BolillaGanadora = 4) or (BolillaGanadora = 7)

or (BolillaGanadora = 10) or (BolillaGanadora = 13) or (BolillaGanadora = 16) or (BolillaGanadora = 19)

or (BolillaGanadora = 22) or (BolillaGanadora = 25) or (BolillaGanadora = 28) or (BolillaGanadora = 31) or (BolillaGanadora = 34);

end;

{------------------------------------------------------------------------------}

function IsIn\_SegundaColumna(BolillaGanadora: tBolilla): boolean;

begin

IsIn\_SegundaColumna:= (BolillaGanadora = 2) or (BolillaGanadora = 5) or (BolillaGanadora = 8)

or (BolillaGanadora = 11) or (BolillaGanadora = 14) or (BolillaGanadora = 17) or (BolillaGanadora = 20)

or (BolillaGanadora = 23) or (BolillaGanadora = 26) or (BolillaGanadora = 29) or (BolillaGanadora = 32) or (BolillaGanadora = 35);

end;

{------------------------------------------------------------------------------}

function IsIn\_TerceraColumna(BolillaGanadora: tBolilla): boolean;

begin

IsIn\_TerceraColumna:= (BolillaGanadora = 3) or (BolillaGanadora = 6) or (BolillaGanadora = 9)

or (BolillaGanadora = 12) or (BolillaGanadora = 15) or (BolillaGanadora = 18) or (BolillaGanadora = 21)

or (BolillaGanadora = 24) or (BolillaGanadora = 27) or (BolillaGanadora = 30) or (BolillaGanadora = 33) or (BolillaGanadora = 36);

end;

{------------------------------------------------------------------------------}

function IsIn\_PrimerDocena(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_PrimerDocena:= (BolillaGanadora>=1) and (BolillaGanadora<=12);

end;

{------------------------------------------------------------------------------}

function IsIn\_SegundaDocena(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_SegundaDocena:= (BolillaGanadora>=13) and (BolillaGanadora<=24);

end;

{------------------------------------------------------------------------------}

function IsIn\_TercerDocena(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_TercerDocena:= (BolillaGanadora>=25) and (BolillaGanadora<=38);

end;

{------------------------------------------------------------------------------}

function IsIn\_PASA(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_PASA:= (BolillaGanadora>0) and (BolillaGanadora<=18);

end;

{------------------------------------------------------------------------------}

function IsIn\_FALTA(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_FALTA:= (BolillaGanadora>18) and (BolillaGanadora<=38);

end;

{------------------------------------------------------------------------------}

function IsIn\_PAR(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_PAR:= (BolillaGanadora = 2) or (BolillaGanadora = 4) or (BolillaGanadora = 6)

or (BolillaGanadora = 8) or (BolillaGanadora = 10) or (BolillaGanadora = 12) or (BolillaGanadora = 14)

or (BolillaGanadora = 16) or (BolillaGanadora = 18) or (BolillaGanadora = 20) or (BolillaGanadora = 22) or (BolillaGanadora = 24)

or (BolillaGanadora = 26) or (BolillaGanadora = 28) or (BolillaGanadora = 30) or (BolillaGanadora = 32) or (BolillaGanadora = 34)

or (BolillaGanadora = 36);

end;

{------------------------------------------------------------------------------}

function IsIn\_IMPAR(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_IMPAR:= (BolillaGanadora = 1) or (BolillaGanadora = 3) or (BolillaGanadora = 5)

or (BolillaGanadora = 7) or (BolillaGanadora = 9) or (BolillaGanadora = 11) or (BolillaGanadora = 13)

or (BolillaGanadora = 15) or (BolillaGanadora = 17) or (BolillaGanadora = 19) or (BolillaGanadora = 21) or (BolillaGanadora = 23)

or (BolillaGanadora = 25) or (BolillaGanadora = 27) or (BolillaGanadora = 29) or (BolillaGanadora = 31) or (BolillaGanadora = 33)

or (BolillaGanadora = 35);

end;

{------------------------------------------------------------------------------}

//1,3,5,7,9,12,14,16,18,19,21,23,25,27,30,32,34,36 (rojos)

function IsIn\_ROJOS(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_ROJOS:= (BolillaGanadora = 1) or (BolillaGanadora = 3) or (BolillaGanadora = 5)

or (BolillaGanadora = 7) or (BolillaGanadora = 9) or (BolillaGanadora = 12) or (BolillaGanadora = 14)

or (BolillaGanadora = 16) or (BolillaGanadora = 18) or (BolillaGanadora = 19) or (BolillaGanadora = 21) or (BolillaGanadora = 23)

or (BolillaGanadora = 25) or (BolillaGanadora = 27) or (BolillaGanadora = 30) or (BolillaGanadora = 32) or (BolillaGanadora = 34)

or (BolillaGanadora = 36);

end;

{------------------------------------------------------------------------------}

//2,4,6,8,10,11,13,15,17,20,22,24,26,28,29,31,33,35 (negros)

function IsIn\_NEGROS(BolillaGanadora: tbolilla): boolean;

begin

IsIn\_NEGROS:= (BolillaGanadora = 2) or (BolillaGanadora = 4) or (BolillaGanadora = 6)

or (BolillaGanadora = 8) or (BolillaGanadora = 10) or (BolillaGanadora = 11) or (BolillaGanadora = 13)

or (BolillaGanadora = 15) or (BolillaGanadora = 17) or (BolillaGanadora = 20) or (BolillaGanadora = 22) or (BolillaGanadora = 24)

or (BolillaGanadora = 26) or (BolillaGanadora = 28) or (BolillaGanadora = 29) or (BolillaGanadora = 31) or (BolillaGanadora = 33)

or (BolillaGanadora = 35);

end;

{------------------------------------------------------------------------------}

Function ExistenApuestas\_enEsteJuego(Num\_Jugada: Type\_JUEGO.tClave):boolean;

var

i:tPos;

encontre:boolean;

begin

encontre:= false;

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while ((i<>ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA)) and (not encontre)) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA,i,regApuesta);

if (regApuesta.NroJugada = Num\_Jugada) then

begin

encontre:= true;

ExistenApuestas\_enEsteJuego:= true;

end

else

i:= ME\_APUESTAS.Proximo(ME\_APUESTA,i);

END; //while

end;

{------------------------------------------------------------------------------}

Procedure CantGanadores\_EnUnaJugada(RaizGanadores: Lib\_Auxiliar.tPos; jugada: Type\_JUEGO.TClave; var cantGanadores: tCantidad);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

i:= ME\_GANADORES.PrimerJugada\_Ganador(ME\_GANADOR, RegNodoGanador);

while i<> ME\_GANADORES.PosNula\_Ganadores(ME\_GANADOR) do

begin

ME\_GANADORES.CapturarJugada\_Ganador(ME\_GANADOR, i, RegAlmacen);

if (intTostr(RegAlmacen.NroJugada) = jugada) then

cantGanadores:= cantGanadores + 1;

i:= ME\_GANADORES.SiguienteJugada\_Ganador(ME\_GANADOR, i);

end; //while

CantGanadores\_EnUnaJugada(ME\_GANADORES.ProximoIzq(ME\_GANADOR, RaizGanadores), jugada, cantGanadores);

CantGanadores\_EnUnaJugada(ME\_GANADORES.ProximoDer(ME\_GANADOR, RaizGanadores), jugada, cantGanadores);

end;

end;

{------------------------------------------------------------------------------}

Procedure CantGanadores\_xNomenclador(RaizGanadores: Lib\_Auxiliar.tPos; nomenclador: tNomenclador; var cantGanadores: tCantidad);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

i:= ME\_GANADORES.PrimerJugada\_Ganador(ME\_GANADOR, RegNodoGanador);

while i<> ME\_GANADORES.PosNula\_Ganadores(ME\_GANADOR) do

begin

ME\_GANADORES.CapturarJugada\_Ganador(ME\_GANADOR, i, RegAlmacen);

if (RegAlmacen.Nomenclador = nomenclador) then

cantGanadores:= cantGanadores + 1;

i:= ME\_GANADORES.SiguienteJugada\_Ganador(ME\_GANADOR, i);

end; //while

CantGanadores\_xNomenclador(ME\_GANADORES.ProximoIzq(ME\_GANADOR, RaizGanadores), nomenclador, cantGanadores);

CantGanadores\_xNomenclador(ME\_GANADORES.ProximoDer(ME\_GANADOR, RaizGanadores), nomenclador, cantGanadores);

end;

end;

{--------------------------------------------------------------------------------}

procedure MontoEnApuestaXPartida (NroJugada: Type\_JUEGO.TClave; var montoTotalApuestas: tImporte);

var

i: tpos;

regApuesta: Type\_APUESTA.TipoRegDatos;

begin

montoTotalApuestas:= 0;

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while i<>ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA,i,regApuesta);

if (regapuesta.NroJugada= nrojugada) then //pregunto si es la misma jugada que le paso

montoTotalApuestas:= montoTotalApuestas + regapuesta.Importe; //voy juntando el dinero de la partida

i:= ME\_APUESTAS.Proximo(ME\_APUESTA,i);

end;

end;

{--------------------------------------------------------------------------------}

procedure MontoPremiosXPartida(RaizGanadores: Lib\_Auxiliar.tPos; NroJugada: Type\_JUEGO.TClave; var montoTotalPremios: tImporte);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

i:= ME\_GANADORES.PrimerJugada\_Ganador(ME\_GANADOR, RegNodoGanador);

while i<> ME\_GANADORES.PosNula\_Ganadores(ME\_GANADOR) do

begin

ME\_GANADORES.CapturarJugada\_Ganador(ME\_GANADOR, i, RegAlmacen);

if (intTostr(RegAlmacen.NroJugada) = NroJugada) then

montoTotalPremios:= montoTotalPremios + RegAlmacen.Importe;

i:= ME\_GANADORES.SiguienteJugada\_Ganador(ME\_GANADOR, i);

end; //while

MontoPremiosXPartida(ME\_GANADORES.ProximoIzq(ME\_GANADOR, RaizGanadores), NroJugada, montoTotalPremios);

MontoPremiosXPartida(ME\_GANADORES.ProximoDer(ME\_GANADOR, RaizGanadores), NroJugada, montoTotalPremios);

end;

end;

{--------------------------------------------------------------------------------}

Procedure Cant\_RealesYFicticios\_xPartida(RaizJugadores: Lib\_Auxiliar.tPos; NroJugada: Type\_JUEGO.TClave; var cantReales: tCantidad; var cantFicticios: tCantidad);

var

RD:Type\_JUGADOR.tRegDatos;

i: tPos;

enc: boolean;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

enc:= false;

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while i<>ME\_APUESTAS.PosNula\_Apuestas(ME\_aPUESTA) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_aPUESTA,i,regApuesta);

if (regApuesta.NroJugada = NroJugada) and (regApuesta.Nick = RD.Nick) and not(enc) then

begin

if RD.TipoJugador then

cantReales:= cantReales + 1

else

cantFicticios:= cantFicticios + 1;

enc:= true;

end;

i:= ME\_APUESTAS.Proximo(ME\_aPUESTA,i);

end; //while

Cant\_RealesYFicticios\_xPartida(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), NroJugada, cantReales, cantFicticios);

Cant\_RealesYFicticios\_xPartida(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), NroJugada, cantReales, cantFicticios);

end;

end;

{--------------------------------------------------------------------------------}

Procedure JugadorEsUnGanandor(pos:tPos; Nick: tClave; var esGanador: boolean);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (pos<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, pos);

if (RegNodoGanador.clave = Nick) then

esGanador:=true;

JugadorEsUnGanandor(ME\_GANADORES.ProximoIzq(ME\_GANADOR, pos), Nick, esGanador);

JugadorEsUnGanandor(ME\_GANADORES.ProximoDer(ME\_GANADOR, pos), Nick, esGanador);

end;

end;

{--------------------------------------------------------------------------------}

procedure ReordenarMasPremiados (var grid: TStringGrid; num:integer; cantFilas: integer);

var

i, j, saldoA, saldoB: integer;

nickA, nickB: Lib\_Auxiliar.tClave;

begin

for I := 1 to cantFilas-1 do

for j := 1 to cantFilas-1 do

begin

SaldoA:= strtoint(grid.Cells[1,i]);

SaldoB:= strtoint(grid.Cells[1,j]);

NickA:= grid.Cells [0,i];

NickB:= grid.Cells [0,j];

if (saldoA>saldoB) then

begin

grid.Cells[1,i]:=inttostr(saldoB);

grid.Cells[0,i]:=nickB;

grid.Cells[1,j]:=inttostr(saldoA);

grid.Cells[0,j]:=nickA;

end;

end;

Grid.RowCount:= num + 1;

end;

{--------------------------------------------------------------------------------}

Procedure MovimientoHaberApostadores();

var

RegApuesta: type\_APUESTA.TipoRegDatos;

RegCtaCte: Type\_ALMACEN.tipoRegAlmacen;

i: tpos;

begin

//Recorro el ME\_APUESTAS capturando las apuestas de la ultima jugada e inserto movimientos del jugador con el importe apostado en su haber

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuegoRuleta);

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while i<> ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA,i, regApuesta);

if (regApuesta.NroJugada = regJuegoRuleta.NroJugada) then

begin

RegCtaCte.Nick:= Uppercase(regApuesta.Nick);

RegCtaCte.FechaHora:= now();

RegCtaCte.Concepto:= Type\_ALMACEN.tipoConceptos.Numero\_Apuesta + regJuegoRuleta.NroJugada;

RegCtaCte.Debe:= 0;

RegCtaCte.haber:= regApuesta.Importe;

RegCtaCte.saldo:= (ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, Uppercase(regApuesta.Nick)) - regApuesta.Importe);

ME\_CTACTE.InsertarAlmacen(ME\_CUENTACORRIENTE, RegCtaCte);

end;

i:= ME\_APUESTAS.Proximo(ME\_APUESTA,i);

end;

end;

{--------------------------------------------------------------------------------}

Procedure EliminarApuestasUnJugador(Nick: tClave);

var

RegApuesta: type\_APUESTA.TipoRegDatos;

RegCtaCte: Type\_ALMACEN.tipoRegAlmacen;

i: tpos;

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuegoRuleta);

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while i<> ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA,i, regApuesta);

if (regApuesta.Nick = Nick) then

begin

if (regApuesta.NroJugada = regJuegoRuleta.NroJugada) then

ME\_APUESTAS.EliminarInfoME\_Apuestas(ME\_APUESTA,i);

end;

i:= ME\_APUESTAS.Proximo(ME\_APUESTA,i);

end;

end;

{--------------------------------------------------------------------------------}

Procedure Cant\_JugadoresDesbloqueados(RaizJugadores: Lib\_Auxiliar.tPos; var cantDesbloq: Lib\_Auxiliar.tCantidad);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if (RD.Nick <> 'ADMINISTRADOR') then

begin

if not (RD.Bloqueado) then

cantDesbloq:= cantDesbloq + 1;

end;

Cant\_JugadoresDesbloqueados(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), cantDesbloq);

Cant\_JugadoresDesbloqueados(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), cantDesbloq);

end;

end;

{--------------------------------------------------------------------------------}

Procedure Cant\_JugadoresActivos(RaizJugadores: Lib\_Auxiliar.tPos; var cantActivos: Lib\_Auxiliar.tCantidad);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if (RD.Nick <> 'ADMINISTRADOR') and (RD.Estado) then //si no es el administrador y esta activo...

cantActivos:= cantActivos + 1;

Cant\_JugadoresActivos(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), cantActivos);

Cant\_JugadoresActivos(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), cantActivos);

end;

end;

{--------------------------------------------------------------------------------}

Procedure ListadoGeneral\_JugadoresEnGrid(RaizJugadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if (RD.Nick<>'ADMINISTRADOR') then

begin

grid.RowCount:= grid.RowCount + 1;

grid.Cells[0, grid.RowCount-1] := RD.Nombre;

grid.Cells[1, grid.RowCount-1] := RD.Apellido;

grid.Cells[2, grid.RowCount-1] := RD.Nick;

if RD.TipoJugador then

grid.Cells[3, grid.RowCount-1] :='Real'

else

grid.Cells[3, grid.RowCount-1] :='Ficticio';

if (RD.Bloqueado) then

grid.Cells[4, grid.RowCount-1] :='Bloqueado'

else

grid.Cells[4, grid.RowCount-1] :='Desbloqueado';

if (RD.Estado) then

grid.Cells[5, grid.RowCount-1] :='SI'

else

grid.Cells[5, grid.RowCount-1] :='NO';

grid.Cells[6, grid.RowCount-1] := datetimetostr(RD.Alta);

grid.Cells[7, grid.RowCount-1] := datetimetostr(RD.UltimaConexion);

grid.Cells[8, grid.RowCount-1]:= inttostr(ME\_CTACTE.SaldoAcumulado(me\_cuentacorriente,RD.Nick));

end;

ListadoGeneral\_JugadoresEnGrid(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), grid);

ListadoGeneral\_JugadoresEnGrid(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), grid);

end;

end;

{--------------------------------------------------------------------------------}

Procedure Listado\_JugadoresActivos(RaizJugadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if (RD.Nick <> 'ADMINISTRADOR') then

begin

if (RD.Estado) then

begin

Grid.RowCount:= Grid.RowCount + 1;

Grid.Cells[0, Grid.RowCount-1] := RD.Nombre;

Grid.Cells[1, Grid.RowCount-1] := RD.Apellido;

Grid.Cells[2, Grid.RowCount-1] := RD.Nick;

if (RD.TipoJugador) then

Grid.Cells[3, Grid.RowCount-1] :='Real'

else

Grid.Cells[3, Grid.RowCount-1] :='Ficticio';

if (RD.Bloqueado) then

Grid.Cells[4, Grid.RowCount-1] :='BLOQUEADO'

else

Grid.Cells[4, Grid.RowCount-1] :='DESBLOQUEADO';

if (RD.Estado) then

Grid.Cells[5, Grid.RowCount-1] :='SI'

else

Grid.Cells[5, Grid.RowCount-1] :='NO';

Grid.Cells[6, Grid.RowCount-1] := datetimetostr(RD.Alta);

Grid.Cells[7, Grid.RowCount-1] := datetimetostr(RD.UltimaConexion);

grid.Cells[8, grid.RowCount-1]:= inttostr(ME\_CTACTE.SaldoAcumulado(me\_cuentacorriente,RD.Nick));

end;

end;

Listado\_JugadoresActivos(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), grid);

Listado\_JugadoresActivos(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), grid);

end;

end;

{--------------------------------------------------------------------------------}

Procedure ListadoArbolGanadores(RaizGanadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ListadoArbolGanadores(ME\_GANADORES.ProximoIzq(ME\_GANADOR,RaizGanadores), grid);

ListadoArbolGanadores(ME\_GANADORES.ProximoDer(ME\_GANADOR,RaizGanadores), grid);

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

Grid.RowCount:= Grid.RowCount + 1;

grid.Cells[0, grid.RowCount-1] := intTostr(ME\_GANADORES.ObtenerNivel\_enArbolTri(ME\_GANADOR, RegNodoGanador.clave));

grid.Cells[1, grid.RowCount-1] := RegNodoGanador.clave;

end;

end;

{--------------------------------------------------------------------------------}

Procedure ListadoArbolJugadores(RaizJugadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ListadoArbolJugadores(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), grid);

ListadoArbolJugadores(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), grid);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

grid.RowCount:= grid.RowCount + 1;

grid.Cells[0, grid.RowCount-1] := intTostr(ME\_JUGADORES.ObtenerNivel\_enArbol(ME\_JUGADOR, RD.nick));

grid.Cells[1, grid.RowCount-1] := RD.nick;

end;

end;

{--------------------------------------------------------------------------------}

procedure ListadoPartidas (var grid: TStringGrid);

var

regPartida: Type\_JUEGO.TipoRegDatos;

posPartida: tpos;

i: tpos;

regnulo: Type\_JUEGO.TipoRegDatos;

cantReales: tCantidad;

cantFicticios: tCantidad;

montoTotalApuestas: tImporte;

montoTotalPremios: tImporte;

begin

ME\_JUEGO.RegistroNulo(regnulo);

ME\_JUEGO.Insertar(ME\_JUEGORULETA, regnulo);

ME\_JUEGO.Frente(ME\_JUEGORULETA,regPartida);

ME\_JUEGO.Decolar(ME\_JUEGORULETA);

while regpartida.NroJugada<>ME\_JUEGO.ClaveNula(ME\_JUEGORULETA) do

begin

montoTotalPremios:= 0;

cantReales:= 0;

cantFicticios:= 0;

grid.RowCount:= grid.RowCount + 1;

MontoEnApuestaXPartida(regpartida.NroJugada, montoTotalApuestas);

MontoPremiosXPartida(ME\_GANADORES.Raiz(ME\_GANADOR), regpartida.NroJugada , montoTotalPremios);

Cant\_RealesYFicticios\_xPartida(ME\_JUGADORES.Raiz(ME\_JUGADOR), regpartida.NroJugada, cantReales, cantFicticios);

grid.Cells[0, grid.RowCount-1]:= regpartida.nrojugada;

grid.Cells[1, grid.RowCount-1]:= datetimetostr(regPartida.FechaHora);;

grid.Cells[2, grid.RowCount-1]:= inttostr(cantReales);

grid.Cells[3, grid.RowCount-1]:= inttostr(cantFicticios);

grid.Cells[4, grid.RowCount-1]:= inttostr(montoTotalApuestas);

grid.Cells[5, grid.RowCount-1]:= inttostr(montoTotalPremios);

grid.Cells[6, grid.RowCount-1]:= inttostr(regPartida.Bolilla);

ME\_JUEGO.Insertar(ME\_JUEGORULETA, regPartida);

ME\_JUEGO.Frente(ME\_JUEGORULETA,regPartida);

ME\_JUEGO.Decolar(ME\_JUEGORULETA);

end;

end;

{--------------------------------------------------------------------------------}

Procedure ListadoJugadoresNuncaGanaron(RaizJugadores: Lib\_Auxiliar.tPos; var grid: TStringGrid);

var

RD:Type\_JUGADOR.tRegDatos;

esGanador: boolean;

begin

esGanador:= false;

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if (RD.Nick<>'ADMINISTRADOR') then

begin

JugadorEsUnGanandor(ME\_GANADORES.Raiz(ME\_GANADOR), RD.Nick, esGanador);

if not(esGanador) then

begin

grid.RowCount:= grid.RowCount + 1;

grid.Cells[0, grid.RowCount-1] := RD.Nombre;

grid.Cells[1, grid.RowCount-1] := RD.Apellido;

grid.Cells[2, grid.RowCount-1] := RD.Nick;

if RD.TipoJugador then

grid.Cells[3, grid.RowCount-1] :='Real'

else

grid.Cells[3, grid.RowCount-1] :='Ficticio';

if (RD.Bloqueado) then

grid.Cells[4, grid.RowCount-1] :='Bloqueado'

else

grid.Cells[4, grid.RowCount-1] :='Desbloqueado';

if (RD.Estado) then

grid.Cells[5, grid.RowCount-1] :='SI'

else

grid.Cells[5, grid.RowCount-1] :='NO';

grid.Cells[6, grid.RowCount-1] := datetimetostr(RD.Alta);

grid.Cells[7, grid.RowCount-1] := datetimetostr(RD.UltimaConexion);

grid.Cells[8, grid.RowCount-1]:= inttostr(ME\_CTACTE.SaldoAcumulado(me\_cuentacorriente,RD.Nick));

end;

end;

ListadoJugadoresNuncaGanaron(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), grid);

ListadoJugadoresNuncaGanaron(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), grid);

end;

end;

{--------------------------------------------------------------------------------}

//Procedimiento recursivo donde muestra la cuenta corriente de un jugador de forma ordenada por fecha y hora

procedure ListadoCtaCte\_deUnJugador(posListaParcial: tPos; Nick: tclave; var grid: TStringGrid);

var

i: tpos;

regAlm: Type\_ALMACEN.tipoRegAlmacen;

begin

i:= posListaParcial;

If i<>ME\_CTACTE.PosNula(ME\_CUENTACORRIENTE) then

begin

ListadoCtaCte\_deUnJugador(ME\_CTACTE.ProximoEnLdParcial(ME\_CUENTACORRIENTE,i), Nick, grid);

ME\_CTACTE.CapturarEnAlmacen(ME\_CUENTACORRIENTE,i,regAlm);

if (regAlm.Nick = Nick) then

begin

grid.RowCount:= grid.RowCount + 1;

grid.Cells[0, grid.RowCount-1]:= regAlm.Nick;

grid.Cells[1, grid.RowCount-1]:= datetimetostr(regAlm.FechaHora);

grid.Cells[2, grid.RowCount-1]:= regAlm.Concepto;

grid.Cells[3, grid.RowCount-1]:= inttostr(regAlm.Debe);

grid.Cells[4, grid.RowCount-1]:= inttostr(regAlm.haber);

grid.Cells[5, grid.RowCount-1]:= inttostr(regAlm.saldo);

end;

end;

end;

{--------------------------------------------------------------------------------}

procedure ReordenarGridPorFecha (var grid: TStringGrid; cantFilas: integer);

var

i, j, cant: integer;

saldo1, saldo2, debe1, debe2, haber1, haber2: integer;

nick1, nick2: Lib\_Auxiliar.tClave;

concepto1, concepto2: string;

FechaHora1, FechaHora2: tDateTime;

begin

showmessage(intTostr(cantFilas));

for I := 1 to cantFilas - 1 do

begin

for j := 1 to cantFilas - 1 do

begin

Saldo1:= strtoint(grid.Cells[5,i]); Saldo2:= strtoint(grid.Cells[5,j]);

Haber1:= strtoint(grid.Cells[4,i]); Haber2:= strtoint(grid.Cells[4,j]);

Debe1:= strtoint(grid.Cells[3,i]); Debe2:= strtoint(grid.Cells[3,j]);

Concepto1:= (grid.Cells[2,i]); Concepto2:=(grid.Cells[2,j]);

FechaHora1:= strTodatetime(grid.Cells[1,i]); FechaHora2:= strTodatetime(grid.Cells[1,j]);

Nick1:= grid.Cells [0,i]; Nick2:= grid.Cells [0,j];

//Intercambio celdas

grid.Cells[5,i]:= intTostr(Saldo2);

grid.Cells[5,j]:= intTostr(Saldo1);

grid.Cells[4,i]:= intTostr(Haber2);

grid.Cells[4,j]:= intTostr(Haber1);

grid.Cells[3,i]:= intTostr(Debe2);

grid.Cells[3,j]:= intTostr(Debe1);

grid.Cells[2,i]:= Concepto2;

grid.Cells[2,j]:= Concepto1;

grid.Cells[1,i]:= datetimetostr(FechaHora2);

grid.Cells[1,j]:= datetimetostr(FechaHora1);

grid.Cells[0,i]:= nick2;

grid.Cells[0,j]:= nick1;

end;

end;

grid.RowCount:= cantFilas + 1;

end;

{--------------------------------------------------------------------------------}

Procedure ListadoPremiados\_Nomenclador(RaizGanadores: Lib\_Auxiliar.tPos; nomenclador: tNomenclador; var grid: TStringGrid);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

i:= ME\_GANADORES.PrimerJugada\_Ganador(ME\_GANADOR, RegNodoGanador);

while i<> ME\_GANADORES.PosNula\_Ganadores(ME\_GANADOR) do

begin

ME\_GANADORES.CapturarJugada\_Ganador(ME\_GANADOR, i, RegAlmacen);

if (RegAlmacen.Nomenclador = nomenclador) then

begin

Grid.RowCount:= Grid.RowCount + 1;

Grid.Cells[0, Grid.RowCount-1] := RegNodoGanador.clave;

Grid.Cells[1, Grid.RowCount-1] := intTostr(RegAlmacen.NroJugada);

Grid.Cells[2, Grid.RowCount-1] := RegAlmacen.Nomenclador;

Grid.Cells[3, Grid.RowCount-1] := intTostr(RegAlmacen.Numero);

Grid.Cells[4, Grid.RowCount-1] := intTostr(RegAlmacen.Importe);

end;

i:= ME\_GANADORES.SiguienteJugada\_Ganador(ME\_GANADOR, i);

end; //while

ListadoPremiados\_Nomenclador(ME\_GANADORES.ProximoIzq(ME\_GANADOR, RaizGanadores), nomenclador, grid);

ListadoPremiados\_Nomenclador(ME\_GANADORES.ProximoDer(ME\_GANADOR, RaizGanadores), nomenclador, grid);

end;

end;

{--------------------------------------------------------------------------------}

Procedure ListadoMasPremiados(RaizGanadores: Lib\_Auxiliar.tPos; var importe:tImporte; var grid: TStringGrid; var cantFilas: integer);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

i:= ME\_GANADORES.PrimerJugada\_Ganador(ME\_GANADOR, RegNodoGanador);

while i<> ME\_GANADORES.PosNula\_Ganadores(ME\_GANADOR) do

begin

ME\_GANADORES.CapturarJugada\_Ganador(ME\_GANADOR, i, RegAlmacen);

importe:= importe + RegAlmacen.Importe;

i:= ME\_GANADORES.SiguienteJugada\_Ganador(ME\_GANADOR, i);

end; //while

cantFilas:= cantFilas + 1;

Grid.RowCount:= Grid.RowCount + 1;

Grid.Cells[0, Grid.RowCount-1] := RegNodoGanador.clave;

Grid.Cells[1, Grid.RowCount-1] := intTostr(importe);

ListadoMasPremiados(ME\_GANADORES.ProximoIzq(ME\_GANADOR, RaizGanadores), importe, grid, cantFilas);

ListadoMasPremiados(ME\_GANADORES.ProximoDer(ME\_GANADOR, RaizGanadores), importe, grid, cantFilas);

end;

end;

{--------------------------------------------------------------------------------}

Procedure Listado\_GeneralApuestas(var grid: TStringGrid; Nro\_Jugada: Type\_JUEGO.tClave);

var

i:tPos;

begin

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while i<>ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA,i,regApuesta);

if (regApuesta.NroJugada = Nro\_Jugada) then

begin

grid.RowCount:= grid.RowCount + 1;

{Agrego renglon}

grid.Cells[0, grid.RowCount-1] := regApuesta.Nick;

grid.Cells[1, grid.RowCount-1] := regApuesta.Nomenclador;

grid.Cells[2, grid.RowCount-1] := regApuesta.Valor;

grid.Cells[3, grid.RowCount-1] := intTostr(regApuesta.Importe);

end;

i:= ME\_APUESTAS.Proximo(ME\_APUESTA,i);

END; //while

end;

{--------------------------------------------------------------------------------}

Procedure Listado\_ApuestasUnJugador(var grid: TStringGrid; Nro\_Jugada: Type\_JUEGO.tClave; nick: Type\_JUGADOR.TClave);

var

i:tPos;

begin

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while i<>ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA,i,regApuesta);

if (regApuesta.NroJugada = Nro\_Jugada) then

begin

if (regApuesta.Nick = nick) then

begin

grid.RowCount:= grid.RowCount + 1;

{Agrego renglon}

grid.Cells[0, grid.RowCount-1] := regApuesta.Nick;

grid.Cells[1, grid.RowCount-1] := regApuesta.Nomenclador;

grid.Cells[2, grid.RowCount-1] := regApuesta.Valor;

grid.Cells[3, grid.RowCount-1] := intTostr(regApuesta.Importe);

end;

end;

i:= ME\_APUESTAS.Proximo(ME\_APUESTA,i);

END; //while

end;

{--------------------------------------------------------------------------------}

Procedure Listado\_Premios(RaizGanadores: Lib\_Auxiliar.tPos; jugada: Type\_JUEGO.TClave; var grid: TStringGrid);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

i:= ME\_GANADORES.PrimerJugada\_Ganador(ME\_GANADOR, RegNodoGanador);

while i<> ME\_GANADORES.PosNula\_Ganadores(ME\_GANADOR) do

begin

ME\_GANADORES.CapturarJugada\_Ganador(ME\_GANADOR, i, RegAlmacen);

if (intTostr(RegAlmacen.NroJugada) = jugada) then

begin

grid.RowCount:= grid.RowCount + 1;

grid.Cells[0, grid.RowCount-1] := RegNodoGanador.clave;

grid.Cells[1, grid.RowCount-1] := RegAlmacen.Nomenclador;

grid.Cells[2, grid.RowCount-1] := intTostr(RegAlmacen.Numero);

grid.Cells[3, grid.RowCount-1] := intTostr(RegAlmacen.Importe);

end;

i:= ME\_GANADORES.SiguienteJugada\_Ganador(ME\_GANADOR, i);

end; //while

Listado\_Premios(ME\_GANADORES.ProximoIzq(ME\_GANADOR, RaizGanadores), jugada, grid);

Listado\_Premios(ME\_GANADORES.ProximoDer(ME\_GANADOR, RaizGanadores), jugada, grid);

end;

end;

{--------------------------------------------------------------------------------}

procedure Ini\_Sesion (var Me: ME\_JUGADORES.ME\_JUGADOR; var regJugador: Type\_JUGADOR.tRegDatos);

var

pos:tPos;

begin

regJugador.UltimaConexion:= NOW();

regJugador.Estado:= true;

ME\_JUGADORES.BuscarInfoME\_Jugadores(Me, pos, regJugador.Nick);

ME\_JUGADORES.ModificarInfoME\_Jugadores(Me, regJugador,pos);

end;

{--------------------------------------------------------------------------------}

procedure Fin\_Sesion (var Me: ME\_JUGADORES.ME\_JUGADOR; var regJugador: Type\_JUGADOR.tRegDatos);

var

pos:tPos;

begin

regJugador.UltimaConexion:= NOW();

regJugador.Estado:= false;

ME\_JUGADORES.BuscarInfoME\_Jugadores(Me, pos, regJugador.Nick);

ME\_JUGADORES.ModificarInfoME\_Jugadores(Me, regJugador,pos);

end;

{--------------------------------------------------------------------------------}

procedure recuperarRegPartida (Nro\_Jugada: Type\_JUEGO.tClave; var regBuscado: Type\_JUEGO.TipoRegDatos);

var

regnulo, regPartida: Type\_JUEGO.TipoRegDatos;

corte: boolean;

begin

ME\_JUEGO.RegistroNulo(regnulo);

ME\_JUEGO.Insertar(ME\_JUEGORULETA, regnulo);

ME\_JUEGO.Frente(ME\_JUEGORULETA,regPartida);

ME\_JUEGO.Decolar(ME\_JUEGORULETA);

//Ciclo hasta que se encuentra con el regBandera

while regPartida.NroJugada<>ME\_JUEGO.ClaveNula(ME\_JUEGORULETA) do

begin

//Si encontre el regustro de la jugada que busco, lo guardo en el registro parametrizado por referencia

if Nro\_Jugada = regPartida.NroJugada then

regBuscado:= regPartida;

ME\_JUEGO.Insertar(ME\_JUEGORULETA, regPartida);

ME\_JUEGO.Frente(ME\_JUEGORULETA,regPartida);

ME\_JUEGO.Decolar(ME\_JUEGORULETA);

end;

end;

{--------------------------------------------------------------------------------}

Procedure CantidadJugadoresNuncaGanaron(RaizJugadores: Lib\_Auxiliar.tPos; var cant: tCantidad);

var

RD:Type\_JUGADOR.tRegDatos;

esGanador: boolean;

begin

esGanador:= false;

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if (RD.Nick<>'ADMINISTRADOR') then

begin

JugadorEsUnGanandor(ME\_GANADORES.Raiz(ME\_GANADOR), RD.Nick, esGanador);

if not(esGanador) then

cant:= cant + 1;

end;

CantidadJugadoresNuncaGanaron(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores), cant);

CantidadJugadoresNuncaGanaron(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores), cant);

end;

end;

{--------------------------------------------------------------------------------}

Procedure InsertoPremioUnGanador(regGanador: Type\_GANADOR.TipoRegDatos; var regApuesta: Type\_APUESTA.TipoRegDatos; tipoGanador: tNomenclador; Importe: tImporte);

begin

if (tipoGanador <> type\_APUESTA.Nom\_Pleno) then

regApuesta.Valor:= '-1';

regGanador.Nomenclador:= regApuesta.Nomenclador;

regGanador.Numero:= strToint(regApuesta.Valor); //

regGanador.NroJugada:= strToint(regApuesta.NroJugada);

regGanador.Importe:= Importe; //

//Inserto ganador

ME\_GANADORES.InsertarInfoME\_Ganadores(ME\_GANADOR,regGanador, regApuesta.Nick);

end;

{--------------------------------------------------------------------------------}

Procedure MovimientoApuestaUnJugador(nick: tClave; regApuesta: Type\_APUESTA.TipoRegDatos; Importe: tImporte);

begin

//Completo registro movimiento en CtaCte

regCtaCte.Nick:= nick;

regCtaCte.FechaHora:= Now();

regCtaCte.Concepto:= Type\_ALMACEN.\_PremioJuego + regApuesta.NroJugada; //

regCtaCte.Debe:= Importe;

regCtaCte.haber:= 0;

regCtaCte.saldo:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, regApuesta.Nick) + Importe;

//Inserto movimiento en CtaCte

ME\_CTACTE.InsertarAlmacen(ME\_CUENTACORRIENTE, regCtaCte);

end;

{--------------------------------------------------------------------------------}

Procedure Premios\_AcumuladosXJugador(RaizGanadores: Lib\_Auxiliar.tPos; Nick:Type\_JUGADOR.tClave; var cantPremios: tCantidad; var Monto: tImporte);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

if (RegNodoGanador.clave = Nick) then

begin

i:= ME\_GANADORES.PrimerJugada\_Ganador(ME\_GANADOR, RegNodoGanador);

while i<> ME\_GANADORES.PosNula\_Ganadores(ME\_GANADOR) do

begin

ME\_GANADORES.CapturarJugada\_Ganador(ME\_GANADOR, i, RegAlmacen);

Monto:= Monto + RegAlmacen.Importe;

cantPremios:= cantPremios + 1;

i:= ME\_GANADORES.SiguienteJugada\_Ganador(ME\_GANADOR, i);

end; //while

end;

Premios\_AcumuladosXJugador(ME\_GANADORES.ProximoIzq(ME\_GANADOR, RaizGanadores), Nick, cantPremios, Monto);

Premios\_AcumuladosXJugador(ME\_GANADORES.ProximoDer(ME\_GANADOR, RaizGanadores), Nick, cantPremios, Monto);

end;

end;

{------------------------------------------------------------------------------}

Procedure Premios\_AcumuladosXJugadorUnaJugada(RaizGanadores: Lib\_Auxiliar.tPos; Nick:Type\_JUGADOR.tClave; jugada: Type\_JUEGO.TClave; var Monto: tImporte);

var

RegNodoGanador: regNodo;

RegAlmacen: type\_GANADOR.TipoRegDatos;

i: tPos;

begin

if (RaizGanadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_GANADORES.CapturarInfoME\_Ganadores(ME\_GANADOR, RegNodoGanador, RaizGanadores);

if (RegNodoGanador.clave = Nick) then

begin

i:= ME\_GANADORES.PrimerJugada\_Ganador(ME\_GANADOR, RegNodoGanador);

while i<> ME\_GANADORES.PosNula\_Ganadores(ME\_GANADOR) do

begin

ME\_GANADORES.CapturarJugada\_Ganador(ME\_GANADOR, i, RegAlmacen);

if (intTostr(RegAlmacen.NroJugada) = jugada) then

Monto:= Monto + RegAlmacen.Importe;

i:= ME\_GANADORES.SiguienteJugada\_Ganador(ME\_GANADOR, i);

end; //while

end;

Premios\_AcumuladosXJugadorUnaJugada(ME\_GANADORES.ProximoIzq(ME\_GANADOR, RaizGanadores), Nick, jugada, Monto);

Premios\_AcumuladosXJugadorUnaJugada(ME\_GANADORES.ProximoDer(ME\_GANADOR, RaizGanadores), Nick, jugada, Monto);

end;

end;

{--------------------------------------------------------------------------------}

Function JugadorTieneApuestasEnPartida (NroJugada: Type\_JUEGO.TClave; nick: Type\_JUGADOR.tClave): boolean;

var

i: tpos;

regApuesta: Type\_APUESTA.TipoRegDatos;

begin

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while i<>ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA,i,regApuesta);

if (regapuesta.NroJugada= nrojugada) then //pregunto si es la misma jugada que le paso

begin

if (regapuesta.Nick= nick) then //pregunto si es el jugador que le paso

JugadorTieneApuestasEnPartida:= true;

end;

i:= ME\_APUESTAS.Proximo(ME\_APUESTA,i);

end;

end;

{--------------------------------------------------------------------------------}

function Mensaje\_Confirmacion(Msje: String): boolean;

begin

if messagedlg(Msje, mtConfirmation, [mbOk, mbCancel], 0) = mrOk then

Mensaje\_Confirmacion := true

else

Mensaje\_Confirmacion := false;

end;

{------------------------------------------------------------------------------}

procedure limpiarGrid(grid: TStringGrid);

var

i:integer;

begin

with grid do

for i := 0 to RowCount - 1 do

Rows[i].Clear;

end;

{------------------------------------------------------------------------------}

Procedure EncabezadoDetalladoJugadores(grid: TStringGrid);

Begin

with grid do

Begin

//cambio el total de columnas

grid.ColCount:= 9;

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[2] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[3] := Canvas.TextWidth('xxxxxxxxxxxx');

ColWidths[4] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxx');

ColWidths[5] := Canvas.TextWidth('xxxxxxxxxx');

ColWidths[6] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxxxxx');

ColWidths[7] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxxxxx');

ColWidths[8] := Canvas.TextWidth('xxxxxxxxxxxxx');

Cells[0, 0] := 'NOMBRE';

Cells[1, 0] := 'APELLIDO';

Cells[2, 0] := 'NICK';

Cells[3, 0] := 'TIPO';

Cells[4, 0] := 'ESTADO';

Cells[5, 0] := 'ACTIVO';

Cells[6, 0] := 'FECHA ALTA';

Cells[7, 0] := 'ULT. CONEXIÓN';

Cells[8, 0] := 'SALDO';

End;

End;

{--------------------------------------------------------------------------------}

Procedure EncabezadoArbolJugadores(grid: TStringGrid);

Begin

with grid do

Begin

//cambio el total de columnas

grid.ColCount:= 2;

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxx');

Cells[0, 0] := 'NIVEL';

Cells[1, 0] := 'NICK';

End;

End;

{--------------------------------------------------------------------------------}

Procedure EncabezadoListadoPartidas(grid: TStringGrid);

Begin

with grid do

Begin

//cambio el total de columnas

grid.ColCount:= 7;

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxxxx');

ColWidths[2] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[3] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxx');

ColWidths[4] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[5] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxxxx');

ColWidths[6] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

Cells[0, 0] := 'NRO.PARTIDA';

Cells[1, 0] := 'FECHA-HORA';

Cells[2, 0] := 'TOTAL J.REALES';

Cells[3, 0] := 'TOTAL J.FICTICIOS';

Cells[4, 0] := '$ APOSTADO';

Cells[5, 0] := '$ PREMIOS PAGADOS';

Cells[6, 0] := 'NRO. BOLILLA';

End;

End;

{--------------------------------------------------------------------------------}

Procedure EncabezadoJugadoresMasGanaron(grid: TStringGrid);

Begin

with grid do

Begin

//cambio el total de columnas

grid.ColCount:= 2;

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxx');

Cells[0, 0] := 'NICK';

Cells[1, 0] := 'SALDO';

End;

End;

{--------------------------------------------------------------------------------}

Procedure EncabezadoGanadoresXPremio(grid: TStringGrid);

Begin

with grid do

Begin

//cambio el total de columnas

grid.ColCount:= 5;

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[2] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[3] := Canvas.TextWidth('xxxxxxxxxxxx');

ColWidths[4] := Canvas.TextWidth('xxxxxxxxxxxxxx');

Cells[0, 0] := 'NICK';

Cells[1, 0] := 'JUGADA';

Cells[2, 0] := 'NOMENCLADOR';

Cells[3, 0] := 'NUMERO';

Cells[4, 0] := 'PREMIO';

End;

End;

{--------------------------------------------------------------------------------}

Procedure EncabezadoFiltroCtaCte(grid: TStringGrid);

Begin

with grid do

Begin

//cambio el total de columnas

grid.ColCount:= 6;

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxx');

ColWidths[2] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxxxxxxxxx');

ColWidths[3] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[4] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[5] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

Cells[0, 0] := 'NICK';

Cells[1, 0] := 'FECHA/HR';

Cells[2, 0] := 'CONCEPTO';

Cells[3, 0] := 'DEBE';

Cells[4, 0] := 'HABER';

Cells[5, 0] := 'SALDO';

End;

End;

{--------------------------------------------------------------------------------}

Procedure EncabezadosComunApuestas(grid: TStringGrid);

begin

with grid do

Begin

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxxxxx');

ColWidths[2] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[3] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

Cells[0, 0] := 'NICK';

Cells[1, 0] := 'NOMENCLADOR';

Cells[2, 0] := 'VALOR';

Cells[3, 0] := 'IMPORTE';

End;

end;

{--------------------------------------------------------------------------------}

Procedure EncabezadosComunJugadores(grid: TStringGrid);

Begin

with grid do

Begin

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[2] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[3] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[4] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[5] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxxxx');

Cells[0, 0] := 'NICK';

Cells[1, 0] := 'NOMBRE J.';

Cells[2, 0] := 'APELLIDO J.';

Cells[3, 0] := 'TIPO';

Cells[4, 0] := 'BLOQUEADO';

Cells[5, 0] := 'ULT. CONEXIÓN';

End;

End;

{------------------------------------------------------------------------------}

Procedure EncabezadoGridPremios(grid: TStringGrid);

begin

with grid do

Begin

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[2] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

ColWidths[3] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxx');

Cells[0, 0] := 'JUGADOR';

Cells[1, 0] := 'NOMENCLADOR';

Cells[2, 0] := 'NUMERO';

Cells[3, 0] := 'PREMIO';

End;

end;

{------------------------------------------------------------------------------}

Procedure EncabezadosComunFicticios(grid: TStringGrid);

Begin

with grid do

Begin

// Título de las columnas

ColWidths[0] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[1] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[2] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[3] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[4] := Canvas.TextWidth('xxxxxxxxxxxxxxxxx');

ColWidths[5] := Canvas.TextWidth('xxxxxxxxxx');

ColWidths[6] := Canvas.TextWidth('xxxxxxxxxxxxxxxxxxxxxxxxxxxx');

Cells[0, 0] := 'NICK';

Cells[1, 0] := 'NOMBRE J.';

Cells[2, 0] := 'APELLIDO J.';

Cells[3, 0] := 'TIPO';

Cells[4, 0] := 'ESTADO';

Cells[5, 0] := 'SALDO';

Cells[6, 0] := 'ALTA';

End;

End;

END.

## Formularios

### Formulario Login

unit Form\_Login;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Lib\_AuxJuego,

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, Buttons, StdCtrls, ExtCtrls, jpeg, Vcl.Menus;

type

TF\_Login = class(TForm)

Panel1: TPanel;

PanelGeneral: TPanel;

Label6: TLabel;

PanelBotones1: TPanel;

BtnLogin: TButton;

BtnAltas: TButton;

PanelFormulario: TPanel;

Lab\_apellido: TLabel;

Lab\_nombre: TLabel;

Lab\_contrasenia: TLabel;

Lab\_nick: TLabel;

Txt\_Nombre: TEdit;

Txt\_Contrasenia: TEdit;

Txt\_NICK: TEdit;

BtnOk: TBitBtn;

BtnCancelar: TBitBtn;

Txt\_Apellido: TEdit;

Btn\_HideShowContrasenia: TBitBtn;

Menu\_Login: TMainMenu;

SALIR1: TMenuItem;

Function CamposAltaOk():Boolean;

Function NickValido():Boolean;

Function CamposLoginOk():Boolean;

procedure Limpiar\_Formulario();

procedure show\_apellidoNombre();

procedure Pantalla\_InicioLogin();

procedure hide\_apellidoNombre();

procedure BtnLoginClick(Sender: TObject);

procedure BtnOkClick(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure BtnAltasClick(Sender: TObject);

procedure BtnCancelarClick(Sender: TObject);

procedure BtnEraseClick(Sender: TObject);

procedure Txt\_ContraseniaKeyUp(Sender: TObject; var Key: Word;

Shift: TShiftState);

procedure Txt\_NombreKeyUp(Sender: TObject; var Key: Word;

Shift: TShiftState);

procedure Txt\_NombreKeyPress(Sender: TObject; var Key: Char);

procedure Txt\_ApellidoKeyPress(Sender: TObject; var Key: Char);

procedure FormShow(Sender: TObject);

procedure Btn\_HideShowContraseniaClick(Sender: TObject);

procedure Txt\_NICKKeyPress(Sender: TObject; var Key: Char);

procedure Txt\_ContraseniaKeyPress(Sender: TObject; var Key: Char);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure SALIR1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_Login: TF\_Login;

Opcion: char;

AdminLog : boolean; //si es el administrador quien se loguea es true

NickLogueado : Type\_JUGADOR.tClave;

ME\_JUGADOR: ME\_JUGADORES.ME\_Jugador;

regJugador: Type\_JUGADOR.tRegDatos;

posjugador: Lib\_Auxiliar.tpos;

ME\_CUENTACORRIENTE: ME\_CTACTE.ME\_CUENTA;

RegCtaCte: Type\_ALMACEN.tipoRegAlmacen;

posAlmacen: Lib\_Auxiliar.tpos;

saldoAcumulado: tImporte;

ME\_JUEGORULETA: ME\_JUEGO.ME\_RULETA;

RegJuegoRuleta: Type\_JUEGO.TipoRegDatos;

ME\_APUESTA: ME\_APUESTAS.ME\_APUESTA;

RegApuesta: Type\_APUESTA.TipoRegDatos;

ME\_GANADOR: ME\_GANADORES.ME\_GANADOR;

RegGanador: Type\_GANADOR.TipoRegDatos;

implementation

uses

Form\_Jugadores, Form\_Croupier, Form\_Juego;

{$R \*.dfm}

{------------------------------------------------------------------------------}

procedure TF\_Login.Btn\_HideShowContraseniaClick(Sender: TObject);

begin

if (Txt\_Contrasenia.PasswordChar = #0) then

begin

Txt\_Contrasenia.PasswordChar:='\*';

Btn\_HideShowContrasenia.Glyph.LoadFromFile(Lib\_Auxiliar.rutaIconoOjoAbierto);

end

else

begin

Txt\_Contrasenia.PasswordChar:= #0;

Btn\_HideShowContrasenia.Glyph.LoadFromFile(Lib\_Auxiliar.rutaIconoOjoCerrado);

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.BtnAltasClick(Sender: TObject);

begin

Opcion:='B';

BtnLogin.Enabled:=false;

panelformulario.Enabled:= true;

show\_ApellidoNombre;

Txt\_NICK.SetFocus;

Limpiar\_Formulario;

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.BtnCancelarClick(Sender: TObject);

begin

Btn\_HideShowContrasenia.hide;

hide\_apellidoNombre;

BtnLogin.Enabled:=true;

BtnAltas.Enabled:=true;

Limpiar\_Formulario;

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.BtnLoginClick(Sender: TObject);

begin

Opcion:='A';

BtnAltas.Enabled:=false;

panelformulario.Enabled:= true;

Txt\_NICK.Enabled:= true;

Txt\_Contrasenia.Enabled:= true;

Txt\_Nombre.Enabled:= true;

Txt\_Apellido.Enabled:= true;

Txt\_NICK.SetFocus;

Limpiar\_Formulario;

hide\_apellidoNombre;

Txt\_NICK.Text:='ADMINISTRADOR';

Txt\_Contrasenia.Text:='MONDONGO';

PanelFormulario.Enabled:= true;

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.BtnOkClick(Sender: TObject);

begin

Btn\_HideShowContrasenia.hide;

NickLogueado:= Txt\_NICK.Text;

case Opcion of

'A': begin //LOGIN

if (CamposLoginOk) then

begin

if not (ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,PosJugador,NickLogueado)) then

begin //Nick incorrecto

MessageDlg('Nick incorrecto',mtError , [mbOK], 0);

Txt\_NICK.Clear;

Txt\_CONTRASENIA.Clear;

Txt\_NICK.SetFocus;

end

else

begin

//capturo registro para comparar nick con su contraseña

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,regJugador,PosJugador);

if not (Txt\_CONTRASENIA.Text = regJugador.Contrasenia) then

begin //contraseña incorrecta

MessageDlg('La contraseña no coincide con el Nick ingresado',mtError , [mbOK], 0);

Txt\_CONTRASENIA.Clear;

Txt\_CONTRASENIA.SetFocus;

end

else //Usuario logueado correctamente

begin

Form\_Login.F\_Login.hide;

//Consulto si el jugador ya esta en el juego. Si esta activo, esta en el juego

if not (regJugador.Estado) then

begin //jugador no esta en el juego. Pregunto si es el crupier

if (regJugador.Nick = 'ADMINISTRADOR') then

begin //ADMINISTRADOR LOGUEADO...

AdminLog:=true;

Limpiar\_Formulario;

Form\_Croupier.F\_Croupier.Show;

end

else //USUARIO COMUN LOGUEADO...

begin

AdminLog:=false;

Form\_Jugadores.F\_Jugadores.Show;

//busco saldo acumulado del jugador logueado

saldoAcumulado:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, regJugador.Nick);

end;

Lib\_AuxJuego.Ini\_Sesion(ME\_JUGADOR, regJugador); //Inicio sesion, ya sea administrador o usuario común

end

else //jugador ya esta en el juego

begin

MessageDlg('El usuario ya esta en el juego',mtInformation , [mbOK], 0);

Limpiar\_Formulario;

Txt\_NICK.SetFocus;

end;

end; //fin usuario logueado correctamente

end; //fin existe nick

end //fin campos completados correctamente

else

MessageDlg('Debe completar todos los campos.',mtError , [mbOK], 0);

end;

'B': begin //ALTA

if (CamposAltaOk) then

begin

if (NickValido) then

begin

if (ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,PosJugador, NickLogueado)) then

begin //Nick en uso

MessageDlg('Nick ya esta en uso. Intente con otro',mtConfirmation , [mbOK], 0);

Txt\_NICK.Clear;

Txt\_NICK.SetFocus;

end

else

begin //Inserto juegador

regJugador.Nick:= NickLogueado;

regJugador.Contrasenia:= Txt\_Contrasenia.Text;

regJugador.Apellido:= Txt\_Apellido.Text;

regJugador.Nombre:= Txt\_Nombre.Text;

regJugador.Alta:= now();

regJugador.UltimaConexion:= now();

regJugador.Bloqueado:= false;

regJugador.Estado:= false;

regJugador.TipoJugador:= true;

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR, PosJugador, regJugador.Nick);

ME\_JUGADORES.InsertarInfoME\_Jugadores(ME\_JUGADOR,regJugador,PosJugador);

//Cargo datos de ctacte

RegCtaCte.Nick:=regJugador.Nick;

RegCtaCte.FechaHora:= Now();

RegCtaCte.Concepto:= Type\_ALMACEN.tipoConceptos.Creacion\_Cuenta;

RegCtaCte.Debe:= ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA)\*15; //50\*15 = 750

RegCtaCte.haber:= 0;

RegCtaCte.saldo:= RegCtaCte.Debe;//al crearse la cuenta se carga 15 veces la apuesta minima y se refleja en el saldo

//inserto cuenta corriente

ME\_CTACTE.InsertarAlmacen(ME\_CUENTACORRIENTE,RegCtaCte);

MessageDlg('El usuario ha sido registrado!',mtConfirmation , [mbOK], 0);

Limpiar\_Formulario;

hide\_apellidoNombre;

PanelFormulario.Enabled:=false;

BtnLogin.Enabled:=true;

BtnAltas.Enabled:=true;

BtnLogin.SetFocus;

end;

end//NickValido

else

begin

MessageDlg('NICK INVÁLIDO!' + #13 + #13 +'Comienzo del Nick con "' + Lib\_Auxiliar.Fict\_IniNick + '"' + #13 + 'está reservado solo para los ficticios!',mtWarning , [mbOK], 0);

Txt\_Nick.Clear;

Txt\_Nick.SetFocus;

end;

end //CamposAltaOk

else

MessageDlg('Debe completar todos los campos.',mtError , [mbOK], 0);

end;//fin opcion B

end;//case

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.BtnEraseClick(Sender: TObject);

begin

ME\_JUGADORES.CerrarME\_Jugadores(ME\_JUGADOR);

ME\_JUGADORES.DestruirME\_Jugadores(ME\_JUGADOR);

ME\_CTACTE.CerrarMe\_CtaCte(ME\_CUENTACORRIENTE);

ME\_CTACTE.DestruirMe\_CtaCte(ME\_CUENTACORRIENTE);

ME\_JUEGO.CerrarME(ME\_JUEGORULETA);

ME\_JUEGO.DestruirME(ME\_JUEGORULETA);

ME\_APUESTAS.CerrarME\_Apuestas(ME\_APUESTA);

ME\_APUESTAS.DestruirME\_Apuestas(ME\_APUESTA);

ME\_GANADORES.CerrarME\_Ganadores(ME\_GANADOR);

ME\_GANADORES.DestruirME\_Ganadores(ME\_GANADOR);

end;

{------------------------------------------------------------------------------}

Function TF\_Login.CamposAltaOk():Boolean;

begin

CamposAltaOk:= (Txt\_Nombre.Text<>'')and(Txt\_Contrasenia.Text<>'')and (Txt\_NICK.Text<>'') and

(Txt\_Apellido.Text<>'');

end;

{------------------------------------------------------------------------------}

Function TF\_Login.NickValido():Boolean;

var

comienzoNick: string[2];

begin

comienzoNick:= copy(Txt\_NICK.Text, 1, 2);

NickValido:= (comienzoNick <> Lib\_Auxiliar.Fict\_IniNick);

end;

{------------------------------------------------------------------------------}

Function TF\_Login.CamposLoginOk():Boolean;

begin

CamposLoginOk:= (Txt\_NICK.Text<>'') and (Txt\_Contrasenia.Text<>'');

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.FormClose(Sender: TObject; var Action: TCloseAction);

begin

ME\_JUGADORES.CerrarME\_Jugadores(ME\_JUGADOR);

ME\_CTACTE.CerrarMe\_CtaCte(ME\_CUENTACORRIENTE);

ME\_JUEGO.CerrarME(ME\_JUEGORULETA);

ME\_APUESTAS.CerrarME\_Apuestas(ME\_APUESTA);

ME\_GANADORES.CerrarME\_Ganadores(ME\_GANADOR);

end;

procedure TF\_Login.FormCreate(Sender: TObject);

var

regAuxJuego: TYPE\_JUEGO.TipoRegDatos;

begin

ME\_JUGADORES.CrearME\_Jugadores(ME\_JUGADOR);

ME\_JUGADORES.AbrirME\_Jugadores(ME\_JUGADOR);

ME\_CTACTE.CrearMe\_CtaCte(ME\_CUENTACORRIENTE);

ME\_CTACTE.AbrirMe\_CtaCte(ME\_CUENTACORRIENTE);

ME\_JUEGO.CrearME(ME\_JUEGORULETA);

ME\_JUEGO.AbrirME(ME\_JUEGORULETA);

ME\_APUESTAS.CrearME(ME\_APUESTA);

ME\_APUESTAS.AbrirME\_Apuestas(ME\_APUESTA);

ME\_GANADORES.CrearME\_Ganadores(ME\_GANADOR);

ME\_GANADORES.AbrirME\_Ganadores(ME\_GANADOR);

if ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) then

begin

regJugador.Nick:='ADMINISTRADOR';

regJugador.Contrasenia:= 'MONDONGO';

regJugador.Nombre:='CRUPIER';

regJugador.Apellido:='CRUPIER';

regJugador.Alta:=NOW();

regJugador.UltimaConexion:=NOW();

regJugador.Estado:=false;

regJugador.TipoJugador:=true;

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,regJugador.Nick);

ME\_JUGADORES.InsertarInfoME\_Jugadores(ME\_JUGADOR,regJugador,PosJugador);

end;

//operaciones generales

AdminLog:=false;

hide\_apellidoNombre;

self.Position := poScreenCenter;

self.WindowState := wsMaximized;

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.FormShow(Sender: TObject);

begin

Pantalla\_InicioLogin();

ME\_JUGADORES.AbrirME\_Jugadores(ME\_JUGADOR);

ME\_CTACTE.AbrirMe\_CtaCte(ME\_CUENTACORRIENTE);

ME\_JUEGO.AbrirME(ME\_JUEGORULETA);

ME\_APUESTAS.AbrirME\_Apuestas(ME\_APUESTA);

ME\_GANADORES.AbrirME\_Ganadores(ME\_GANADOR);

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.Limpiar\_Formulario();

begin

Txt\_NICK.Clear;

Txt\_Contrasenia.Clear;

Txt\_Nombre.Clear;

Txt\_Apellido.Clear;

end;

{------------------------------------------------------------------------------}

procedure TF\_Login.show\_apellidoNombre();

begin

Lab\_nombre.Visible:=true;

Lab\_apellido.Visible:=true;

Txt\_Nombre.Visible:=true;

Txt\_Apellido.Visible:=true;

Lab\_nombre.Show;

Lab\_apellido.Show;

Txt\_Nombre.Show;

Txt\_Apellido.Show;

end;

{-----------------------------------------------------------------------}

procedure TF\_Login.Txt\_ApellidoKeyPress(Sender: TObject; var Key: Char);

const

CARAC\_HABILITADOS = ['a'..'z', 'A'..'Z', #0..#27];//solo letras y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

{-----------------------------------------------------------------------}

procedure TF\_Login.Txt\_ContraseniaKeyPress(Sender: TObject; var Key: Char);

const

CARAC\_HABILITADOS = ['a'..'z', 'A'..'Z', '0'..'9', #0..#27];//solo letras, numero y borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

procedure TF\_Login.Txt\_ContraseniaKeyUp(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

if (length(Txt\_NICK.Text)<6) then

begin

MessageDlg('Nick debe tener al menos 6 caracteres',mtConfirmation , [mbOK], 0);

Txt\_CONTRASENIA.Clear;

Txt\_NICK.Clear;

Txt\_NICK.SetFocus;

end

else

Btn\_HideShowContrasenia.show;

end;

{-----------------------------------------------------------------------}

procedure TF\_Login.Txt\_NICKKeyPress(Sender: TObject; var Key: Char);

const

CARAC\_HABILITADOS = ['a'..'z', 'A'..'Z', '0'..'9', #95, #46, #0..#27];//solo letras, numeros, signo punto, signo guion bajo y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

{-----------------------------------------------------------------------}

procedure TF\_Login.Txt\_NombreKeyPress(Sender: TObject; var Key: Char);

const

CARAC\_HABILITADOS = ['a'..'z', 'A'..'Z', #0..#27];//solo letras y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

{-----------------------------------------------------------------------}

Procedure TF\_Login.Txt\_NombreKeyUp(Sender: TObject; var Key: Word;

Shift: TShiftState);

begin

if (length(Txt\_Contrasenia.Text)<4) then

begin

MessageDlg('La contraseña debe tener 4 caracteres como minimo',mtConfirmation , [mbOK], 0);

Txt\_Contrasenia.Clear;

Txt\_Contrasenia.SetFocus;

end;

end;

{-----------------------------------------------------------------------}

procedure TF\_Login.hide\_apellidoNombre();

begin

Lab\_nombre.Visible:=false;

Lab\_apellido.Visible:=false;

Txt\_Nombre.Visible:=false;

Txt\_Apellido.Visible:=false;

Lab\_nombre.Hide;

Lab\_apellido.Hide;

Txt\_Nombre.Hide;

Txt\_Apellido.Hide;

end;

{-----------------------------------------------------------------------}

procedure TF\_Login.Pantalla\_InicioLogin();

begin

Form\_Juego.F\_Juego.Hide;

Btn\_HideShowContrasenia.Hide;

BtnLogin.Enabled:= true;

BtnAltas.Enabled:= true;

PanelFormulario.Enabled:= false;

Txt\_NICK.Clear;

Txt\_Contrasenia.Clear;

Txt\_Nombre.Clear;

Txt\_Apellido.Clear;

end;

{-----------------------------------------------------------------------}

procedure TF\_Login.SALIR1Click(Sender: TObject);

begin

F\_Login.Close;

end;

END.

### Formulario Croupier

unit Form\_Croupier;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Lib\_AuxJuego,

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Menus, Vcl.StdCtrls, Vcl.ExtCtrls,

Vcl.Buttons, Vcl.Imaging.jpeg;

type

TF\_Croupier = class(TForm)

MainMenu1: TMainMenu;

Juego1: TMenuItem;

Listados1: TMenuItem;

Bloquear1: TMenuItem;

Activos1: TMenuItem;

Jugarpartida1: TMenuItem;

Listadograljugadores1: TMenuItem;

Ganadoresdeterminadopremio1: TMenuItem;

Jugadoresmasdineroganaron1: TMenuItem;

Jugadoresquenuncaganaron1: TMenuItem;

Listagraldepartidas1: TMenuItem;

HerramientasdelME1: TMenuItem;

Mostrararboljugadores1: TMenuItem;

Mostrararbolganadores1: TMenuItem;

Cerrarsesion1: TMenuItem;

Crearficticios1: TMenuItem;

Panel\_CroupierGral: TPanel;

Panel\_CrearFicticios: TPanel;

Label1: TLabel;

Btn\_crearFicticios: TButton;

Bit\_closeCrearFicticios: TBitBtn;

Edit\_cantFicticios: TEdit;

Label2: TLabel;

Fichaunjugador1: TMenuItem;

Panel\_EncabezadoCroupier: TPanel;

Panel2: TPanel;

Label3: TLabel;

Panel15: TPanel;

Label4: TLabel;

Lbl\_nickCroupier: TLabel;

Panel16: TPanel;

Label5: TLabel;

Lbl\_fechaCroupier: TLabel;

Label6: TLabel;

Lbl\_horaCroupier: TLabel;

Img\_fondoCroupier: TImage;

procedure FormCreate(Sender: TObject);

procedure Cerrarsesion1Click(Sender: TObject);

procedure Jugarpartida1Click(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure Crearficticios1Click(Sender: TObject);

procedure Edit\_cantFicticiosKeyPress(Sender: TObject; var Key: Char);

procedure Btn\_crearFicticiosClick(Sender: TObject);

procedure Bloquear1Click(Sender: TObject);

procedure Listadograljugadores1Click(Sender: TObject);

procedure Ganadoresdeterminadopremio1Click(Sender: TObject);

procedure Jugadoresmasdineroganaron1Click(Sender: TObject);

procedure Jugadoresquenuncaganaron1Click(Sender: TObject);

procedure CtaCtedeunjugador1Click(Sender: TObject);

procedure Listagraldepartidas1Click(Sender: TObject);

procedure Mostrararboljugadores1Click(Sender: TObject);

procedure Mostrararbolganadores1Click(Sender: TObject);

procedure Activos1Click(Sender: TObject);

procedure Bit\_closeCrearFicticiosClick(Sender: TObject);

procedure Fichaunjugador1Click(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_Croupier: TF\_Croupier;

implementation

uses

Form\_Login, Form\_Juego, Form\_PanelCaseButtons, Form\_Listados;

{$R \*.dfm}

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Activos1Click(Sender: TObject);

var

Cant: tCantidad; // cantidad jugadores en general

Cant\_Activos: tCantidad; // cantidad de jugadores activos

begin

Panel\_CrearFicticios.Hide;

Form\_Listados.opListado:= 9;

cant:= 0;

Cant\_Activos:= 0;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

Lib\_AuxJuego.Cant\_JugadoresActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR), Cant\_Activos);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

//Si hay jugadores activos. Podria usar una funcion booleana. Pero la cantidad de activos la voy a necesitar para mostrar

if (Cant\_Activos>=1) then

F\_Listados.Show

else

MessageDlg('No hay jugadores activos!',mtWarning, [mbOK], 0);

end

else

MessageDlg('No existen jugadores!!',mtWarning, [mbOK], 0);

end;

{------------------------------------------------------------------------------}

procedure TF\_Croupier.Bit\_closeCrearFicticiosClick(Sender: TObject);

begin

Panel\_CrearFicticios.Hide;

Edit\_cantFicticios.Clear;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Bloquear1Click(Sender: TObject);

var

Cant: tCantidad; // cantidad jugadores en general

begin

Panel\_CrearFicticios.Hide;

Cant:= 0;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not(ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR)) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

F\_PanelCaseButtons.MostrarEnGridJugadores();

end

else

begin

MessageDlg('No existen jugadores!!',mtWarning, [mbOK], 0);

Form\_Croupier.F\_Croupier.Panel\_CroupierGral.Enabled:= true;

end;

F\_PanelCaseButtons.Panel\_LockUnlock.Enabled:= true;

F\_PanelCaseButtons.Panel\_LockUnlock.show;

F\_PanelCaseButtons.Panel\_RegalarCredito.Enabled:= false;

F\_PanelCaseButtons.Panel\_RegalarCredito.Hide;

F\_PanelCaseButtons.Lbl\_regalarCred.Hide;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Btn\_crearFicticiosClick(Sender: TObject);

begin

if(Edit\_cantFicticios.Text <> '')then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿Esta seguro que quiere crear '+ Edit\_cantFicticios.Text + ' ficticios?') then

begin

Lib\_AuxJuego.Insertar\_JugadoresFicticios(ME\_JUGADOR, strToint(Edit\_cantFicticios.Text));

Edit\_cantFicticios.Clear;

Panel\_CrearFicticios.Hide;

MessageDlg(Edit\_cantFicticios.Text + ' jugadores ficticios han sido creado con éxito.',mtConfirmation , [mbOK], 0);

end

else

begin

Edit\_cantFicticios.SetFocus;

Edit\_cantFicticios.Clear;

end;

end

else

begin

MessageDlg('Debe ingresar cantidad a crear.',mtWarning , [mbOK], 0);

Edit\_cantFicticios.SetFocus;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Cerrarsesion1Click(Sender: TObject);

begin

if (Lib\_Auxiliar.Mensaje\_Confirmacion('¿Esta seguro que quiere cerrar sesión?')) then

begin

{CerrarSesion}

Lib\_AuxJuego.Fin\_Sesion(ME\_JUGADOR, regJugador);

F\_Croupier.Hide;

Form\_Login.F\_Login.show;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Crearficticios1Click(Sender: TObject);

begin

Panel\_CrearFicticios.Show;

Edit\_cantFicticios.SetFocus;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.CtaCtedeunjugador1Click(Sender: TObject);

var

cant:tCantidad;

begin

cant:=0;

//- Cuenta corriente de un Jugador

Form\_Listados.opListado:= 5;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

Form\_Listados.F\_Listados.Show

else

MessageDlg('No existen jugadores!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Edit\_cantFicticiosKeyPress(Sender: TObject;

var Key: Char);

const

CARAC\_HABILITADOS = ['0'..'9', #0..#27];//solo letras y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Fichaunjugador1Click(Sender: TObject);

var

cant:tCantidad;

begin

Panel\_CrearFicticios.Hide;

cant:=0;

//- Ficha de un Jugador

Form\_Listados.opListado:= 5;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

Form\_Listados.F\_Listados.Show

else

MessageDlg('No existen jugadores!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.FormCreate(Sender: TObject);

begin

self.Position := poScreenCenter;

self.WindowState := wsMaximized;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.FormShow(Sender: TObject);

begin

Panel\_CrearFicticios.Hide;

Lbl\_nickCroupier.Caption:= NickLogueado;

F\_Juego.ActualizarFechaHora(Lbl\_fechaCroupier, Lbl\_horaCroupier); //Actualizo fecha y hora a mostrar

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Ganadoresdeterminadopremio1Click(Sender: TObject);

var

cantJ:tCantidad;

begin

Panel\_CrearFicticios.Hide;

cantJ:=0;

//- Listado ganadores determinado premio

Form\_Listados.opListado:= 2;

cantJ:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cantJ>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

if not(ME\_JUEGO.MEVacio(ME\_JUEGORULETA)) then

begin

if not ME\_GANADORES.MeVacio\_Ganadores(ME\_GANADOR) then

Form\_Listados.F\_Listados.Show

else

MessageDlg('No existen ganadores!!',mtWarning , [mbOK], 0);

end

else

MessageDlg('No se ha efectuado ninguna jugada aún!!',mtWarning , [mbOK], 0);

end

else

MessageDlg('No existen jugadores!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Jugadoresmasdineroganaron1Click(Sender: TObject);

var

cantJ: tCantidad; //cantidad jugadores

begin

Panel\_CrearFicticios.Hide;

cantJ:=0;

//- Listado de los jugadores que mas dinero ganaron

Form\_Listados.opListado:= 3;

cantJ:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cantJ>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

if not(ME\_JUEGO.MEVacio(ME\_JUEGORULETA)) then

begin

if not ME\_GANADORES.MeVacio\_Ganadores(ME\_GANADOR) then

Form\_Listados.F\_Listados.Show

else

MessageDlg('No existen ganadores!!',mtWarning , [mbOK], 0);

end

else

MessageDlg('No se ha efectuado ninguna jugada aún!!',mtWarning , [mbOK], 0);

end

else

MessageDlg('No existen jugadores!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Jugadoresquenuncaganaron1Click(Sender: TObject);

var

cantJ:tCantidad;//cantidad jugadores

cantJNG: tCantidad; //cantidad jugadores que nunca ganaron

begin

Panel\_CrearFicticios.Hide;

cantJ:= 0;

cantJNG:= 0;

//- Listado de los jugadores que nunca ganaron apuestas.

Form\_Listados.opListado:= 4;

cantJ:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

Lib\_AuxJuego.CantidadJugadoresNuncaGanaron(ME\_JUGADORES.Raiz(ME\_JUGADOR), cantJNG);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cantJ>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

if not(ME\_JUEGO.MEVacio(ME\_JUEGORULETA)) then

begin

if (cantJNG>0) then

Form\_Listados.F\_Listados.Show

else

MessageDlg('No existen jugadores que no hayan perdido alguna apuesta!!',mtWarning , [mbOK], 0);

end

else

MessageDlg('No se ha efectuado ninguna jugada aún!!',mtWarning , [mbOK], 0);

end

else

MessageDlg('No existen jugadores!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Jugarpartida1Click(Sender: TObject);

begin

//Msj de advertencia para que no se intente salir del juego si el mismo no esta en estado 4 (fin de juego y repartido los premios)

MessageDlg('TENGA PRESENTE QUE: ' + #13 + #13 + ' Una vez seleccionado "CREAR PARTIDA NUEVA" solo podrá salir del juego cuando el mismo finalice',mtInformation , [mbOK], 0);

F\_Croupier.Hide;

F\_Juego.Show;

F\_Juego.pantallaIniJuegoCrupier();

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Listadograljugadores1Click(Sender: TObject);

var

cant:tCantidad;

begin

Panel\_CrearFicticios.Hide;

cant:=0;

//Listado General de Usuarios (jugadores). Debe incluir nombre, Nick, foto, clave, estado, etc. de cada uno (línea por línea).

Form\_Listados.opListado:= 1;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

Form\_Listados.F\_Listados.Show

else

MessageDlg('No existen jugadores!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Listagraldepartidas1Click(Sender: TObject);

var

cant:tCantidad;

begin

Panel\_CrearFicticios.Hide;

cant:=0;

//- Listado general de las partidas

Form\_Listados.opListado:= 6;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

if not(ME\_JUEGO.MEVacio(ME\_JUEGORULETA)) then

Form\_Listados.F\_Listados.Show

else

MessageDlg('No se ha efectuado ninguna jugada aún!!',mtWarning , [mbOK], 0);

end

else

MessageDlg('No existen jugadores!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Mostrararbolganadores1Click(Sender: TObject);

begin

//Voy a mostrar arbol de ganadores indicando el nick y nivel del mismo

Panel\_CrearFicticios.Hide;

Form\_Listados.opListado:= 8;

if (not ME\_GANADORES.MeVacio\_Ganadores(ME\_GANADOR)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

Form\_Listados.F\_Listados.Show

else

MessageDlg('No existen ganadores!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_Croupier.Mostrararboljugadores1Click(Sender: TObject);

var

cant:tCantidad;

begin

Panel\_CrearFicticios.Hide;

cant:=0;

//Voy a mostrar de jugadores indicando el nick y nivel del mismo

Form\_Listados.opListado:= 7;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

Form\_Listados.F\_Listados.Show

else

MessageDlg('No existen jugadores!!',mtWarning , [mbOK], 0);

end;

END.

### Formulario Jugadores

unit Form\_Jugadores;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Lib\_AuxJuego,

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Grids, Vcl.Buttons,

Vcl.ExtCtrls, Vcl.Menus, Vcl.Imaging.jpeg;

type

TF\_Jugadores = class(TForm)

Menu\_Jugadores: TMainMenu;

Entraraljuego1: TMenuItem;

Comprarcredito1: TMenuItem;

Jugadoresactivos1: TMenuItem;

Modificardatos1: TMenuItem;

Cambiarcontrasea1: TMenuItem;

SALIR1: TMenuItem;

Panel1: TPanel;

Panel\_BajaModificacion: TPanel;

caption\_panelGral: TLabel;

Panel\_Botones1: TPanel;

BtnModificacion: TButton;

BtnBajas: TButton;

Panel\_Formulario: TPanel;

Lab\_apellido: TLabel;

Lab\_nombre: TLabel;

Lab\_nick: TLabel;

Txt\_Nombre: TEdit;

Txt\_NICK: TEdit;

BtnOk: TBitBtn;

BtnCancelar: TBitBtn;

Txt\_Apellido: TEdit;

Btn\_closePnelBM: TBitBtn;

panel\_CambioContrasenia: TPanel;

btnCambiarContraseña: TBitBtn;

Bit\_ShowHide1: TBitBtn;

Bit\_ShowHide2: TBitBtn;

Btn\_closePnelAlterContrasenia: TBitBtn;

panel\_Compracredito: TPanel;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

btnRecargar: TBitBtn;

Txt\_Saldo: TEdit;

Btn\_closePnelCredito: TBitBtn;

lbEdit\_Recarga: TEdit;

Label4: TLabel;

lbEdit\_Contraseña: TEdit;

Label5: TLabel;

lbEdit\_ConfirmContraseñaConf: TEdit;

Label6: TLabel;

Panel\_EncabezadoUser: TPanel;

Panel2: TPanel;

Label7: TLabel;

Panel15: TPanel;

Label8: TLabel;

Lbl\_nickUser: TLabel;

Panel16: TPanel;

Label9: TLabel;

Lbl\_fechaUser: TLabel;

Label10: TLabel;

Lbl\_horaUser: TLabel;

Img\_fondoJugadores: TImage;

procedure FormCreate(Sender: TObject);

procedure Pantalla\_InicioJugadores();

procedure FormShow(Sender: TObject);

procedure SALIR1Click(Sender: TObject);

procedure Jugadoresactivos1Click(Sender: TObject);

procedure Entraraljuego1Click(Sender: TObject);

procedure BtnModificacionClick(Sender: TObject);

procedure BtnOkClick(Sender: TObject);

procedure BtnBajasClick(Sender: TObject);

procedure Modificardatos1Click(Sender: TObject);

procedure BtnCancelarClick(Sender: TObject);

procedure Txt\_NombreKeyPress(Sender: TObject; var Key: Char);

procedure Txt\_ApellidoKeyPress(Sender: TObject; var Key: Char);

procedure Cambiarcontrasea1Click(Sender: TObject);

procedure lbEdit\_ContraseñaKeyPress(Sender: TObject; var Key: Char);

procedure lbEdit\_ConfirmContraseñaConfKeyPress(Sender: TObject;

var Key: Char);

procedure Bit\_ShowHide1Click(Sender: TObject);

procedure Bit\_ShowHide2Click(Sender: TObject);

procedure btnCambiarContraseñaClick(Sender: TObject);

procedure Comprarcredito1Click(Sender: TObject);

procedure Btn\_closePnelBMClick(Sender: TObject);

procedure Btn\_closePnelAlterContraseniaClick(Sender: TObject);

procedure Btn\_closePnelCreditoClick(Sender: TObject);

procedure btnRecargarClick(Sender: TObject);

procedure lbEdit\_RecargaKeyPress(Sender: TObject; var Key: Char);

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_Jugadores: TF\_Jugadores;

implementation

uses

Form\_login, Form\_juego, Form\_Listados;

VAR

Opcion: char;

{$R \*.dfm}

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.FormCreate(Sender: TObject);

begin

self.Position := poScreenCenter;

self.WindowState := wsMaximized;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.FormShow(Sender: TObject);

begin

F\_Jugadores.Pantalla\_InicioJugadores();

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Entraraljuego1Click(Sender: TObject);

begin

ME\_JUEGO.RegistroNulo(regJuegoRuleta);

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuegoRuleta);

//busco y capturo jugador para verificar si esta bloqueado y a su vez para activarlo si ingresa al juego

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR, posJugador, RegJugador.Nick);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

//Solo podra entrar a la partida si el estado es igual 0 (Juego creado)

if (regJuegoRuleta.Estado = 0) then

begin

//Podra entrar al juego sino esta bloquado

if not(regJugador.Bloqueado) then

begin

F\_Jugadores.Hide;

//---------------

regJugador.Nick:= regJugador.Nick;

regJugador.Contrasenia:= regJugador.Contrasenia;

regJugador.Nombre:= regJugador.Nombre;

regJugador.Apellido:= regJugador.Apellido;

regJugador.Alta:= regJugador.Alta;

regJugador.UltimaConexion:= Now();

regJugador.Bloqueado:= regJugador.Bloqueado;

regJugador.Estado:= true; //se conecta al juego, lo activo

ME\_JUGADORES.ModificarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

//---------------

F\_Juego.Show;

F\_Juego.refrezcarPantallaNormalUser();

end

else

MessageDlg('USTED ESTA BLOQUEADO: No puede ingresar a ningun juego hasta que sea desbloqueado!',mtWarning , [mbOK], 0);

end

else

MessageDlg('No puede ingresar aún. El juego no ha sido creado o está en curso!',mtWarning , [mbOK], 0);

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Jugadoresactivos1Click(Sender: TObject);

var

Cant: tCantidad; // cantidad jugadores en general

Cant\_Activos: tCantidad; // cantidad de jugadores activos

begin

Form\_Listados.opListado:= 9;

cant:= 0;

Cant\_Activos:= 0;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

Lib\_AuxJuego.Cant\_JugadoresActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR), Cant\_Activos);

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

//Si hay jugadores activos

if (Cant\_Activos>=1) then

F\_Listados.Show

else

MessageDlg('No hay jugadores activos!',mtWarning, [mbOK], 0);

end

else

MessageDlg('No existen jugadores!!',mtWarning, [mbOK], 0);

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.lbEdit\_ConfirmContraseñaConfKeyPress(Sender: TObject;

var Key: Char);

const

CARAC\_HABILITADOS = ['a'..'z', 'A'..'Z', '0'..'9', #0..#27];//solo letras, numero y borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.lbEdit\_RecargaKeyPress(Sender: TObject; var Key: Char);

const

CARAC\_HABILITADOS = ['0'..'9', #0..#27];//solo numero y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Modificardatos1Click(Sender: TObject);

var

i: integer;

begin

//Cuando usuario va a modificar sus datos, como lo que se muestra es un panel y no un formulario, debo dehabilitar el menu principal

for i:=0 to menu\_Jugadores.Items.Count-1 do

menu\_Jugadores.Items[I].Enabled:= false;

panel\_Compracredito.Hide;

panel\_CambioContrasenia.Hide;

Panel\_BajaModificacion.Show;

Panel\_Formulario.Enabled:= false;

BtnModificacion.Enabled:= true;

BtnBajas.Enabled:= true;

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,PosJugador,F\_Jugadores.Txt\_NICK.Text);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,regJugador,PosJugador);

F\_Jugadores.Txt\_Nombre.Text:= regJugador.Nombre;

F\_Jugadores.Txt\_Apellido.Text:= regJugador.Apellido;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Bit\_ShowHide1Click(Sender: TObject);

begin

if (lbEdit\_Contraseña.PasswordChar = #0) then

begin

lbEdit\_Contraseña.PasswordChar:='\*';

Bit\_ShowHide1.Glyph.LoadFromFile(Lib\_Auxiliar.rutaIconoOjoAbierto);

end

else

begin

lbEdit\_Contraseña.PasswordChar:= #0;

Bit\_ShowHide1.Glyph.LoadFromFile(Lib\_Auxiliar.rutaIconoOjoCerrado);

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Bit\_ShowHide2Click(Sender: TObject);

begin

if (lbEdit\_ConfirmContraseñaConf.PasswordChar = #0) then

begin

lbEdit\_ConfirmContraseñaConf.PasswordChar:='\*';

Bit\_ShowHide2.Glyph.LoadFromFile(Lib\_Auxiliar.rutaIconoOjoAbierto);

end

else

begin

lbEdit\_ConfirmContraseñaConf.PasswordChar:= #0;

Bit\_ShowHide2.Glyph.LoadFromFile(Lib\_Auxiliar.rutaIconoOjoCerrado);

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.BtnBajasClick(Sender: TObject);

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('MENSAJE DE CONFIRMACIÓN: ' + #13 + #13 + '¿Seguro que quiere darse de baja?') then

begin

//Cargo datos de ctacte

RegCtaCte.Nick:= F\_Jugadores.Txt\_Nick.Text;

RegCtaCte.FechaHora:= Now();

RegCtaCte.Concepto:= Type\_ALMACEN.tipoConceptos.Baja\_cuenta;

RegCtaCte.Debe:= 0;

RegCtaCte.haber:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, RegCtaCte.Nick);

RegCtaCte.saldo:= 0;

//inserto cuenta corriente

ME\_CTACTE.InsertarAlmacen(ME\_CUENTACORRIENTE,RegCtaCte);

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador, F\_Jugadores.Txt\_NICK.Text);

ME\_JUGADORES.EliminarInfoME\_Jugadores(ME\_JUGADOR, posJugador);

Showmessage('HA SIDO DADA DE BAJA: ' + #13 + #13 + 'Será redirigido a la página principal');

F\_Jugadores.Close;

F\_login.Show;

end

else

Pantalla\_InicioJugadores();

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.btnCambiarContraseñaClick(Sender: TObject);

begin

//usuario va a cambiar su contrasenia

if ((lbEdit\_Contraseña.Text <> '') and (lbEdit\_ConfirmContraseñaConf.Text <> '')) then

begin

if (lbEdit\_Contraseña.Text = lbEdit\_ConfirmContraseñaConf.Text) then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('MENSAJE DE CONFIRMACIÓN: ' + #13 + #13 + '¿Seguro que cambiar su contraseña?') then

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador, F\_Jugadores.Txt\_NICK.Text);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

regJugador.Contrasenia:= lbEdit\_Contraseña.Text;

ME\_JUGADORES.ModificarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

Pantalla\_InicioJugadores();

messagedlg('MODICICACIÓN CON ÉXITO: ' + #13 + #13 + 'Su contraseña ha sido modificada con éxito!', mtInformation, [mbOk], 0);

end;

end

else

begin

MessageDlg('Las contraseñas deben ser iguales!',mtError , [mbOK], 0);

lbEdit\_Contraseña.Clear;

lbEdit\_ConfirmContraseñaConf.Clear;

lbEdit\_Contraseña.SetFocus;

end;

end

else

begin

MessageDlg('Debe completar los dos campos!!',mtError , [mbOK], 0);

if (lbEdit\_Contraseña.Text = '') then

lbEdit\_Contraseña.SetFocus

else

lbEdit\_ConfirmContraseñaConf.SetFocus

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.BtnCancelarClick(Sender: TObject);

begin

BtnModificacion.Enabled:=true;

BtnBajas.Enabled:=true;

Panel\_Formulario.Enabled:= false;

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,PosJugador,F\_Jugadores.Txt\_NICK.Text);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,regJugador,PosJugador);

F\_Jugadores.Txt\_Nombre.Text:= regJugador.Nombre;

F\_Jugadores.Txt\_Apellido.Text:= regJugador.Apellido;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.BtnModificacionClick(Sender: TObject);

begin

Panel\_Formulario.Enabled:= true;

BtnBajas.Enabled:= false;

F\_Jugadores.Txt\_Nombre.SetFocus;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.BtnOkClick(Sender: TObject);

begin

//usuario comun modifica sus datos

if ((F\_Jugadores.Txt\_Nombre.Text <> '') and (F\_Jugadores.Txt\_Apellido.Text <> '')) then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('MENSAJE DE CONFIRMACIÓN: ' + #13 + #13 + '¿Seguro que quiere modificar sus datos?') then

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador, F\_Jugadores.Txt\_NICK.Text);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

regJugador.Nick:= regJugador.Nick;

regJugador.Nombre:= F\_Jugadores.Txt\_Nombre.Text;

regJugador.Apellido:= F\_Jugadores.Txt\_Apellido.Text;

regJugador.TipoJugador:= regJugador.TipoJugador;

regJugador.Estado:= regJugador.Estado;

ME\_JUGADORES.ModificarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

Pantalla\_InicioJugadores();

messagedlg('MODICICACIÓN CON ÉXITO: ' + #13 + #13 + 'Sus datos han sido modificados con éxito!', mtInformation, [mbOk], 0);

end

else

Pantalla\_InicioJugadores();

end

else

begin

MessageDlg('Debe completar los dos campos!!',mtError , [mbOK], 0);

if (F\_Jugadores.Txt\_Nombre.Text = '') then

F\_Jugadores.Txt\_Nombre.SetFocus

else

F\_Jugadores.Txt\_Apellido.SetFocus

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.btnRecargarClick(Sender: TObject);

begin

if (lbEdit\_Recarga.Text<>'') then

begin

if (strtoint(lbEdit\_Recarga.Text) >= ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA)) and (strtoint(lbEdit\_Recarga.Text) <= ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA)) then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿Está seguro que quiere comprar $'+ lbEdit\_Recarga.Text +' en créditos?') then

begin

regCtaCte.Nick:= RegJugador.Nick;

regCtaCte.FechaHora:= Now();

regCtaCte.Concepto:= Type\_ALMACEN.tipoConceptos.Compra\_Fichas;

regCtaCte.Debe:= strToint(lbEdit\_Recarga.Text);

regCtaCte.haber:= 0;

regCtaCte.saldo:= regCtaCte.Debe + ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, RegJugador.Nick);

ME\_CTACTE.InsertarAlmacen(ME\_CUENTACORRIENTE, RegCtaCte);

lbEdit\_Recarga.Clear;

Pantalla\_InicioJugadores();

messagedlg('Compra relizada con éxito!', mtInformation, [mbOk], 0);

end;

lbEdit\_Recarga.Clear;

Pantalla\_InicioJugadores();

end

else

begin

MessageDlg('El crédito minimo debe ser de $ ' +intTostr(ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA))+

', y el crédito máximo debe ser de $ ' +intTostr(ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA)),mtWarning, [mbOK], 0);

lbEdit\_Recarga.Clear;

lbEdit\_Recarga.SetFocus;

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Btn\_closePnelAlterContraseniaClick(Sender: TObject);

var

i: integer;

begin

panel\_CambioContrasenia.Hide;

for i:=0 to menu\_Jugadores.Items.Count-1 do

menu\_Jugadores.Items[I].Enabled:= true;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Btn\_closePnelBMClick(Sender: TObject);

var

i:integer;

begin

Panel\_BajaModificacion.Hide;

for i:=0 to menu\_Jugadores.Items.Count-1 do

menu\_Jugadores.Items[I].Enabled:= true;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Btn\_closePnelCreditoClick(Sender: TObject);

var

i: integer;

begin

panel\_Compracredito.Hide;

lbEdit\_Recarga.Clear;

for i:=0 to menu\_Jugadores.Items.Count-1 do

menu\_Jugadores.Items[I].Enabled:= true;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Cambiarcontrasea1Click(Sender: TObject);

var

i: integer;

begin

panel\_Compracredito.Hide;

Panel\_BajaModificacion.Hide;

panel\_CambioContrasenia.Show;

for i:=0 to menu\_Jugadores.Items.Count-1 do

menu\_Jugadores.Items[I].Enabled:= false;

lbEdit\_Contraseña.Clear;

lbEdit\_ConfirmContraseñaConf.Clear;

lbEdit\_Contraseña.SetFocus;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Comprarcredito1Click(Sender: TObject);

var

i: integer;

begin

panel\_CambioContrasenia.Hide;

Panel\_BajaModificacion.Hide;

panel\_Compracredito.Show;

for i:=0 to menu\_Jugadores.Items.Count-1 do

menu\_Jugadores.Items[I].Enabled:= false;

F\_Jugadores.Txt\_Saldo.Text:=intTostr(ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, RegJugador.Nick));

lbEdit\_Recarga.SetFocus;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Pantalla\_InicioJugadores();

var

i: integer;

begin

Lbl\_nickUser.Caption:= NickLogueado;

F\_Juego.ActualizarFechaHora(Lbl\_fechaUser, Lbl\_horaUser); //Actualizo fecha y hora a mostrar

F\_Jugadores.panel\_Compracredito.Hide;

F\_Jugadores.panel\_CambioContrasenia.Hide;

F\_Jugadores.Panel\_BajaModificacion.Hide;

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,PosJugador,RegJugador.Nick);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,regJugador,PosJugador);

F\_Jugadores.Txt\_NICK.Text:= RegJugador.Nick;

F\_Jugadores.Txt\_Nombre.Text:= regJugador.Nombre;

F\_Jugadores.Txt\_Apellido.Text:= regJugador.Apellido;

for i:=0 to menu\_Jugadores.Items.Count-1 do

menu\_Jugadores.Items[I].Enabled:= true;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.SALIR1Click(Sender: TObject);

begin

if (Lib\_Auxiliar.Mensaje\_Confirmacion('¿Esta seguro que quiere salir?')) then

begin

Lib\_AuxJuego.Fin\_Sesion(ME\_JUGADOR, regJugador);

F\_Jugadores.Close;

F\_Login.Show;

F\_Login.Limpiar\_Formulario();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Jugadores.Txt\_ApellidoKeyPress(Sender: TObject; var Key: Char);

const

CARAC\_HABILITADOS = ['a'..'z', 'A'..'Z', #0..#27];//solo letras y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

procedure TF\_Jugadores.Txt\_NombreKeyPress(Sender: TObject; var Key: Char);

const

CARAC\_HABILITADOS = ['a'..'z', 'A'..'Z', #0..#27];//solo letras y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

procedure TF\_Jugadores.lbEdit\_ContraseñaKeyPress(Sender: TObject;

var Key: Char);

const

CARAC\_HABILITADOS = ['a'..'z', 'A'..'Z', '0'..'9', #0..#27];//solo letras, numero y borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

END.

### Formulario Juego

unit Form\_Juego;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Lib\_AuxJuego,

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls, Vcl.Buttons,

Vcl.Imaging.pngimage, Vcl.ToolWin, Vcl.ComCtrls, Vcl.Imaging.jpeg, Vcl.Menus,

Vcl.Imaging.GIFImg, Vcl.Grids;

type

TF\_Juego = class(TForm)

Splitter1: TSplitter;

Panel\_JuegoGral: TPanel;

Panel\_EncabezadoJuego: TPanel;

Panel2: TPanel;

Label1: TLabel;

Lbl\_nroPartida: TLabel;

Panel15: TPanel;

Label4: TLabel;

Lbl\_usuarioActual: TLabel;

Panel16: TPanel;

Label2: TLabel;

Lbl\_fechaActual: TLabel;

Label3: TLabel;

Lbl\_horaActual: TLabel;

Panel\_Crupier: TPanel;

Panel4: TPanel;

Panel5: TPanel;

Panel6: TPanel;

Panel8: TPanel;

Panel9: TPanel;

Panel10: TPanel;

Panel7: TPanel;

Panel11: TPanel;

Lbl\_bolillaASalir: TLabel;

Lbl\_nroBolillaG: TLabel;

Panel12: TPanel;

Panel\_NameTablero: TPanel;

Label9: TLabel;

menuFormJuego: TMainMenu;

SALIR1: TMenuItem;

Panel\_Jugadores: TPanel;

Panel18: TPanel;

Label11: TLabel;

Lbl\_estadoJuego: TLabel;

Label10: TLabel;

Panel19: TPanel;

Label12: TLabel;

Lbl\_saldoAcumulado: TLabel;

Label14: TLabel;

Panel20: TPanel;

Label13: TLabel;

Label15: TLabel;

Lbl\_ApuestaMax: TLabel;

Panel21: TPanel;

Label16: TLabel;

Label17: TLabel;

Lbl\_ApuestaMin: TLabel;

Panel23: TPanel;

Label19: TLabel;

Spbtn\_haganApuestas: TSpeedButton;

Spbtn\_noVaMas: TSpeedButton;

Spbtn\_tirarBolilla: TSpeedButton;

Spbtn\_obsequiarCred: TSpeedButton;

Spbtn\_activarFicticios: TSpeedButton;

SpeedButton12: TSpeedButton;

Spbtn\_CrearPartida: TSpeedButton;

Lbl\_totalFicticios: TLabel;

Lbl\_cantFicticios: TLabel;

Listarapuestas1: TMenuItem;

Premiados1: TMenuItem;

Spbtn\_repartirPremios: TBitBtn;

Spbtn\_trampa: TSpeedButton;

Panel13: TPanel;

Spbtn\_refresh: TSpeedButton;

Spbtn\_bloqDesbloq: TSpeedButton;

Panel14: TPanel;

Label5: TLabel;

Label6: TLabel;

Lbl\_CantEnLinea: TLabel;

Lbl\_CantFictActivados: TLabel;

panel\_tablero: TPanel;

tabla\_Image0: TImage;

tabla\_Image6: TImage;

tabla\_Image15: TImage;

tabla\_Image21: TImage;

tabla\_Image12: TImage;

tabla\_Image18: TImage;

tabla\_Image9: TImage;

tabla\_Image24: TImage;

tabla\_Image27: TImage;

tabla\_Image30: TImage;

tabla\_Image33: TImage;

tabla\_Image36: TImage;

tabla\_ImageTerceraCol: TImage;

tabla\_Image2: TImage;

tabla\_Image5: TImage;

tabla\_Image8: TImage;

tabla\_Image11: TImage;

tabla\_Image14: TImage;

tabla\_Image17: TImage;

tabla\_Image20: TImage;

tabla\_Image23: TImage;

tabla\_Image26: TImage;

tabla\_Image29: TImage;

tabla\_Image32: TImage;

tabla\_Image35: TImage;

tabla\_Image4: TImage;

tabla\_Image7: TImage;

tabla\_Image10: TImage;

tabla\_Image13: TImage;

tabla\_Image16: TImage;

tabla\_Image19: TImage;

tabla\_Image22: TImage;

tabla\_Image25: TImage;

tabla\_Image28: TImage;

tabla\_Image31: TImage;

tabla\_Image34: TImage;

tabla\_ImageSegundaCol: TImage;

tabla\_ImagePrimeraCol: TImage;

tabla\_ImagePrimeros12: TImage;

tabla\_ImageSegundos12: TImage;

tabla\_ImageTerceros12: TImage;

tabla\_ImageRojo: TImage;

tabla\_ImageNegro: TImage;

tabla\_ImageImpares: TImage;

tabla\_Image\_19a36: TImage;

tabla\_Image3: TImage;

tabla\_Image1: TImage;

tabla\_Image\_1a18: TImage;

tabla\_ImagePares: TImage;

Panel\_Winner: TPanel;

Timer\_showWinner1: TTimer;

Timer\_showWinner2: TTimer;

Timer\_showWinner3: TTimer;

Timer\_showWinner4: TTimer;

Timer\_showLoser1: TTimer;

Timer\_showLoser2: TTimer;

Timer\_showLoser3: TTimer;

Timer\_showLoser4: TTimer;

Timer\_showWinner5: TTimer;

Timer\_showWinner6: TTimer;

Timer\_showLoser5: TTimer;

Timer\_showLoser6: TTimer;

Panel\_ShowWinner: TPanel;

Label7: TLabel;

Label18: TLabel;

Lbl\_montoWinner: TLabel;

Label20: TLabel;

Lbl\_bolillaGWinner: TLabel;

gifWinner: TImage;

Btn\_closePnelWinner: TBitBtn;

Lbl\_waiting: TLabel;

Panel\_Loser: TPanel;

Label8: TLabel;

Lbl\_bolillaGLoser: TLabel;

Label22: TLabel;

Btn\_closePnelLoser: TBitBtn;

procedure FormCreate(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure AutoSize\_ImagenesTablero();

procedure pantallaIniJuegoCrupier();

procedure pantallaIniJuegoJugadores();

procedure ListarPremiados(Nro\_Jugada: Type\_JUEGO.tClave);

procedure ListandoApuestas();

procedure Spbtn\_bloqDesbloqClick(Sender: TObject);

procedure Spbtn\_obsequiarCredClick(Sender: TObject);

procedure SALIR1Click(Sender: TObject);

procedure TableroLegalSeleccionado(nomenclador:Type\_APUESTA.tNomenclador;valor:Type\_APUESTA.tNomenclador; nameOp:string);

procedure TableroTrampaSeleccionado(num\_bolilla:Type\_JUEGO.tBolilla);

procedure refrezcarPantallaNormalUser();

procedure ActualizarFechaHora(var Lbl\_fecha:TLabel; var Lbl\_hora: TLabel);

procedure tabla\_Image3Click(Sender: TObject);

procedure Spbtn\_cancelarSelectTabClick(Sender: TObject);

procedure tabla\_Image1Click(Sender: TObject);

procedure tabla\_Image2Click(Sender: TObject);

procedure tabla\_Image4Click(Sender: TObject);

procedure tabla\_Image5Click(Sender: TObject);

procedure tabla\_Image6Click(Sender: TObject);

procedure tabla\_Image7Click(Sender: TObject);

procedure tabla\_Image8Click(Sender: TObject);

procedure tabla\_Image9Click(Sender: TObject);

procedure tabla\_Image10Click(Sender: TObject);

procedure tabla\_Image11Click(Sender: TObject);

procedure tabla\_Image12Click(Sender: TObject);

procedure tabla\_Image13Click(Sender: TObject);

procedure tabla\_Image14Click(Sender: TObject);

procedure tabla\_Image15Click(Sender: TObject);

procedure tabla\_Image16Click(Sender: TObject);

procedure tabla\_Image17Click(Sender: TObject);

procedure tabla\_Image18Click(Sender: TObject);

procedure tabla\_Image19Click(Sender: TObject);

procedure tabla\_Image20Click(Sender: TObject);

procedure tabla\_Image21Click(Sender: TObject);

procedure tabla\_Image22Click(Sender: TObject);

procedure tabla\_Image23Click(Sender: TObject);

procedure tabla\_Image24Click(Sender: TObject);

procedure tabla\_Image25Click(Sender: TObject);

procedure tabla\_Image26Click(Sender: TObject);

procedure tabla\_Image27Click(Sender: TObject);

procedure tabla\_Image28Click(Sender: TObject);

procedure tabla\_Image29Click(Sender: TObject);

procedure tabla\_Image30Click(Sender: TObject);

procedure tabla\_Image31Click(Sender: TObject);

procedure tabla\_Image32Click(Sender: TObject);

procedure tabla\_Image33Click(Sender: TObject);

procedure tabla\_Image34Click(Sender: TObject);

procedure tabla\_Image35Click(Sender: TObject);

procedure tabla\_Image36Click(Sender: TObject);

procedure tabla\_Image0Click(Sender: TObject);

procedure tabla\_ImageTerceraColClick(Sender: TObject);

procedure tabla\_ImageSegundaColClick(Sender: TObject);

procedure tabla\_ImagePrimeraColClick(Sender: TObject);

procedure tabla\_ImagePrimeros12Click(Sender: TObject);

procedure tabla\_ImageSegundos12Click(Sender: TObject);

procedure tabla\_ImageTerceros12Click(Sender: TObject);

procedure tabla\_Image\_1a18Click(Sender: TObject);

procedure tabla\_ImageParesClick(Sender: TObject);

procedure tabla\_ImageImparesClick(Sender: TObject);

procedure tabla\_Image\_19a36Click(Sender: TObject);

procedure tabla\_ImageRojoClick(Sender: TObject);

procedure tabla\_ImageNegroClick(Sender: TObject);

procedure Spbtn\_activarFicticiosClick(Sender: TObject);

procedure Spbtn\_CrearPartidaClick(Sender: TObject);

procedure SpeedButton12Click(Sender: TObject);

procedure Spbtn\_haganApuestasClick(Sender: TObject);

procedure Spbtn\_noVaMasClick(Sender: TObject);

procedure Spbtn\_tirarBolillaClick(Sender: TObject);

procedure Spbtn\_trampaClick(Sender: TObject);

procedure Spbtn\_repartirPremiosClick(Sender: TObject);

procedure ActiveBtnTrampa();

procedure DesactiveBtnTrampa();

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure Listarapuestas1Click(Sender: TObject);

//procedure SoloMostrarApuestas();

procedure Premiados1Click(Sender: TObject);

procedure SalirDelJuego();

procedure Spbtn\_refreshClick(Sender: TObject);

procedure Timer\_showWinner1Timer(Sender: TObject);

procedure Timer\_showWinner2Timer(Sender: TObject);

procedure Timer\_showWinner3Timer(Sender: TObject);

procedure Timer\_showWinner4Timer(Sender: TObject);

procedure Timer\_showLoser1Timer(Sender: TObject);

procedure Timer\_showLoser2Timer(Sender: TObject);

procedure Timer\_showLoser3Timer(Sender: TObject);

procedure Timer\_showLoser4Timer(Sender: TObject);

procedure Btn\_closePnelWinnerClick(Sender: TObject);

procedure Btn\_closePnelLoserClick(Sender: TObject);

procedure Timer\_showWinner5Timer(Sender: TObject);

procedure Timer\_showWinner6Timer(Sender: TObject);

procedure Timer\_showLoser5Timer(Sender: TObject);

procedure Timer\_showLoser6Timer(Sender: TObject);

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_Juego: TF\_Juego;

Nomenclador: Type\_APUESTA.tNomenclador;

totalFicticios: Lib\_Auxiliar.tCantidad;

apuestaMax, apuestaMin: Type\_JUEGO.tImporte;

btnActiveNow:string;

BtnTrampaIsActive: boolean;

nameOp:string;

valor: string;

EstadoJuego: integer;

SoloListar: boolean;

ShowWinner: boolean;

implementation

uses

Form\_login, Form\_Jugadores, Form\_Croupier, Form\_PanelCaseButtons, Form\_Apostar, Form\_PanelFicticios, Form\_Premios;

{$R \*.dfm}

{--------------------------------------------------------------------------------}

procedure TF\_Juego.FormClose(Sender: TObject; var Action: TCloseAction);

begin

F\_PanelFicticios.Close;

F\_PanelCaseButtons.Close;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.FormCreate(Sender: TObject);

begin

apuestaMax:= ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA);

apuestaMin:= ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA);

Spbtn\_bloqDesbloq.Caption:= '[ BLOQUEAR/' +#13+ 'DESBLOQUEAR ]';

//generalidades de formulario

self.Position := poScreenCenter;

self.WindowState := wsMaximized;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.FormShow(Sender: TObject);

var

i:integer;

begin

Panel\_Winner.Visible:= false;

Panel\_Loser.Visible:= false;

Panel\_ShowWinner.Visible:= false;

//Si es croupier

if (Form\_login.AdminLog) then

pantallaIniJuegoCrupier()

else // si es jugador comun

pantallaIniJuegoJugadores();

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Listarapuestas1Click(Sender: TObject);

var

exist: boolean;

Nro\_Jugada: Type\_JUEGO.tClave;

begin

SoloListar:= true;

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

Nro\_Jugada:= Lbl\_nroPartida.Caption;

if not (ME\_APUESTAS.MEVacio\_Apuestas(ME\_APUESTA)) then

begin

exist:= Lib\_AuxJuego.ExistenApuestas\_enEsteJuego(Nro\_Jugada);

if (exist) then

begin

F\_Apostar.Show;

F\_Apostar.Lbl\_ListaDeApuestas.Hide;

F\_Apostar.Lbl\_SelectJugador.hide;

F\_Apostar.Lbl\_ListaJugadores.Show;

F\_Apostar.Grid\_ListaJugadores.Hide;

F\_Apostar.Panel\_DatosApostar.hide;

F\_Apostar.Grid\_ListaApuestas.Show;

Lib\_Auxiliar.limpiarGrid(F\_Apostar.Grid\_ListaApuestas);

Lib\_Auxiliar.EncabezadosComunApuestas(F\_Apostar.Grid\_ListaApuestas);

F\_Apostar.Grid\_ListaApuestas.RowCount:=1;

Lib\_AuxJuego.Listado\_GeneralApuestas(F\_Apostar.Grid\_ListaApuestas, Nro\_Jugada);

F\_Apostar.Grid\_ListaApuestas.FixedRows:=1;

end

else

MessageDlg('No hay apuestas efectuadas en este juego!!',mtWarning, [mbOK], 0);

end

else

MessageDlg('No hay apuestas efectuadas!!',mtWarning, [mbOK], 0);

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Premiados1Click(Sender: TObject);

var

cantGanadores: tCantidad;

Nro\_Jugada: Type\_JUEGO.tClave;

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

Nro\_Jugada:= Lbl\_nroPartida.Caption;

cantGanadores:= 0;

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuegoRuleta);

if (regJuegoRuleta.NroJugada = Nro\_Jugada) then

begin //Soy el croupier

if (regJuegoRuleta.Estado = 4) then // (4) premios repartidos y fin de juego

begin

if not (ME\_GANADORES.MeVacio\_Ganadores(ME\_GANADOR)) then

begin

Lib\_AuxJuego.CantGanadores\_EnUnaJugada(ME\_GANADORES.Raiz(ME\_GANADOR), Nro\_Jugada, cantGanadores);

if (cantGanadores>0) then

listarPremiados(Nro\_Jugada)

else

MessageDlg('NO HAY GANADORES!' + #13 + #13 +' No hay ganadores en la jugada Nro: ' +RegJuegoRuleta.NroJugada,mtInformation, [mbOK], 0);

end

else

MessageDlg('NO HAY GANADORES!' + #13 + #13 +' No hay ganadores en la jugada Nro: ' +RegJuegoRuleta.NroJugada,mtInformation, [mbOK], 0);

end

else

MessageDlg('EL JUEGO ESTA EN CURSO!' + #13 + #13 +' Una vez finalizado la partida podra listar los ganadores.',mtInformation, [mbOK], 0);

end

else //Soy jugador comun. El juego habia finalizado

begin

if not (ME\_GANADORES.MeVacio\_Ganadores(ME\_GANADOR)) then

begin

Lib\_AuxJuego.CantGanadores\_EnUnaJugada(ME\_GANADORES.Raiz(ME\_GANADOR), Nro\_Jugada, cantGanadores);

if (cantGanadores>0) then

listarPremiados(Nro\_Jugada)

else

MessageDlg('NO HAY GANADORES!' + #13 + #13 +' No hay ganadores en la jugada Nro: ' +Nro\_Jugada,mtInformation, [mbOK], 0);

end

else

MessageDlg('NO HAY GANADORES!' + #13 + #13 +' No hay ganadores en la jugada Nro: ' +Nro\_Jugada,mtInformation, [mbOK], 0);

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.SALIR1Click(Sender: TObject);

var

Nro\_Jugada: Type\_JUEGO.tClave;

begin

Nro\_Jugada:= Lbl\_nroPartida.Caption;

ME\_JUEGO.RegistroNulo(regJuegoRuleta);

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuegoRuleta);

//Si el numero de jugada de la pantalla es distito a la ultima jugada significa que el jugador no esta jugando la partida actual. La partida que juga ya culmino. Puede salir

if (Nro\_Jugada <> regJuegoRuleta.NroJugada) then

RegJuegoRuleta.estado:= 4;

if (RegJuegoRuleta.Estado = 4) then

begin

Form\_Juego.F\_Juego.Close;

if AdminLog then

Form\_Croupier.F\_Croupier.Show

else

Form\_Jugadores.F\_Jugadores.Show;

end

else

MessageDlg('No puede salir del juego hasta que el mismo haya finalizado!',mtWarning , [mbOK], 0)

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.ActiveBtnTrampa();

begin

panel\_tablero.Enabled:= true;

tabla\_ImagePrimeros12.Hide;

tabla\_ImageSegundos12.Hide;

tabla\_ImageTerceros12.Hide;

tabla\_ImageTerceraCol.Hide;

tabla\_ImageSegundaCol.Hide;

tabla\_ImagePrimeraCol.Hide;

tabla\_Image\_1a18.Hide;

tabla\_ImagePares.Hide;

tabla\_ImageRojo.Hide;

tabla\_ImageNegro.Hide;

tabla\_ImageImpares.Hide;

tabla\_Image\_19a36.Hide;

panel\_tablero.Enabled:=true;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.DesactiveBtnTrampa();

begin

panel\_tablero.Enabled:= false;

tabla\_ImagePrimeros12.show;

tabla\_ImageSegundos12.show;

tabla\_ImageTerceros12.show;

tabla\_ImageTerceraCol.show;

tabla\_ImageSegundaCol.show;

tabla\_ImagePrimeraCol.show;

tabla\_Image\_1a18.show;

tabla\_ImagePares.show;

tabla\_ImageRojo.show;

tabla\_ImageNegro.show;

tabla\_ImageImpares.show;

tabla\_Image\_19a36.show;

panel\_tablero.Enabled:=false;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_cancelarSelectTabClick(Sender: TObject);

begin

AutoSize\_ImagenesTablero();

panel\_tablero.Enabled:= true;

F\_Apostar.Edit\_nomenclador.Clear;

F\_Apostar.Edit\_valor.Clear;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.SpeedButton12Click(Sender: TObject);

var

Nro\_Jugada: Type\_JUEGO.tClave;

regBuscado: Type\_JUEGO.TipoRegDatos;

begin

Nro\_Jugada:= Lbl\_nroPartida.Caption;

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR, posJugador, Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

if (regJugador.Bloqueado) then

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

else

begin

//Si el numero de jugada de la pantalla es distito a la ultima jugada significa que el jugador no esta jugando la partida actual

if (Nro\_Jugada <> RegJuegoRuleta.NroJugada) then

begin

//.. me fijo en que estado esta la jugada actual. Si el estado es igual a 0 (patida creada), lo invito a unirse a la nueva partida

if (RegJuegoRuleta.Estado = 0) then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('PARTIDA NRO. ' +RegJuegoRuleta.NroJugada+ ' HA SIDO CREADA:' +#13+#13+'¿Quiere unirse a la nueva partida?') then

begin

pantallaIniJuegoJugadores;

refrezcarPantallaNormalUser();

end;

end

else

MessageDlg('SE ESTA JUGANDO PARTIDA NRO. ' +RegJuegoRuleta.NroJugada+ ':'+#13+#13+'Deberá esperar hasta que una nueva partida comience!',mtInformation , [mbOK], 0)

end

else

refrezcarPantallaNormalUser();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_CrearPartidaClick(Sender: TObject);

var

i:integer;

cantFictActivos, cantEnLinea: tCantidad;

nroJugada: Type\_JUEGO.tClave;

begin

cantFictActivos:= 0;

cantEnLinea:= 0;

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

Lbl\_nroBolillaG.Caption:= '--';

Spbtn\_CrearPartida.Enabled:= false;

Spbtn\_haganApuestas.Enabled:= true;

Spbtn\_activarFicticios.Enabled:= true;

Spbtn\_obsequiarCred.Enabled:= true;

//Muestroa la cantidad de ficticios creados al ingresar al juego buscando el ultimo ficticio

totalFicticios:= ME\_JUGADORES.UltFicticio(ME\_JUGADOR);

Lbl\_cantFicticios.Caption:= intTostr(totalFicticios);

if ME\_JUEGO.MEVacio(ME\_JUEGORULETA) then

begin //El ME\_Juego esta vacio, inserto la primera jugada

RegJuegoRuleta.NroJugada:=inttostr(1);

RegJuegoRuleta.FechaHora:= Now();

RegJuegoRuleta.Estado:= 0;

RegJuegoRuleta.Bolilla:= Lib\_Auxiliar.PosNula;

ME\_JUEGO.Insertar(ME\_JUEGORULETA, RegJuegoRuleta);

Lbl\_cantFicticios.Show;

Lbl\_estadoJuego.Caption:='"Juego Creado"';

end

else

begin //El ME\_Juego no esta vacio. Solo sera posible si el estado es = 4 (premios repartidos y fin de juego)

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

if (RegJuegoRuleta.Estado = 4) then

begin

//creo un nuevo RegJuegoRuleta y lo inserto

RegJuegoRuleta.NroJugada:= intTostr(strToint(RegJuegoRuleta.NroJugada)+1);

RegJuegoRuleta.FechaHora:= Now();

RegJuegoRuleta.Estado:= 0;

RegJuegoRuleta.Bolilla:= Lib\_Auxiliar.PosNula;

ME\_JUEGO.Insertar(ME\_JUEGORULETA, RegJuegoRuleta);

Lbl\_estadoJuego.Caption:='"Juego Creado"';

end;

end;

//Muestro numero de partida luego de haber iniciado la partida

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

Lbl\_nroPartida.Caption:= RegJuegoRuleta.NroJugada;

//Muestro la cantidad de ficticios activas. Al comienzo siempre sera cero, porque el croupier debera activarlos

Lib\_AuxJuego.Cant\_FicticiosActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR), cantFictActivos);

F\_Juego.Lbl\_CantFictActivados.Caption:= intTostr(cantFictActivos);

//Muestro la cantidad de jugadores que se unen a la partida y estaran en linea (activos) una vez creada la partida

Lib\_AuxJuego.Cant\_JugadoresEnLinea(ME\_JUGADORES.Raiz(ME\_JUGADOR), cantEnLinea);

F\_Juego.Lbl\_cantEnLinea.Caption:= intTostr(cantEnLinea); //menos el croupier

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_haganApuestasClick(Sender: TObject);

var

cantActivos: tCantidad;

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

cantActivos:= 0;

Spbtn\_haganApuestas.Enabled:=false;

Spbtn\_noVaMas.Enabled:=true;

//habilito tablero para que lleven a cabo sus apuestas tanto crupier como usuario comun

panel\_tablero.Enabled:= true;

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

if (RegJuegoRuleta.Estado = 0) then

begin

RegJuegoRuleta.NroJugada:= RegJuegoRuleta.NroJugada;

RegJuegoRuleta.FechaHora:= Now();

RegJuegoRuleta.Estado:= 1;

RegJuegoRuleta.Bolilla:= Lib\_Auxiliar.PosNula;

ME\_JUEGO.ModificoUltimo(ME\_JUEGORULETA, RegJuegoRuleta);

Lbl\_estadoJuego.Caption:='"Hagan sus apuestas"';

if (Spbtn\_activarFicticios.Enabled = true) then

Spbtn\_activarFicticios.Enabled:=false;

Spbtn\_obsequiarCred.Enabled:=false;

//Se llevaran a cabo las apuestas de los jugadores ficticios que esten activos

Lib\_AuxJuego.Cant\_FicticiosActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR),cantActivos);

if (cantActivos > 0) then

Lib\_AuxJuego.Apuestas\_JugadoresFicticios(ME\_JUGADORES.Raiz(ME\_JUGADOR), cantActivos);

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_noVaMasClick(Sender: TObject);

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

Spbtn\_noVaMas.Enabled:=false;

Spbtn\_bloqDesbloq.Enabled:= false;

Spbtn\_tirarBolilla.Enabled:=true;

panel\_tablero.Enabled:= false;

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

if (RegJuegoRuleta.Estado = 1) then

begin

RegJuegoRuleta.NroJugada:= RegJuegoRuleta.NroJugada;

RegJuegoRuleta.FechaHora:= Now();

RegJuegoRuleta.Estado:= 2;

RegJuegoRuleta.Bolilla:= Lib\_Auxiliar.PosNula;

ME\_JUEGO.ModificoUltimo(ME\_JUEGORULETA, RegJuegoRuleta);

Lib\_AuxJuego.MovimientoHaberApostadores();

Lbl\_estadoJuego.Caption:='"No va más"';

Spbtn\_obsequiarCred.Enabled:=false;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_bloqDesbloqClick(Sender: TObject);

var

Cant: tCantidad; // cantidad jugadores en general

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

Cant:= 0;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

if (not(ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR)) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

F\_PanelCaseButtons.MostrarEnGridJugadores();

end

else

begin

MessageDlg('No existen jugadores!!',mtWarning, [mbOK], 0);

Form\_Juego.F\_Juego.Panel\_JuegoGral.Enabled:= true;

end;

F\_PanelCaseButtons.Panel\_LockUnlock.Enabled:= true;

F\_PanelCaseButtons.Panel\_LockUnlock.show;

F\_PanelCaseButtons.Panel\_RegalarCredito.Enabled:= false;

F\_PanelCaseButtons.Panel\_RegalarCredito.Hide;

F\_PanelCaseButtons.Lbl\_regalarCred.Hide;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_activarFicticiosClick(Sender: TObject);

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

F\_PanelFicticios.Show;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_obsequiarCredClick(Sender: TObject);

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

F\_PanelCaseButtons.Show;

F\_PanelCaseButtons.Panel\_LockUnlock.hide;

F\_PanelCaseButtons.Panel\_LockUnlock.Enabled:= false;

F\_PanelCaseButtons.Panel\_RegalarCredito.show;

F\_PanelCaseButtons.Panel\_RegalarCredito.Enabled:= true;

F\_PanelCaseButtons.Lbl\_bloquearDesbloquear.hide;

F\_PanelCaseButtons.Lbl\_regalarCred.show;

F\_PanelCaseButtons.Edit\_cantCredito.Enabled:= false;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_refreshClick(Sender: TObject);

var

cantEnLinea: tCantidad;

begin

cantEnLinea:= 0;

//Actualizo fecha y hora a mostrar

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual);

//Al refrezcar vuelvo a mostrar la cantidad de jugadores que se van uniendo a la partida y estaran en linea (activos) una vez creada la partida

Lib\_AuxJuego.Cant\_JugadoresEnLinea(ME\_JUGADORES.Raiz(ME\_JUGADOR), cantEnLinea);

F\_Juego.Lbl\_cantEnLinea.Caption:= intTostr(cantEnLinea); //menos el croupier

end;

procedure TF\_Juego.Spbtn\_repartirPremiosClick(Sender: TObject);

var

cantGanadores: tCantidad;

cantFictActivos: tCantidad;

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

cantFictActivos:= 0;

cantGanadores:= 0;

Spbtn\_trampa.Enabled:=false;

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

if (RegJuegoRuleta.Estado = 3) then

begin

RegJuegoRuleta.NroJugada:= RegJuegoRuleta.NroJugada;

RegJuegoRuleta.FechaHora:= Now();

RegJuegoRuleta.Estado:= 4;

RegJuegoRuleta.Bolilla:= RegJuegoRuleta.Bolilla;

ME\_JUEGO.ModificoUltimo(ME\_JUEGORULETA, RegJuegoRuleta);

//Reparto premio de los apostadores de la ultima jugada

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuegoRuleta);

Lib\_AuxJuego.repartirPremio\_xJugada(ME\_JUGADORES.Raiz(ME\_JUGADOR), regJuegoRuleta.NroJugada);

//Desactivo jugadores ficticios

Lib\_AuxJuego.Desactivar\_JugadoresFicticios(ME\_JUGADORES.Raiz(ME\_JUGADOR));

//Muestro cantidad de ficticios activos una vez culminada la partida. Siempre sera cero

Lib\_AuxJuego.Cant\_FicticiosActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR), cantFictActivos);

F\_Juego.Lbl\_CantFictActivados.Caption:= intTostr(cantFictActivos);

//Muestro la cantidad de ganadores si es que los hay para que luego pueda listarlo en el menu de arriba (Premiados)

if not (ME\_GANADORES.MeVacio\_Ganadores(ME\_GANADOR)) then

begin

Lib\_AuxJuego.CantGanadores\_EnUnaJugada(ME\_GANADORES.Raiz(ME\_GANADOR), RegJuegoRuleta.NroJugada,cantGanadores);

if (cantGanadores>0) then

MessageDlg('HAY ' +intTostr(cantGanadores)+ ' GANADORES EN JUGADA NRO ' +RegJuegoRuleta.NroJugada,mtInformation , [mbOK], 0)

else

MessageDlg('NO HAY GANADORES EN JUGADA NRO ' +RegJuegoRuleta.NroJugada,mtInformation , [mbOK], 0);

end

else

MessageDlg('NO HAY GANADORES EN JUGADA NRO ' +RegJuegoRuleta.NroJugada,mtInformation , [mbOK], 0);

end;

//Invoco al procedimiento que va a mostrar la pantalla de inicio para el croupier una vez culminada una partida

pantallaIniJuegoCrupier();

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_tirarBolillaClick(Sender: TObject);

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

Spbtn\_tirarBolilla.Enabled:=false;

Spbtn\_repartirPremios.Enabled:=true;

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

if (RegJuegoRuleta.Estado = 2) then

begin

RegJuegoRuleta.NroJugada:= RegJuegoRuleta.NroJugada;

RegJuegoRuleta.FechaHora:= Now();

RegJuegoRuleta.Estado:= 3;

RegJuegoRuleta.Bolilla:= (random(36));

ME\_JUEGO.ModificoUltimo(ME\_JUEGORULETA, RegJuegoRuleta);

Lbl\_estadoJuego.Caption:='"Bolilla tirada"';

Lbl\_nroBolillaG.Caption:= intTostr(RegJuegoRuleta.Bolilla);

Spbtn\_tirarBolilla.Enabled:= false;

Spbtn\_trampa.Enabled:=true;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.Spbtn\_trampaClick(Sender: TObject);

begin

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

MessageDlg('SELECCIONE NUMERO DE LA TABLA PARA HACER TRAMPA.',mtInformation , [mbOK], 0);

BtnTrampaIsActive:=true;

ActiveBtnTrampa();

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image0Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image0.Width := tabla\_Image0.Width + 5;

tabla\_Image0.Height := tabla\_Image0.Height + 25;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(0);

nameOp:= intTostr(0);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(0)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image3Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image3.Width := tabla\_Image3.Width + 5;

tabla\_Image3.Height := tabla\_Image3.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(3);

nameOp:= intTostr(3);

//Si es el croupier y se activo el boton trampa, llamo a TableroTrampaSeleccionado parametrizando numero de bolilla a hacer trampa

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(3)

else //es el croupier o jugador comun que estan apostando

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

//si el estado del juego es distinto a 2 (no va mas) esntonces parametrizo nomenclado, valor, y nombre de opcion (nombre de valores que no son numero planos)

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else //el estado del juego es 2 (no va mas) entonces de deshabilita tablero, se refrezca pantalla del usuario comun, y el tamaño del tablero vuelve a su estado principal

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image10Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image10.Width := tabla\_Image10.Width + 5;

tabla\_Image10.Height := tabla\_Image10.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(10);

nameOp:= intTostr(10);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(10)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image11Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image11.Width := tabla\_Image11.Width + 5;

tabla\_Image11.Height := tabla\_Image11.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(11);

nameOp:= intTostr(11);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(11)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image12Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image12.Width := tabla\_Image12.Width + 5;

tabla\_Image12.Height := tabla\_Image12.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(12);

nameOp:= intTostr(12);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(12)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image13Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image13.Width := tabla\_Image13.Width + 5;

tabla\_Image13.Height := tabla\_Image13.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(13);

nameOp:= intTostr(13);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(13)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image14Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image14.Width := tabla\_Image14.Width + 5;

tabla\_Image14.Height := tabla\_Image14.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(14);

nameOp:= intTostr(14);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(14)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image15Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image15.Width := tabla\_Image15.Width + 5;

tabla\_Image15.Height := tabla\_Image15.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(15);

nameOp:= intTostr(15);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(15)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image16Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image16.Width := tabla\_Image16.Width + 5;

tabla\_Image16.Height := tabla\_Image16.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(16);

nameOp:= intTostr(16);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(16)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image17Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image17.Width := tabla\_Image17.Width + 5;

tabla\_Image17.Height := tabla\_Image17.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(17);

nameOp:= intTostr(17);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(17)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image18Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image18.Width := tabla\_Image18.Width + 5;

tabla\_Image18.Height := tabla\_Image18.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(18);

nameOp:= intTostr(18);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(18)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image19Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image19.Width := tabla\_Image19.Width + 5;

tabla\_Image19.Height := tabla\_Image19.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(19);

nameOp:= intTostr(19);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(19)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image1Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image1.Width := tabla\_Image1.Width + 5;

tabla\_Image1.Height := tabla\_Image1.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(1);

nameOp:= intTostr(1);

if (BtnTrampaIsActive) then

TableroTrampaSeleccionado(1)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image20Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image20.Width := tabla\_Image20.Width + 5;

tabla\_Image20.Height := tabla\_Image20.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(20);

nameOp:= intTostr(20);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(20)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image21Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image21.Width := tabla\_Image21.Width + 5;

tabla\_Image21.Height := tabla\_Image21.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(21);

nameOp:= intTostr(21);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(21)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image22Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image22.Width := tabla\_Image22.Width + 5;

tabla\_Image22.Height := tabla\_Image22.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(22);

nameOp:= intTostr(22);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(22)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image23Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image23.Width := tabla\_Image23.Width + 5;

tabla\_Image23.Height := tabla\_Image23.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(23);

nameOp:= intTostr(23);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(23)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image24Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image24.Width := tabla\_Image24.Width + 5;

tabla\_Image24.Height := tabla\_Image24.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(24);

nameOp:= intTostr(24);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(24)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image25Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image25.Width := tabla\_Image25.Width + 5;

tabla\_Image25.Height := tabla\_Image25.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(25);

nameOp:= intTostr(25);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(25)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image26Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image26.Width := tabla\_Image26.Width + 5;

tabla\_Image26.Height := tabla\_Image26.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(26);

nameOp:= intTostr(26);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(26)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image27Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image27.Width := tabla\_Image27.Width + 5;

tabla\_Image27.Height := tabla\_Image27.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(27);

nameOp:= intTostr(27);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(27)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image28Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image28.Width := tabla\_Image28.Width + 5;

tabla\_Image28.Height := tabla\_Image28.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(28);

nameOp:= intTostr(28);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(28)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image29Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image29.Width := tabla\_Image29.Width + 5;

tabla\_Image29.Height := tabla\_Image29.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(29);

nameOp:= intTostr(29);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(29)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image2Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image2.Width := tabla\_Image2.Width + 5;

tabla\_Image2.Height := tabla\_Image2.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(2);

nameOp:= intTostr(2);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(2)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image30Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image30.Width := tabla\_Image30.Width + 5;

tabla\_Image30.Height := tabla\_Image30.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(30);

nameOp:= intTostr(30);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(30)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image31Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image31.Width := tabla\_Image31.Width + 5;

tabla\_Image31.Height := tabla\_Image31.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(31);

nameOp:= intTostr(31);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(31)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image32Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image32.Width := tabla\_Image32.Width + 5;

tabla\_Image32.Height := tabla\_Image32.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(32);

nameOp:= intTostr(32);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(32)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image33Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image33.Width := tabla\_Image33.Width + 5;

tabla\_Image33.Height := tabla\_Image33.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(33);

nameOp:= intTostr(33);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(33)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image34Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image34.Width := tabla\_Image34.Width + 5;

tabla\_Image34.Height := tabla\_Image34.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(34);

nameOp:= intTostr(34);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(34)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image35Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image35.Width := tabla\_Image35.Width + 5;

tabla\_Image35.Height := tabla\_Image35.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(35);

nameOp:= intTostr(35);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(35)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image36Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image36.Width := tabla\_Image36.Width + 5;

tabla\_Image36.Height := tabla\_Image36.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(36);

nameOp:= intTostr(36);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(36)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image4Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image4.Width := tabla\_Image4.Width + 5;

tabla\_Image4.Height := tabla\_Image4.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(4);

nameOp:= intTostr(4);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(4)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image5Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image5.Width := tabla\_Image5.Width + 5;

tabla\_Image5.Height := tabla\_Image5.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(5);

nameOp:= intTostr(5);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(5)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image6Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image6.Width := tabla\_Image6.Width + 5;

tabla\_Image6.Height := tabla\_Image6.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(6);

nameOp:= intTostr(6);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(6)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image7Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image7.Width := tabla\_Image7.Width + 5;

tabla\_Image7.Height := tabla\_Image7.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(7);

nameOp:= intTostr(7);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(7)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image8Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image8.Width := tabla\_Image8.Width + 5;

tabla\_Image8.Height := tabla\_Image8.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(8);

nameOp:= intTostr(8);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(8)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image9Click(Sender: TObject);

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image9.Width := tabla\_Image9.Width + 5;

tabla\_Image9.Height := tabla\_Image9.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Pleno;

valor:= intTostr(9);

nameOp:= intTostr(9);

if ((AdminLog) and (BtnTrampaIsActive)) then

TableroTrampaSeleccionado(9)

else

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImageImparesClick(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImageImpares.Width := tabla\_ImageImpares.Width + 5;

tabla\_ImageImpares.Height := tabla\_ImageImpares.Height + 10;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_ParImpar;

valor:= 'I';

nameOp:= '"IMPARES"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImageNegroClick(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImageNegro.Width := tabla\_ImageNegro.Width + 5;

tabla\_ImageNegro.Height := tabla\_ImageNegro.Height + 10;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Rojonegro;

valor:= 'N';

nameOp:= '"Negro"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImageParesClick(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImagePares.Width := tabla\_ImagePares.Width + 5;

tabla\_ImagePares.Height := tabla\_ImagePares.Height + 10;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_ParImpar;

valor:= 'P';

nameOp:= '"PARES"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImagePrimeraColClick(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImagePrimeraCol.Width := tabla\_ImagePrimeraCol.Width + 5;

tabla\_ImagePrimeraCol.Height := tabla\_ImagePrimeraCol.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Columna;

valor:= intTostr(1);

nameOp:= '"Primer columna"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImagePrimeros12Click(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImagePrimeros12.Width := tabla\_ImagePrimeros12.Width + 20;

tabla\_ImagePrimeros12.Height := tabla\_ImagePrimeros12.Height + 5;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Docena;

valor:= intTostr(1);

nameOp:= '"Primeros doce"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImageRojoClick(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImageRojo.Width := tabla\_ImageRojo.Width + 5;

tabla\_ImageRojo.Height := tabla\_ImageRojo.Height + 10;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Rojonegro;

valor:= 'R';

nameOp:= '"Rojo"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImageSegundaColClick(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImageSegundaCol.Width := tabla\_ImageSegundaCol.Width + 5;

tabla\_ImageSegundaCol.Height := tabla\_ImageSegundaCol.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Columna;

valor:= intTostr(2);

nameOp:= '"Segunda columna"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImageSegundos12Click(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImageSegundos12.Width := tabla\_ImageSegundos12.Width + 20;

tabla\_ImageSegundos12.Height := tabla\_ImageSegundos12.Height + 5;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Docena;

valor:= intTostr(2);

nameOp:= '"Segundos doce"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImageTerceraColClick(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImageTerceraCol.Width := tabla\_ImageTerceraCol.Width + 5;

tabla\_ImageTerceraCol.Height := tabla\_ImageTerceraCol.Height + 15;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Columna;

valor:= intTostr(3);

nameOp:= '"Tercer columna"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_ImageTerceros12Click(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_ImageTerceros12.Width := tabla\_ImageTerceros12.Width + 20;

tabla\_ImageTerceros12.Height := tabla\_ImageTerceros12.Height + 5;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_Docena;

valor:= intTostr(3);

nameOp:= '"Terceros doce"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image\_19a36Click(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image\_19a36.Width := tabla\_Image\_19a36.Width + 5;

tabla\_Image\_19a36.Height := tabla\_Image\_19a36.Height + 10;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_PasaFalta;

valor:= 'F';

nameOp:= '"Falta(19-36)"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.tabla\_Image\_1a18Click(Sender: TObject);

//Esta casilla sera estará disponible si se lleva a cabo una apuesta y no cuando hace trampa

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Lbl\_usuarioActual.Caption);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

tabla\_Image\_1a18.Width := tabla\_Image\_1a18.Width + 5;

tabla\_Image\_1a18.Height := tabla\_Image\_1a18.Height + 10;

panel\_tablero.Enabled:= false;

Nomenclador:= Type\_APUESTA.Nom\_PasaFalta;

valor:= 'P';

nameOp:= '"Pasa (1-18)"';

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

if not (regJugador.Bloqueado) then //si no esta bloqueado quien va a llevar a cabo la apuesta ...

begin

if (RegJuegoRuleta.Estado<2) then

TableroLegalSeleccionado(Nomenclador,valor,nameOp)

else

begin

F\_Juego.panel\_tablero.Enabled:=false;

refrezcarPantallaNormalUser();

AutoSize\_ImagenesTablero();

end;

end

else

begin

MessageDlg('Lo siento, ha sido bloqueado!!' ,mtWarning , [mbOK], 0);

SalirDelJuego();

end;

end;

{------------------------------------------------------------------------------}

//El tablero volvera a su tamaño original luego de que se haya efecuado un 'click'

Procedure TF\_Juego.AutoSize\_ImagenesTablero;

begin

tabla\_Image0.AutoSize := true;tabla\_Image0.AutoSize := false;

tabla\_Image1.AutoSize := true;tabla\_Image1.AutoSize := false;

tabla\_Image2.AutoSize := true;tabla\_Image2.AutoSize := false;

tabla\_Image3.AutoSize := true;tabla\_Image3.AutoSize := false;

tabla\_Image3.AutoSize := true;tabla\_Image3.AutoSize := false;

tabla\_Image3.AutoSize := true;tabla\_Image3.AutoSize := false;

tabla\_Image4.AutoSize := true;tabla\_Image4.AutoSize := false;

tabla\_Image5.AutoSize := true;tabla\_Image5.AutoSize := false;

tabla\_Image6.AutoSize := true;tabla\_Image6.AutoSize := false;

tabla\_Image7.AutoSize := true;tabla\_Image7.AutoSize := false;

tabla\_Image8.AutoSize := true;tabla\_Image8.AutoSize := false;

tabla\_Image9.AutoSize := true;tabla\_Image9.AutoSize := false;

tabla\_Image10.AutoSize := true;tabla\_Image10.AutoSize := false;

tabla\_Image11.AutoSize := true;tabla\_Image11.AutoSize := false;

tabla\_Image12.AutoSize := true;tabla\_Image12.AutoSize := false;

tabla\_Image13.AutoSize := true;tabla\_Image13.AutoSize := false;

tabla\_Image14.AutoSize := true;tabla\_Image14.AutoSize := false;

tabla\_Image15.AutoSize := true;tabla\_Image15.AutoSize := false;

tabla\_Image16.AutoSize := true;tabla\_Image16.AutoSize := false;

tabla\_Image17.AutoSize := true;tabla\_Image17.AutoSize := false;

tabla\_Image18.AutoSize := true;tabla\_Image18.AutoSize := false;

tabla\_Image19.AutoSize := true;tabla\_Image19.AutoSize := false;

tabla\_Image20.AutoSize := true;tabla\_Image20.AutoSize := false;

tabla\_Image21.AutoSize := true;tabla\_Image21.AutoSize := false;

tabla\_Image22.AutoSize := true;tabla\_Image22.AutoSize := false;

tabla\_Image23.AutoSize := true;tabla\_Image23.AutoSize := false;

tabla\_Image24.AutoSize := true;tabla\_Image24.AutoSize := false;

tabla\_Image25.AutoSize := true;tabla\_Image25.AutoSize := false;

tabla\_Image26.AutoSize := true;tabla\_Image26.AutoSize := false;

tabla\_Image27.AutoSize := true;tabla\_Image27.AutoSize := false;

tabla\_Image28.AutoSize := true;tabla\_Image28.AutoSize := false;

tabla\_Image29.AutoSize := true;tabla\_Image29.AutoSize := false;

tabla\_Image30.AutoSize := true;tabla\_Image30.AutoSize := false;

tabla\_Image31.AutoSize := true;tabla\_Image31.AutoSize := false;

tabla\_Image32.AutoSize := true;tabla\_Image32.AutoSize := false;

tabla\_Image33.AutoSize := true;tabla\_Image33.AutoSize := false;

tabla\_Image34.AutoSize := true;tabla\_Image34.AutoSize := false;

tabla\_Image35.AutoSize := true;tabla\_Image35.AutoSize := false;

tabla\_Image36.AutoSize := true;tabla\_Image36.AutoSize := false;

tabla\_ImagePrimeraCol.AutoSize:= true;tabla\_ImagePrimeraCol.AutoSize:= false;

tabla\_ImageSegundaCol.AutoSize:= true;tabla\_ImageSegundaCol.AutoSize:= false;

tabla\_ImageTerceraCol.AutoSize:= true;tabla\_ImageTerceraCol.AutoSize:= false;

tabla\_ImagePrimeros12.AutoSize:= true;tabla\_ImagePrimeros12.AutoSize:= false;

tabla\_ImageSegundos12.AutoSize:= true;tabla\_ImageSegundos12.AutoSize:= false;

tabla\_ImageTerceros12.AutoSize:= true;tabla\_ImageTerceros12.AutoSize:= false;

tabla\_Image\_1a18.AutoSize:= true;tabla\_Image\_1a18.AutoSize:= false;

tabla\_Image\_19a36.AutoSize:= true;tabla\_Image\_19a36.AutoSize:= false;

tabla\_ImagePares.AutoSize:= true;tabla\_ImagePares.AutoSize:= false;

tabla\_ImageImpares.AutoSize:= true;tabla\_ImageImpares.AutoSize:= false;

tabla\_ImageRojo.AutoSize:= true;tabla\_ImageRojo.AutoSize:= false;

tabla\_ImageNegro.AutoSize:= true;tabla\_ImageNegro.AutoSize:= false;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Btn\_closePnelLoserClick(Sender: TObject);

var

i: integer;

begin

for i:=0 to menuFormJuego.Items.Count-1 do

menuFormJuego.Items[I].Enabled:= true;

//Un ves mostrado el panel de ganador o perdedor, dejo a usuario ver premios

menuFormJuego.Items[2].Visible:= true;

Panel\_Jugadores.Enabled:= true;

Panel\_Winner.Visible:= false;

Panel\_Loser.Visible:= false;

Panel\_Winner.Color:= clRed;

Lbl\_waiting.Font.Color:= clBlack;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Btn\_closePnelWinnerClick(Sender: TObject);

var

i: integer;

begin

for i:=0 to menuFormJuego.Items.Count-1 do

menuFormJuego.Items[I].Enabled:= true;

Panel\_Jugadores.Enabled:= true;

Panel\_Winner.Visible:= false;

Panel\_ShowWinner.Visible:= false;

(gifWinner.Picture.Graphic as TGIFImage).Animate:= false;

gifWinner.Visible:= false;

Panel\_Winner.Color:= clRed;

Lbl\_waiting.Font.Color:= clBlack;

end;

{--------------------------------------------------------------------------------}

Procedure TF\_Juego.TableroLegalSeleccionado(nomenclador:Type\_APUESTA.tNomenclador;valor:Type\_APUESTA.tNomenclador; nameOp:string);

var

Cant: tCantidad; // cantidad jugadores en general

Cant\_Desbloq: tCantidad; // cantidad de jugadores desbloqueados

Cant\_Activos: tCantidad; // cantidad de jugadores activos

begin

Cant\_Desbloq:= 0;

Cant\_Activos:= 0;

cant:= ME\_JUGADORES.Cantidad\_Jugadores(ME\_JUGADOR);

Lib\_AuxJuego.Cant\_JugadoresDesbloqueados(ME\_JUGADORES.Raiz(ME\_JUGADOR), Cant\_Desbloq);

Lib\_AuxJuego.Cant\_JugadoresActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR), Cant\_Activos);

//Usuario comun

if not (Form\_login.AdminLog) then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿Esta seguro que quiere apostar por: '+nameOp+ ' ?') then

begin

SoloListar:= False; //

F\_Apostar.Show;

F\_Apostar.Lbl\_SelectJugador.hide;

F\_Apostar.Lbl\_ListaJugadores.hide;

F\_Apostar.Grid\_ListaJugadores.hide;

F\_Apostar.Lbl\_ListaDeApuestas.Show;

F\_Apostar.Grid\_ListaApuestas.Show;

F\_Apostar.Panel\_DatosApostar.show;

F\_Apostar.Edit\_nomenclador.Text:= nomenclador;

F\_Apostar.Edit\_valor.Text:=valor;

F\_Apostar.Edit\_nameJugador.Text:= NickLogueado;

Form\_Apostar.F\_Apostar.Edit\_importe.SetFocus;

F\_Juego.ListandoApuestas();

end

else

begin

AutoSize\_ImagenesTablero();

panel\_tablero.Enabled:= true;

end;

end

else //Croupier

begin

F\_Apostar.Lbl\_ListaJugadores.hide;

F\_Apostar.Lbl\_ListaDeApuestas.hide;

F\_Apostar.Grid\_ListaApuestas.hide

;

F\_Apostar.Lbl\_SelectJugador.Show;

F\_Apostar.Grid\_ListaJugadores.Show;

F\_Apostar.Edit\_nameJugador.Clear;

if (not ME\_JUGADORES.MeVacio\_Jugadores(ME\_JUGADOR) and (cant>1)) then //si cant es >1 entonces al menos hay un jugador que no es el administrador

begin

//Si al menos hay un jugador desbloqueado

if (Cant\_Desbloq >=1) then

begin

//Si hay jugadores activos

if (Cant\_Activos>=1) then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿Esta seguro que quiere apostar por: '+nameOp+ ' ?') then

begin

F\_Apostar.Show;

F\_Apostar.Panel\_DatosApostar.show;

F\_Apostar.Edit\_nomenclador.Text:= nomenclador;

F\_Apostar.Edit\_valor.Text:=valor;

F\_Apostar.ListandoJugadores();

end;

end

else

MessageDlg('No hay jugadores activos!',mtWarning, [mbOK], 0);

end

else

MessageDlg('Existen jugadores, pero estan bloqueados',mtWarning, [mbOK], 0);

end

else

MessageDlg('No existen jugadores!!',mtWarning, [mbOK], 0);

AutoSize\_ImagenesTablero();

panel\_tablero.Enabled:= true;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.TableroTrampaSeleccionado(num\_bolilla:Type\_JUEGO.tBolilla);

begin //Soy el Croupier y estoy haciendo trampa

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿Esta seguro que quiere cometer trampa con la bolilla Nro: ' +intTostr(num\_bolilla)+ ' ?') then

begin

Lbl\_nroBolillaG.Caption:= intTostr(num\_bolilla);

Lbl\_nroBolillaG.Font.Color:=clRed;

Lbl\_bolillaASalir.Font.Color:=clRed;

Spbtn\_trampa.Enabled:= false;

//Se que estoy en estado 3 (bolilla tirada), actualizo regJuegoRuleta con la "bolilla ganadora trampa"

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

RegJuegoRuleta.NroJugada:= RegJuegoRuleta.NroJugada;

RegJuegoRuleta.FechaHora:= Now();

RegJuegoRuleta.Estado:= RegJuegoRuleta.Estado;

RegJuegoRuleta.Bolilla:= num\_bolilla;

ME\_JUEGO.ModificoUltimo(ME\_JUEGORULETA, RegJuegoRuleta);

Spbtn\_tirarBolilla.Enabled:=false;

Spbtn\_repartirPremios.Enabled:=true;

BtnTrampaIsActive:= false;

DesactiveBtnTrampa();

end

else

begin

panel\_tablero.Enabled:= true;

F\_Apostar.Edit\_nomenclador.Clear;

F\_Apostar.Edit\_valor.Clear;

end;

AutoSize\_ImagenesTablero();

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showWinner2Timer(Sender: TObject);

begin

Timer\_showWinner2.Enabled:= false;

Panel\_Winner.Visible:= true;

Timer\_showWinner3.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showWinner3Timer(Sender: TObject);

begin

Timer\_showWinner3.Enabled:= false;

Panel\_Winner.Visible:= false;

Timer\_showWinner4.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showWinner4Timer(Sender: TObject);

begin

Timer\_showWinner4.Enabled:= false;

Panel\_Winner.Visible:= true;

Panel\_Winner.Color:= clBlack;

Lbl\_waiting.Font.Color:= clRed;

Lbl\_waiting.Caption:= 'Aguarde ' +#13+ ' unos' +#13+ 'segundos ..';

Timer\_showWinner5.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showWinner5Timer(Sender: TObject);

begin

Timer\_showWinner5.Enabled:= false;

Panel\_Winner.Visible:= false;

Timer\_showWinner6.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showWinner6Timer(Sender: TObject);

begin

Timer\_showWinner6.Enabled:= false;

Panel\_Winner.Visible:= true;

if (ShowWinner) then

begin

Panel\_ShowWinner.Visible:= true;

gifWinner.Visible:= true;

(gifWinner.Picture.Graphic as TGIFImage).Animate:= true;

end

else

Panel\_Loser.Visible:= true;

Lbl\_saldoAcumulado.Show;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showLoser1Timer(Sender: TObject);

begin

Timer\_showLoser1.Enabled:= false;

Panel\_Loser.Visible:= false;

Timer\_showLoser2.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showLoser2Timer(Sender: TObject);

begin

Panel\_Loser.Visible:= true;

Timer\_showLoser2.Enabled:= false;

Timer\_showLoser3.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showLoser3Timer(Sender: TObject);

begin

Timer\_showLoser3.Enabled:= false;

Panel\_Loser.Visible:= false;

Timer\_showLoser4.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showLoser4Timer(Sender: TObject);

begin

Timer\_showLoser4.Enabled:= false;

Panel\_Loser.Visible:= true;

Timer\_showLoser5.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showLoser5Timer(Sender: TObject);

begin

Timer\_showLoser5.Enabled:= false;

Panel\_Loser.Visible:= false;

Timer\_showLoser6.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showLoser6Timer(Sender: TObject);

begin

Timer\_showLoser6.Enabled:= false;

Panel\_Loser.Visible:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.Timer\_showWinner1Timer(Sender: TObject);

begin

Timer\_showWinner1.Enabled:= false;

Timer\_showWinner2.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Juego.refrezcarPantallaNormalUser();

var

AuxRegJuego: Type\_JUEGO.TipoRegDatos;

saldoActual: tImporte;

cant:tCantidad;

i: integer;

corte: boolean;

monto: tImporte;

begin

//Actualizo fecha y hora a mostrar

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual);

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, AuxRegJuego);

//Si el estado es mayor o igual a 2 (no va mas), dejo a usuario ver apuestas efectuadas

if ((AuxRegJuego.estado>=2) and (AuxRegJuego.estado<4)) then

menuFormJuego.Items[1].Visible:= true;

CASE (AuxRegJuego.estado) of

0:begin

Lbl\_estadoJuego.Caption:='"Juego Creado"';

//acumulacionDeApuestas:= 0;

saldoAcumulado:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, regJugador.Nick);

Lbl\_saldoAcumulado.Caption:= '$ ' + inttostr(saldoAcumulado);

end;//end estado=0

1:begin

Lbl\_estadoJuego.Caption:='"Hagan sus apuestas"';

panel\_tablero.Enabled:= true;

saldoActual:= F\_Apostar.obtenerSaldoActualJugador(NickLogueado);

F\_Juego.Lbl\_saldoAcumulado.Caption:= '$ ' + intTostr(saldoActual);

end;//end estado=1

2:begin

Lbl\_estadoJuego.Caption:='"No va más"';

MessageDlg('Se han cerrado las apuestas!',mtInformation , [mbOK], 0);

//acumulacionDeApuestas:= 0;

AutoSize\_ImagenesTablero();

saldoAcumulado:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, regJugador.Nick);

Lbl\_saldoAcumulado.Caption:= '$ ' + inttostr(saldoAcumulado);

end;//end estado=2

3:begin

Lbl\_estadoJuego.Caption:='"Bolilla tirada"';

saldoAcumulado:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, regJugador.Nick);

Lbl\_saldoAcumulado.Caption:= '$ ' + inttostr(saldoAcumulado);

end;//end estado=3

4:begin

monto:= 0;

Lbl\_estadoJuego.Caption:='"Premios repartidos y fin de juego"';

saldoAcumulado:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, regJugador.Nick);

Lbl\_saldoAcumulado.Caption:= '$ ' + inttostr(saldoAcumulado);

Lbl\_saldoAcumulado.Hide;

//Muestro formulario ganador

Lib\_AuxJuego.Premios\_AcumuladosXJugadorUnaJugada(ME\_GANADORES.Raiz(ME\_GANADOR), regJugador.Nick, RegJuegoRuleta.NroJugada, monto);

if (monto > 0) then

begin

ShowWinner:= true;

Timer\_showWinner1.Enabled:= true;

Lbl\_montoWinner.Caption:= intTostr(monto);

Lbl\_bolillaGWinner.Caption:= intTostr(AuxRegJuego.Bolilla);

end

else

begin

ShowWinner:= false;

Timer\_showWinner1.Enabled:= true;

Lbl\_bolillaGLoser.Caption:= intTostr(AuxRegJuego.Bolilla);

end;

Lbl\_waiting.Caption:= 'Aguarde ' +#13+ ' unos' +#13+ 'segundos';

//Deshabilito panel jugadores

Panel\_Jugadores.Enabled:= false;

//Deshabilito menu principal del form\_juego al usuario comun

for i:=0 to menuFormJuego.Items.Count-1 do

menuFormJuego.Items[I].Enabled:= false;

end;//end estado=4

end;//end CASE

//consulto si el estado del juego es >= 2 para asegurarme que nadie puede apostar si el ya 'no va mas'

if (AuxRegJuego.Estado>=2) then

begin

//Oculto tablero del jugador

panel\_tablero.Hide;

Panel\_NameTablero.Hide;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.SalirDelJuego();

begin

Form\_Juego.F\_Juego.close;

Form\_Jugadores.F\_Jugadores.Show;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.pantallaIniJuegoCrupier();

begin

panel\_tablero.Enabled:= false;

AutoSize\_ImagenesTablero();

Lbl\_usuarioActual.Caption:= NickLogueado;

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

//--------------------------------

Panel\_Jugadores.Hide;

Panel\_Crupier.Show;

F\_Juego.Lbl\_CantFictActivados.Caption:='--';;

Lbl\_cantFicticios.Caption:='--';

Lbl\_nroBolillaG.Caption:='--';

//--------------------------------

Spbtn\_CrearPartida.Enabled:=true;

Spbtn\_bloqDesbloq.Enabled:= true;

Spbtn\_haganApuestas.Enabled:=false;

Spbtn\_noVaMas.Enabled:=false;

Spbtn\_tirarBolilla.Enabled:=false;

Spbtn\_repartirPremios.Enabled:=false;

Spbtn\_activarFicticios.Enabled:= false;

Spbtn\_obsequiarCred.Enabled:= false;

Spbtn\_trampa.Enabled:= false;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.pantallaIniJuegoJugadores();

var

i:tPos;

begin

panel\_tablero.Enabled:= false;

AutoSize\_ImagenesTablero();

Lbl\_usuarioActual.Caption:= NickLogueado;

ActualizarFechaHora(Lbl\_fechaActual, Lbl\_horaActual); //Actualizo fecha y hora a mostrar

Panel\_Crupier.hide;

Panel\_Jugadores.Show;

ME\_JUEGO.RegistroNulo(RegJuegoRuleta);

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta); //

Lbl\_nroPartida.Caption:= RegJuegoRuleta.NroJugada; //

panel\_tablero.Show;

Panel\_NameTablero.Show;

Lbl\_ApuestaMax.Caption:= '$ ' + intTostr(apuestaMax);

Lbl\_ApuestaMin.Caption:= '$ ' + intTostr(apuestaMin);

menuFormJuego.Items[0].Visible:= true;

menuFormJuego.Items[1].Visible:= false;

menuFormJuego.Items[2].Visible:= false;

//for i:=1 to menuFormJuego.Items.Count-1 do

// menuFormJuego.Items[i].Visible:= false;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.ListarPremiados(Nro\_Jugada: Type\_JUEGO.tClave);

begin

ME\_JUEGO.RegistroNulo(RegJuegoRuleta);

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, RegJuegoRuleta);

Form\_Premios.F\_Premios.show;

Lib\_Auxiliar.limpiarGrid(F\_Premios.Grid\_Premiados);

Lib\_Auxiliar.EncabezadoGridPremios(F\_Premios.Grid\_Premiados);

F\_Premios.Grid\_premiados.RowCount:=1;

Lib\_AuxJuego.Listado\_Premios(ME\_GANADORES.Raiz(ME\_GANADOR), Nro\_Jugada, F\_Premios.Grid\_Premiados);

F\_Premios.Grid\_premiados.FixedRows:=1;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.ListandoApuestas();

begin

Lib\_Auxiliar.limpiarGrid(F\_Apostar.Grid\_ListaApuestas);

F\_Apostar.Grid\_ListaApuestas.RowCount:=1;

Lib\_Auxiliar.EncabezadosComunApuestas(F\_Apostar.Grid\_ListaApuestas);

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

if (Lib\_AuxJuego.JugadorTieneApuestasEnPartida(RegJuegoRuleta.NroJugada, NickLogueado)) then

begin

F\_Apostar.Grid\_ListaApuestas.RowCount:=1;

Lib\_AuxJuego.Listado\_ApuestasUnJugador(F\_Apostar.Grid\_ListaApuestas, RegJuegoRuleta.NroJugada, NickLogueado);

//F\_Apostar.Grid\_ListaApuestas.FixedRows:=1;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Juego.ActualizarFechaHora(var Lbl\_fecha:TLabel; var Lbl\_hora: TLabel);

begin

Lbl\_fecha.Caption:= datetostr(now);

Lbl\_hora.Caption:= timetostr(now);

end;

END.

### Formulario Apostar

unit Form\_Apostar;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Lib\_AuxJuego,

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Grids, Vcl.StdCtrls, Vcl.Buttons,

Vcl.ExtCtrls;

type

TF\_Apostar = class(TForm)

Panel\_Listado: TPanel;

Lbl\_SelectJugador: TLabel;

Panel\_DatosApostar: TPanel;

Lbl\_nameJugador: TLabel;

Edit\_nameJugador: TEdit;

btn\_okeyApostar: TBitBtn;

Panel\_grid: TPanel;

Lbl\_nomenclador: TLabel;

Edit\_nomenclador: TEdit;

Lbl\_valor: TLabel;

Edit\_valor: TEdit;

Lbl\_importe: TLabel;

Edit\_importe: TEdit;

Lbl\_ListaDeApuestas: TLabel;

Lbl\_ListaJugadores: TLabel;

Grid\_ListaApuestas: TStringGrid;

Grid\_ListaJugadores: TStringGrid;

BitBtn1: TBitBtn;

procedure FormCreate(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

procedure FormShow(Sender: TObject);

Procedure Listado\_GeneralJugadores(RaizJugadores: Lib\_Auxiliar.tPos);

Function obtenerSaldoActualJugador(clave:tClave):tImporte;

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure Btn\_ListarJugadoresClick(Sender: TObject);

procedure Grid\_ListaApuestasClick(Sender: TObject);

procedure Grid\_ListaJugadoresDrawCell(Sender: TObject; ACol, ARow: Integer;

Rect: TRect; State: TGridDrawState);

procedure Grid\_ListaJugadoresClick(Sender: TObject);

procedure btn\_okeyApostarClick(Sender: TObject);

procedure Edit\_importeKeyPress(Sender: TObject; var Key: Char);

procedure Grid\_ListaApuestasDrawCell(Sender: TObject; ACol, ARow: Integer;

Rect: TRect; State: TGridDrawState);

procedure ListandoJugadores();

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_Apostar: TF\_Apostar;

SaldoAcumulado, SaldoApuestas\_EnPartida, saldoActual:tImporte;

implementation

uses

Form\_login, Form\_Juego;

{$R \*.dfm}

{--------------------------------------------------------------------------------}

procedure TF\_Apostar.BitBtn1Click(Sender: TObject);

begin

F\_Apostar.Close;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Apostar.FormClose(Sender: TObject; var Action: TCloseAction);

begin

F\_Juego.Enabled:= true;

F\_Juego.AutoSize\_ImagenesTablero();

F\_Juego.panel\_tablero.Enabled:= true;

if ((not AdminLog) and (not SoloListar)) then

begin

saldoActual:= F\_Apostar.obtenerSaldoActualJugador(NickLogueado);

F\_Juego.Lbl\_saldoAcumulado.Caption:= '$ ' + intTostr(saldoActual);

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Apostar.FormCreate(Sender: TObject);

begin

self.Position := poScreenCenter;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Apostar.FormShow(Sender: TObject);

begin

F\_Juego.Enabled:= false;

F\_Apostar.Grid\_ListaJugadores.RowCount:=1;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Apostar.Grid\_ListaApuestasClick(Sender: TObject);

var

i, posElim: tPos;

Nick: type\_JUGADOR.tClave;

Importe: tImporte;

enc: boolean;

begin

enc:= false;

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA, regJuegoRuleta);

if (not SoloListar) then

begin

//Si el estado es igual a 1 'hagan sus apuestas'

if (regJuegoRuleta.Estado = 1) then

begin

if (Grid\_ListaApuestas.Row <> 0) then

begin

Nick:= Grid\_ListaApuestas.Cells[0, Grid\_ListaApuestas.Row];

Nomenclador:= Grid\_ListaApuestas.Cells[1, Grid\_ListaApuestas.Row];

Valor:= Grid\_ListaApuestas.Cells[2, Grid\_ListaApuestas.Row];

Importe:= strToint(Grid\_ListaApuestas.Cells[3, Grid\_ListaApuestas.Row]);

i:= ME\_APUESTAS.Primero(ME\_APUESTA);

while ((i <> ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA)) or (not enc)) do

begin

ME\_APUESTAS.CapturarInfoME\_Apuestas(ME\_APUESTA, i, RegApuesta);

if (RegApuesta.NroJugada = RegJuegoRuleta.NroJugada) and (RegApuesta.Nick = Nick) then

if ((RegApuesta.Nomenclador = Nomenclador) and (RegApuesta.Valor = Valor) and (RegApuesta.Importe = Importe)) then

begin

enc:= true;

posElim:= i;

end;

i:= ME\_APUESTAS.Proximo(ME\_APUESTA, i);

end;//while

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿Esta seguro que quiere eliminar la apuesta?') then

begin

ME\_APUESTAS.EliminarInfoME\_Apuestas(ME\_APUESTA, posElim);

F\_Juego.ListandoApuestas();

end;

end;

end;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Apostar.Grid\_ListaApuestasDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

begin

with (Sender as TStringGrid) do

begin

if (ARow = 0)

then

begin

Canvas.Brush.Color := clBtnFace;

Canvas.TextRect(Rect, Rect.Left + (Rect.Right - Rect.Left - Canvas.TextWidth(Grid\_ListaApuestas.Cells[ACol,ARow]) + 1) div 2, Rect.Top + 2, Grid\_ListaApuestas.Cells[ACol,ARow]);

end

else

begin

Canvas.Font.Color := clblack;

if (ARow mod 2 = 0)

then

Canvas.Brush.Color := $00E1FFF9

else

Canvas.Brush.Color := $00FFEBDF;

Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top + 2, cells[acol, arow]);

Canvas.FrameRect(Rect);

end;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Apostar.Grid\_ListaJugadoresClick(Sender: TObject);

begin

if (AdminLog) then

begin

if (Grid\_ListaJugadores.Row <> 0) then

begin

Edit\_nameJugador.Text := Grid\_ListaJugadores.Cells[0,Grid\_ListaJugadores.Row];

end;

if (Edit\_nameJugador.Text<>'') then

begin

Edit\_importe.Enabled:=true;

Edit\_importe.SetFocus;

end

else

Edit\_importe.Enabled:= false;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Apostar.Grid\_ListaJugadoresDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

begin

with (Sender as TStringGrid) do

begin

if (ARow = 0)

then

begin

Canvas.Brush.Color := clBtnFace;

Canvas.TextRect(Rect, Rect.Left + (Rect.Right - Rect.Left - Canvas.TextWidth(Grid\_ListaJugadores.Cells[ACol,ARow]) + 1) div 2, Rect.Top + 2, Grid\_ListaJugadores.Cells[ACol,ARow]);

end

else

begin

Canvas.Font.Color := clblack;

if (ARow mod 2 = 0)

then

Canvas.Brush.Color := $00E1FFF9

else

Canvas.Brush.Color := $00FFEBDF;

Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top + 2, cells[acol, arow]);

Canvas.FrameRect(Rect);

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_Apostar.Btn\_ListarJugadoresClick(Sender: TObject);

begin

ListandoJugadores();

end;

{------------------------------------------------------------------------------}

procedure TF\_Apostar.btn\_okeyApostarClick(Sender: TObject);

begin

if (Edit\_nameJugador.Text<>'') then

begin

if (Edit\_importe.Text<>'') then

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,posJugador,Edit\_nameJugador.Text);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,regJugador,posJugador);

if not (regJugador.Bloqueado) then

begin

if (strToint(Edit\_Importe.Text) >= ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA)) and (strToint(Edit\_Importe.Text) <= ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA)) then

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

SaldoApuestas\_EnPartida:= ME\_APUESTAS.SaldoApuestasPartida(ME\_APUESTA,Edit\_nameJugador.Text,RegJuegoRuleta.NroJugada)+strToint(Edit\_importe.Text);

SaldoAcumulado:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE,Edit\_nameJugador.Text);

if ((SaldoApuestas\_EnPartida <= SaldoAcumulado)) then

begin

if (RegJuegoRuleta.Estado = 1) then

begin //todo correcto para llevarse acabo la apuesta ...

//acumulacionDeApuestas:= acumulacionDeApuestas + strToint(Edit\_importe.Text);

//Armo registroApusta para insertar la nueva apuesta

regApuesta.NroJugada:= RegJuegoRuleta.NroJugada;

regApuesta.Nick:= Edit\_nameJugador.Text;

regApuesta.Nomenclador:= Edit\_nomenclador.Text;

regApuesta.Valor:=Edit\_valor.Text;

regApuesta.Importe:= strToint(Edit\_importe.Text);

ME\_APUESTAS.InsertarInfoME\_Apuestas(ME\_APUESTA,regApuesta,ME\_APUESTAS.PosNula\_Apuestas(ME\_APUESTA));

//Obtengo el sueldo actual del jugador que se vera reflejado en su cuenta cuando el estado sea "fin del juego"

saldoActual:= obtenerSaldoActualJugador(Edit\_nameJugador.Text);

F\_Juego.Lbl\_saldoAcumulado.Caption:=intTostr(saldoActual);

Edit\_importe.Clear;

Edit\_importe.SetFocus;

//Grilla apuestas

Lib\_Auxiliar.limpiarGrid(Grid\_ListaApuestas);

Lib\_Auxiliar.EncabezadosComunApuestas(Grid\_ListaApuestas);

F\_Apostar.Grid\_ListaApuestas.RowCount:=1;

Lib\_AuxJuego.Listado\_GeneralApuestas(Grid\_ListaApuestas, RegJuegoRuleta.NroJugada);

F\_Apostar.Grid\_ListaApuestas.FixedRows:=1;

//-----------------------------------------------------------------------------------

Form\_Juego.F\_Juego.AutoSize\_ImagenesTablero();

Form\_Juego.F\_Juego.panel\_tablero.Enabled:= true;

F\_Apostar.Close;

MessageDlg('Apuesta realizada con éxito!',mtInformation, [mbOK], 0);

end

else

//doble validacion, ya el el boton para abrir form de apuestas estara deshabilitado si el tiempo de apostar culmino

MessageDlg('Ya no se permite apuestas. Finalizo el tiempo de apostar',mtInformation, [mbOK], 0);

end

else

begin

MessageDlg('Su saldo no es suficiente para llevar a cabo la apuesta!',mtWarning, [mbOK], 0);

Edit\_importe.Clear;

Edit\_importe.SetFocus;

end;

end

else

begin

MessageDlg('La apuesta minima permitida es de $ ' +intTostr(ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA))+

', y la apuesta maxima permitida es de $ ' +intTostr(ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA)),mtWarning, [mbOK], 0);

Edit\_importe.Clear;

Edit\_importe.SetFocus;

end;

end

else

MessageDlg('El jugador se encuentra bloqueado!',mtWarning, [mbOK], 0);

end

else

MessageDlg('Complete el importe a apostar!',mtWarning, [mbOK], 0);

end

else

MessageDlg('Por favor, seleccione un jugador de la grilla!',mtWarning, [mbOK], 0);

end;

{------------------------------------------------------------------------------}

procedure TF\_Apostar.Edit\_importeKeyPress(Sender: TObject; var Key: Char);

const

CARAC\_HABILITADOS = ['0'..'9', #0..#27];//solo numero y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

{------------------------------------------------------------------------------}

Procedure TF\_Apostar.Listado\_GeneralJugadores(RaizJugadores: Lib\_Auxiliar.tPos);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if (RD.Nick <> 'ADMINISTRADOR') then

BEGIN

if not (RD.Bloqueado) then

begin

if (RD.Estado) then

begin

Grid\_ListaJugadores.RowCount:= Grid\_ListaJugadores.RowCount + 1;

{Agrego renglon}

Grid\_ListaJugadores.Cells[0, Grid\_ListaJugadores.RowCount-1] := RD.Nick;

Grid\_ListaJugadores.Cells[1, Grid\_ListaJugadores.RowCount-1] := RD.Nombre;

Grid\_ListaJugadores.Cells[2, Grid\_ListaJugadores.RowCount-1] := RD.Apellido;

if RD.TipoJugador then

begin

Grid\_ListaJugadores.Cells[3, Grid\_ListaJugadores.RowCount-1] :='Real';

end

else

begin

Grid\_ListaJugadores.Cells[3, Grid\_ListaJugadores.RowCount-1] :='Ficticio';

end;

if RD.Bloqueado then

begin

Grid\_ListaJugadores.Cells[4, Grid\_ListaJugadores.RowCount-1] :='SI';

end

else

begin

Grid\_ListaJugadores.Cells[4, Grid\_ListaJugadores.RowCount-1] :='NO';

end;

Grid\_ListaJugadores.Cells[5, Grid\_ListaJugadores.RowCount-1] := datetimetostr(RD.UltimaConexion);

end;

end;

END;

Listado\_GeneralJugadores(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores));

Listado\_GeneralJugadores(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores));

end;

end;

{------------------------------------------------------------------------------}

//Obtengo el saldo que tiene el jugador mientras apuestas para mostrarlo. Se vera reflejado en la cuenta del jugador cuando

//es estado sea igual a 2 "no va mas".

Function TF\_Apostar.obtenerSaldoActualJugador(clave:tClave):tImporte;

begin

ME\_JUEGO.UltimaJugada(ME\_JUEGORULETA,RegJuegoRuleta);

SaldoAcumulado:= ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE,clave);

SaldoApuestas\_EnPartida:= ME\_APUESTAS.SaldoApuestasPartida(ME\_APUESTA,clave, RegJuegoRuleta.NroJugada);

obtenerSaldoActualJugador:= SaldoAcumulado - SaldoApuestas\_EnPartida;

end;

{------------------------------------------------------------------------------}

procedure TF\_Apostar.ListandoJugadores();

begin

Lib\_Auxiliar.limpiarGrid(Grid\_ListaJugadores);

Lib\_Auxiliar.EncabezadosComunJugadores(Grid\_ListaJugadores);

F\_Apostar.Grid\_ListaJugadores.RowCount:=1;

F\_Apostar.Listado\_GeneralJugadores(ME\_JUGADORES.Raiz(ME\_JUGADOR));

F\_Apostar.Grid\_ListaJugadores.FixedRows:=1;

end;

END.

### Formulario Ficticios

unit Form\_PanelFicticios;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Lib\_AuxJuego,

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Grids, Vcl.StdCtrls, Vcl.Buttons,

Vcl.ExtCtrls;

type

TF\_PanelFicticios = class(TForm)

Panel\_Listado: TPanel;

Panel\_grid: TPanel;

BitBtn1: TBitBtn;

Panel\_CrearFicticios: TPanel;

Lbl\_ActivarFicticios: TLabel;

Btn\_ActivarFicticios: TButton;

Cbox\_CantFicticios: TComboBox;

Panel\_FicticiosActivos: TPanel;

Label2: TLabel;

Grid\_FicticiosActivos: TStringGrid;

Panel2: TPanel;

Label1: TLabel;

Grid\_ListadoFicticios: TStringGrid;

procedure FormShow(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

procedure Grid\_ListadoFicticiosDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

Procedure Listado\_GeneralFicticios(RaizJugadores: Lib\_Auxiliar.tPos);

Procedure Listado\_FicticiosActivos(RaizJugadores: Lib\_Auxiliar.tPos);

procedure Cbox\_CantFicticiosSelect(Sender: TObject);

procedure mostrarFicticiosEnGrid();

procedure mostrarFicticiosActivosEnGrid();

procedure Btn\_ActivarFicticiosClick(Sender: TObject);

procedure Grid\_FicticiosActivosDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_PanelFicticios: TF\_PanelFicticios;

contador:tCantidad;

implementation

uses

Form\_Juego, Form\_Login, Form\_Apostar;

{$R \*.dfm}

{--------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.BitBtn1Click(Sender: TObject);

begin

Form\_Juego.F\_Juego.Panel\_JuegoGral.Enabled:=true;

F\_PanelFicticios.Close;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.FormClose(Sender: TObject; var Action: TCloseAction);

var

cantFictActivos: tCantidad;

begin

cantFictActivos:= 0;

F\_Juego.Enabled:=true;

totalFicticios:= ME\_JUGADORES.UltFicticio(ME\_JUGADOR);

F\_Juego.Lbl\_cantFicticios.Caption:= intTostr(totalFicticios);

Lib\_AuxJuego.Cant\_FicticiosActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR), cantFictActivos);

F\_Juego.Lbl\_CantFictActivados.Caption:= intTostr(cantFictActivos);

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.FormCreate(Sender: TObject);

begin

self.Position := poScreenCenter;

Lbl\_ActivarFicticios.Caption:= 'Seleccione ' + #10#13 + 'la cantidad ' + #10#13 + 'de ficticios ' + #10#13 + 'a activar: ';

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.FormShow(Sender: TObject);

var

i:integer;

begin

F\_Juego.Enabled:=false;

mostrarFicticiosEnGrid();

Panel\_CrearFicticios.Show;

Panel\_FicticiosActivos.Hide;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.Grid\_FicticiosActivosDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

begin

with (Sender as TStringGrid) do

begin

if (ARow = 0)

then

begin

Canvas.Brush.Color := clBtnFace;

Canvas.TextRect(Rect, Rect.Left + (Rect.Right - Rect.Left - Canvas.TextWidth(Grid\_FicticiosActivos.Cells[ACol,ARow]) + 1) div 2, Rect.Top + 2, Grid\_FicticiosActivos.Cells[ACol,ARow]);

end

else

begin

Canvas.Font.Color := clblack;

// Pinto el fondo segun elestado de la Atencion

if (ARow mod 2 = 0)

then

Canvas.Brush.Color := $00E1FFF9

else

Canvas.Brush.Color := $00FFEBDF;

Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top + 2, cells[acol, arow]);

Canvas.FrameRect(Rect);

end;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.Grid\_ListadoFicticiosDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

begin

with (Sender as TStringGrid) do

begin

if (ARow = 0)

then

begin

Canvas.Brush.Color := clBtnFace;

Canvas.TextRect(Rect, Rect.Left + (Rect.Right - Rect.Left - Canvas.TextWidth(Grid\_ListadoFicticios.Cells[ACol,ARow]) + 1) div 2, Rect.Top + 2, Grid\_ListadoFicticios.Cells[ACol,ARow]);

end

else

begin

Canvas.Font.Color := clblack;

// Pinto el fondo segun elestado de la Atencion

if (ARow mod 2 = 0)

then

Canvas.Brush.Color := $00E1FFF9

else

Canvas.Brush.Color := $00FFEBDF;

Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top + 2, cells[acol, arow]);

Canvas.FrameRect(Rect);

end;

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.Btn\_ActivarFicticiosClick(Sender: TObject);

var

cantFict\_Activar:tCantidad;

begin

if Cbox\_CantFicticios.ItemIndex <> Lib\_Auxiliar.PosNula then

begin

cantFict\_Activar:= strToint(Cbox\_CantFicticios.Text);

Lib\_AuxJuego.Activar\_JugadoresFicticios(Form\_Login.ME\_JUGADOR, cantFict\_Activar);

Panel\_FicticiosActivos.Show;

mostrarFicticiosActivosEnGrid();

Cbox\_CantFicticios.ItemIndex:= Lib\_Auxiliar.PosNula;

Panel\_CrearFicticios.Hide;

Form\_Juego.F\_Juego.Spbtn\_activarFicticios.Enabled:=false;

end

else

MessageDlg('Seleccione cantidad de jugadores a activar!',mtWarning , [mbOK], 0)

end;

{------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.Cbox\_CantFicticiosSelect(Sender: TObject);

var

op, cantCrear:integer;

numFicticio:tCantidad;

msjMostrar:string;

cantFict\_Activar, cantFict\_Activos:tCantidad;

begin

op:= strToint(Cbox\_CantFicticios.Text);

numFicticio:= ME\_JUGADORES.UltFicticio(ME\_JUGADOR);

if (op > numFicticio)then

begin

cantCrear:= op - numFicticio;

if (numFicticio <> 0) then

msjMostrar:= 'Solo hay ' + intTostr(numFicticio) + ' jugadores ficticios creados.' + #13 + #13 +'¿Quiere crear ' +intTostr(cantCrear)+ ' ficticios mas?'

else

msjMostrar:= 'No hay jugadores ficticios creados ¿Quiere crearlos?';

if Lib\_Auxiliar.Mensaje\_Confirmacion(msjMostrar) then

begin

Lib\_AuxJuego.Insertar\_JugadoresFicticios(ME\_JUGADOR, cantCrear);

mostrarFicticiosEnGrid();

cantFict\_Activar:= strToint(Cbox\_CantFicticios.Text);

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿CONFIRMA QUE QUIERE ACTIVAR '+intTostr(cantFict\_Activar)+' JUGADORES FICTICIOS?') then

begin

Lib\_AuxJuego.Activar\_JugadoresFicticios(Form\_Login.ME\_JUGADOR, cantFict\_Activar);

Panel\_FicticiosActivos.Show;

mostrarFicticiosActivosEnGrid();

Cbox\_CantFicticios.ItemIndex:= Lib\_Auxiliar.PosNula;

Panel\_CrearFicticios.Hide;

Form\_Juego.F\_Juego.Spbtn\_activarFicticios.Enabled:=false;

end;

end;

end;

end;

{<------------------------------------------------------------------------------}

Procedure TF\_PanelFicticios.Listado\_GeneralFicticios(RaizJugadores: Lib\_Auxiliar.tPos);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if not (RD.TipoJugador)then /// false: entonces es un ficticio

begin

Grid\_ListadoFicticios.RowCount:= Grid\_ListadoFicticios.RowCount + 1;

{Agrego renglon}

Grid\_ListadoFicticios.Cells[0, Grid\_ListadoFicticios.RowCount-1] := RD.Nick;

Grid\_ListadoFicticios.Cells[1, Grid\_ListadoFicticios.RowCount-1] := RD.Nombre;

Grid\_ListadoFicticios.Cells[2, Grid\_ListadoFicticios.RowCount-1] := RD.Apellido;

if RD.TipoJugador then

begin

Grid\_ListadoFicticios.Cells[3, Grid\_ListadoFicticios.RowCount-1] :='Real';

end

else

begin

Grid\_ListadoFicticios.Cells[3, Grid\_ListadoFicticios.RowCount-1] :='Ficticio';

end;

if RD.Estado then

begin

Grid\_ListadoFicticios.Cells[4, Grid\_ListadoFicticios.RowCount-1] :='ACTIVO';

end

else

begin

Grid\_ListadoFicticios.Cells[4, Grid\_ListadoFicticios.RowCount-1] :='NO ACTIVO';

end;

saldoActual:= F\_Apostar.obtenerSaldoActualJugador(RD.Nick);

Grid\_ListadoFicticios.Cells[5, Grid\_ListadoFicticios.RowCount-1] := intTostr(saldoActual);

Grid\_ListadoFicticios.Cells[6, Grid\_ListadoFicticios.RowCount-1] := datetimetostr(RD.Alta);

end;

Listado\_GeneralFicticios(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores));

Listado\_GeneralFicticios(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores));

end;

end;

{------------------------------------------------------------------------------}

Procedure TF\_PanelFicticios.Listado\_FicticiosActivos(RaizJugadores: Lib\_Auxiliar.tPos);

var

RD:Type\_JUGADOR.tRegDatos;

begin

if (RaizJugadores<>Lib\_Auxiliar.PosNula) then

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR,RD,RaizJugadores);

if not (RD.TipoJugador)then /// false: entonces es un ficticio

begin

if (RD.Estado) then ///true: entonces esta activo

begin

Grid\_FicticiosActivos.RowCount:= Grid\_FicticiosActivos.RowCount + 1;

{Agrego renglon}

Grid\_FicticiosActivos.Cells[0, Grid\_FicticiosActivos.RowCount-1] := RD.Nick;

Grid\_FicticiosActivos.Cells[1, Grid\_FicticiosActivos.RowCount-1] := RD.Nombre;

Grid\_FicticiosActivos.Cells[2, Grid\_FicticiosActivos.RowCount-1] := RD.Apellido;

if RD.TipoJugador then

begin

Grid\_FicticiosActivos.Cells[3, Grid\_FicticiosActivos.RowCount-1] :='Real';

end

else

begin

Grid\_FicticiosActivos.Cells[3, Grid\_FicticiosActivos.RowCount-1] :='Ficticio';

end;

if RD.Estado then

begin

Grid\_FicticiosActivos.Cells[4, Grid\_FicticiosActivos.RowCount-1] :='ACTIVO';

end

else

begin

Grid\_FicticiosActivos.Cells[4, Grid\_FicticiosActivos.RowCount-1] :='NO ACTIVO';

end;

saldoActual:= F\_Apostar.obtenerSaldoActualJugador(RD.Nick);

Grid\_FicticiosActivos.Cells[5, Grid\_FicticiosActivos.RowCount-1] := intTostr(saldoActual);

Grid\_FicticiosActivos.Cells[6, Grid\_FicticiosActivos.RowCount-1] := datetimetostr(RD.Alta);

end;

end;

Listado\_FicticiosActivos(ME\_JUGADORES.ProximoIzq(ME\_JUGADOR,RaizJugadores));

Listado\_FicticiosActivos(ME\_JUGADORES.ProximoDer(ME\_JUGADOR,RaizJugadores));

end;

end;

{------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.mostrarFicticiosActivosEnGrid();

begin

Lib\_Auxiliar.limpiarGrid(Grid\_FicticiosActivos);

Lib\_Auxiliar.EncabezadosComunFicticios(Grid\_FicticiosActivos);

Grid\_FicticiosActivos.RowCount:=1;

Listado\_FicticiosActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR));

Grid\_FicticiosActivos.FixedRows:=1;

end;

{------------------------------------------------------------------------------}

procedure TF\_PanelFicticios.mostrarFicticiosEnGrid();

var

numFicticio: Lib\_Auxiliar.tCantidad;

begin

Lib\_Auxiliar.limpiarGrid(Grid\_ListadoFicticios);

Lib\_Auxiliar.EncabezadosComunFicticios(Grid\_ListadoFicticios);

numFicticio:= ME\_JUGADORES.UltFicticio(ME\_JUGADOR);

if (numFicticio>0) then

begin

Grid\_ListadoFicticios.RowCount:=1;

Listado\_GeneralFicticios(ME\_JUGADORES.Raiz(ME\_JUGADOR));

Grid\_ListadoFicticios.FixedRows:=1;

end;

end;

END.

### Formulario Premios

unit Form\_Premios;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.StdCtrls, Vcl.Buttons, Vcl.Grids,

Vcl.ExtCtrls;

type

TF\_Premios = class(TForm)

Panel\_Premiados: TPanel;

Label7: TLabel;

BitBtn1: TBitBtn;

Grid\_premiados: TStringGrid;

procedure FormShow(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure Grid\_premiadosDrawCell(Sender: TObject; ACol, ARow: Integer;

Rect: TRect; State: TGridDrawState);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_Premios: TF\_Premios;

implementation

Uses

Form\_Juego, Form\_login;

{$R \*.dfm}

{--------------------------------------------------------------------------------}

procedure TF\_Premios.BitBtn1Click(Sender: TObject);

begin

Form\_Premios.F\_Premios.Close;

Form\_Juego.F\_Juego.Panel\_JuegoGral.Enabled:=true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Premios.FormClose(Sender: TObject; var Action: TCloseAction);

begin

Form\_Juego.F\_Juego.Panel\_JuegoGral.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Premios.FormCreate(Sender: TObject);

begin

self.Position := poScreenCenter;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Premios.FormShow(Sender: TObject);

begin

Form\_Juego.F\_Juego.Panel\_JuegoGral.Enabled:=false;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Premios.Grid\_premiadosDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

begin

with (Sender as TStringGrid) do

begin

if (ARow = 0)

then

begin

Canvas.Brush.Color := clBtnFace;

Canvas.TextRect(Rect, Rect.Left + (Rect.Right - Rect.Left - Canvas.TextWidth(Grid\_premiados.Cells[ACol,ARow]) + 1) div 2, Rect.Top + 2, Grid\_premiados.Cells[ACol,ARow]);

end

else

begin

Canvas.Font.Color := clblack;

if (ARow mod 2 = 0)

then

Canvas.Brush.Color := $00E1FFF9

else

Canvas.Brush.Color := $00FFEBDF;

Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top + 2, cells[acol, arow]);

Canvas.FrameRect(Rect);

end;

end;

end;

END.

### Formulario Listados

unit Form\_Listados;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Lib\_AuxJuego,

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ExtCtrls, Vcl.StdCtrls, Vcl.Buttons,

Vcl.Grids;

type

TF\_Listados = class(TForm)

Panel\_gralListados: TPanel;

BitBtn1: TBitBtn;

Grid\_ListadoxFiltro: TStringGrid;

Panel\_filtro: TPanel;

Lbl\_tittle: TLabel;

Panel\_Nomenclador: TPanel;

Lbl\_Nomenclador: TLabel;

Cbox\_Nomenclador: TComboBox;

Panel\_CantJugadores: TPanel;

Lbl\_cantJugadores: TLabel;

Cbox\_Cantidad: TComboBox;

Panel\_nickJugador: TPanel;

Edit\_NickJugador: TEdit;

Btn\_aceptar: TBitBtn;

Lbl\_NickJugador: TLabel;

Label1: TLabel;

Label2: TLabel;

Label3: TLabel;

Lbl\_NameJugador: TLabel;

Lbl\_FechaIngreso: TLabel;

Lbl\_PremiosAcumulados: TLabel;

procedure FormShow(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure Cbox\_CantidadSelect(Sender: TObject);

procedure Cbox\_NomencladorSelect(Sender: TObject);

procedure Btn\_aceptarClick(Sender: TObject);

procedure Grid\_ListadoxFiltroDrawCell(Sender: TObject; ACol, ARow: Integer;

Rect: TRect; State: TGridDrawState);

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_Listados: TF\_Listados;

opListado: integer;

implementation

uses

Form\_croupier, Form\_Jugadores, Form\_login;

{$R \*.dfm}

{--------------------------------------------------------------------------------}

procedure TF\_Listados.BitBtn1Click(Sender: TObject);

begin

F\_Listados.Close;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Listados.FormClose(Sender: TObject; var Action: TCloseAction);

begin

if AdminLog then

F\_Croupier.Enabled:= true

else

begin

F\_Jugadores.Enabled:= true;

Lib\_Auxiliar.limpiarGrid(Grid\_ListadoxFiltro);

if (OpListado = 2) then

Cbox\_Nomenclador.ItemIndex:= -1;

if (OpListado = 3) then

Cbox\_Cantidad.ItemIndex:= -1;

if (OpListado = 5) then

Edit\_NickJugador.Clear;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Listados.FormCreate(Sender: TObject);

begin

self.Position := poScreenCenter;

Lbl\_Nomenclador.Caption:= 'Seleccione nomenclador' + #10#13 + 'para filtrar.';

Lbl\_cantJugadores.Caption:= 'Seleccione cantidad' + #10#13 + 'para filtrar.';

Lbl\_NickJugador.Caption:= 'INGRESE NICK DE UN JUGADOR';

end;

{--------------------------------------------------------------------------------}

procedure TF\_Listados.FormShow(Sender: TObject);

begin

if AdminLog then

F\_Croupier.Enabled:= false

else

F\_Jugadores.Enabled:= false;

Lib\_Auxiliar.limpiarGrid(Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.RowCount:= 1;

Panel\_Nomenclador.Hide;

Panel\_CantJugadores.Hide;

Panel\_nickJugador.Hide;

Cbox\_Nomenclador.ItemIndex:= Lib\_Auxiliar.PosNula;

case (OpListado) of

1: begin //Listado General de Usuarios (jugadores)

Lbl\_tittle.Caption:='Listado general de jugadores';

Lib\_Auxiliar.EncabezadoDetalladoJugadores(Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ListadoGeneral\_JugadoresEnGrid(ME\_JUGADORES.Raiz(ME\_JUGADOR), Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.FixedRows:=1;

end;

2:begin ////jugadores que ganaron determinado premio seleccionando nomenclador

Lbl\_tittle.Caption:='Jugadores que ganaron determinado premio';

Panel\_Nomenclador.Show;

Cbox\_Nomenclador.ItemIndex:= Lib\_Auxiliar.PosNula;

Lib\_Auxiliar.EncabezadoGanadoresXPremio(Grid\_ListadoxFiltro);

end;

3:begin //<n> jugadores que más dinero ganaron a través del tiempo.

Lbl\_tittle.Caption:='Jugadores que más dinero ganaron a través del tiempo';

Panel\_CantJugadores.Show;

Cbox\_Cantidad.ItemIndex:= Lib\_Auxiliar.PosNula;

Lib\_Auxiliar.EncabezadoJugadoresMasGanaron(Grid\_ListadoxFiltro);

end;

4:begin //jugadores que nunca ganaron apuestas.

Lbl\_tittle.Caption:='Jugadores que nunca ganaron apuestas';

Lib\_Auxiliar.EncabezadoDetalladoJugadores(Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ListadoJugadoresNuncaGanaron(ME\_JUGADORES.Raiz(ME\_JUGADOR), Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.FixedRows:=1;

end;

5:begin //Ficha de un jugador. Ingresando nick podra listar su cta cte con todos sus movimientos ordenados por fecha y hora de manera cronologia ascendente

Lbl\_tittle.Caption:='Ficha de un Jugador';

Lib\_Auxiliar.EncabezadoFiltroCtaCte(Grid\_ListadoxFiltro);

Panel\_nickJugador.show;

Lbl\_NameJugador.Caption:= '';

Lbl\_FechaIngreso.Caption:= '';

Lbl\_PremiosAcumulados.Caption:= '';

Edit\_NickJugador.Clear;

Edit\_NickJugador.SetFocus;

end; //Listado general de todas las partidas jugadas hasta el momento

6:begin

Lbl\_tittle.Caption:='Listado general de partidas';

Lib\_Auxiliar.EncabezadoListadoPartidas(Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ListadoPartidas(Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.FixedRows:=1;

end;

7:begin //Listado de arbol balanceado de Jugadores

Lbl\_tittle.Caption:='Listado Árbol Binario AVL de Jugadores';

EncabezadoArbolJugadores(Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ListadoArbolJugadores(ME\_JUGADORES.Raiz(ME\_JUGADOR), Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.FixedRows:=1;

end;

8:begin //Listado de arbol balanceado de Ganadores

Lbl\_tittle.Caption:='Listado Árbol Trinario AVL de Ganadores';

Lib\_Auxiliar.EncabezadoArbolJugadores(Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ListadoArbolGanadores(ME\_GANADORES.Raiz(ME\_GANADOR), Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.FixedRows:=1;

end;

9:begin //Listado de jugadores que se encuentran activos (activos = logueados)

Lbl\_tittle.Caption:='Listado Jugadores Activos';

Lib\_Auxiliar.EncabezadoDetalladoJugadores(Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.Listado\_JugadoresActivos(ME\_JUGADORES.Raiz(ME\_JUGADOR), Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.FixedRows:=1;

end;

end;//case

end;

{--------------------------------------------------------------------------------}

procedure TF\_Listados.Btn\_aceptarClick(Sender: TObject);

var

Nick: tClave;

cantPremios: tCantidad;

Monto: tImporte;

begin

Nick:= UpperCase(Edit\_NickJugador.Text);

cantPremios:= 0;

Monto:= 0;

if not (ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR,PosJugador, Nick)) then

begin //Nick incorrecto

MessageDlg('Nick ingresado no existe!',mtError , [mbOK], 0);

Edit\_NickJugador.Clear;

Edit\_NickJugador.SetFocus;

end

else

begin

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regjugador, PosJugador);

Lbl\_NameJugador.Caption:= regjugador.Nick;

Lbl\_FechaIngreso.Caption:= DatetoStr(regJugador.Alta);

//Llamo a procedimiento recursivo que acumula en monto y cantidad los premios que gano

Lib\_AuxJuego.Premios\_AcumuladosXJugador(ME\_GANADORES.Raiz(ME\_GANADOR), Nick, cantPremios, Monto);

Lbl\_PremiosAcumulados.Caption:= intTostr(cantPremios) + ' PREMIOS POR ' +intTostr(monto);

//Llamo a procedimiento recursivo para que muestre los movimientos de ctacte de un jugador

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ListadoCtaCte\_deUnJugador(ME\_CTACTE.PrimeroLdParcial(ME\_CUENTACORRIENTE,Nick), Nick, Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.FixedRows:=1;

Edit\_NickJugador.Clear;

Edit\_NickJugador.SetFocus;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Listados.Grid\_ListadoxFiltroDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

begin

with (Sender as TStringGrid) do

begin

if (ARow = 0)

then

begin

Canvas.Brush.Color := clBtnFace;

Canvas.TextRect(Rect, Rect.Left + (Rect.Right - Rect.Left - Canvas.TextWidth(Grid\_ListadoxFiltro.Cells[ACol,ARow]) + 1) div 2, Rect.Top + 2, Grid\_ListadoxFiltro.Cells[ACol,ARow]);

end

else

begin

Canvas.Font.Color := clblack;

if (ARow mod 2 = 0)

then

Canvas.Brush.Color := $00E1FFF9

else

Canvas.Brush.Color := $00FFEBDF;

Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top + 2, cells[acol, arow]);

Canvas.FrameRect(Rect);

end;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Listados.Cbox\_CantidadSelect(Sender: TObject);

var

num:tCantidad;

importe: tImporte;

cantFilas, cantGanadores: integer;

begin

importe:= 0;

num:= strToint(Cbox\_Cantidad.Text);

cantFilas:= 1;

if not (ME\_GANADORES.MeVacio\_Ganadores(ME\_GANADOR)) then

begin

cantGanadores:= ME\_GANADORES.Cantidad\_Ganadores(ME\_GANADOR);

if (num<=cantGanadores) then

begin

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ListadoMasPremiados(ME\_GANADORES.Raiz(ME\_GANADOR), importe, Grid\_ListadoxFiltro, cantFilas);

Grid\_ListadoxFiltro.FixedRows:=1;

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ReordenarMasPremiados(Grid\_ListadoxFiltro, num, cantFilas);

Grid\_ListadoxFiltro.FixedRows:=1;

end

else

MessageDlg('TENGA PRESENTE QUE : ' +#13+#13 + 'La máxima cantidad a flitrar es de ' + intTostr(cantGanadores) + ' ganadores!', mtWarning, [mbOK], 0);

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_Listados.Cbox\_NomencladorSelect(Sender: TObject);

var

nomenclador: tNomenclador;

cantGanadores: tCantidad;

begin

nomenclador:= Cbox\_Nomenclador.Text;

cantGanadores:= 0;

Lib\_AuxJuego.CantGanadores\_xNomenclador(ME\_GANADORES.Raiz(ME\_GANADOR), nomenclador, cantGanadores);

if (cantGanadores>0) then

begin

Grid\_ListadoxFiltro.RowCount:=1;

Lib\_AuxJuego.ListadoPremiados\_Nomenclador(ME\_GANADORES.Raiz(ME\_GANADOR), nomenclador, Grid\_ListadoxFiltro);

Grid\_ListadoxFiltro.FixedRows:=1;

end

else

MessageDlg('NO HAY GANADORES : ' +#13+#13 + 'No hay ganadores con el nomenclador ' + nomenclador, mtInformation, [mbOK], 0);

end;

END.

### Formulario Premio/Bloqueo

//Formulario que segun eliga el croupier puede ser para bloquear/ desbloquear u obsequiar crédito

unit Form\_PanelCaseButtons;

interface

uses

ME\_JUGADORES,

Type\_JUGADOR,

ME\_CTACTE,

Type\_ALMACEN,

ME\_JUEGO,

Type\_JUEGO,

ME\_APUESTAS,

Type\_APUESTA,

ME\_GANADORES,

Type\_GANADOR,

Lib\_Auxiliar,

Lib\_AuxJuego,

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants, System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Grids, Vcl.StdCtrls, Vcl.Buttons,

Vcl.ExtCtrls;

type

TF\_PanelCaseButtons = class(TForm)

Panel\_Listado: TPanel;

Panel\_RegalarCredito: TPanel;

btn\_okRegalarCredito: TBitBtn;

Panel\_grid: TPanel;

Edit\_cantCredito: TEdit;

Lbl\_cantCredito: TLabel;

Lbl\_bloquearDesbloquear: TLabel;

Lbl\_regalarCred: TLabel;

Grid\_ListadoJugadores: TStringGrid;

Label1: TLabel;

Edit\_Jugador: TEdit;

BitBtn1: TBitBtn;

Panel\_LockUnlock: TPanel;

Label3: TLabel;

btn\_okLockUnlock: TBitBtn;

Edit\_NickJugador: TEdit;

procedure limpiarGrid();

procedure MostrarEnGridJugadores();

procedure FormCreate(Sender: TObject);

procedure BitBtn1Click(Sender: TObject);

procedure FormShow(Sender: TObject);

procedure Grid\_ListadoJugadoresDrawCell(Sender: TObject; ACol,

ARow: Integer; Rect: TRect; State: TGridDrawState);

procedure Grid\_ListadoJugadoresClick(Sender: TObject);

procedure btn\_okRegalarCreditoClick(Sender: TObject);

procedure Edit\_cantCreditoKeyPress(Sender: TObject; var Key: Char);

procedure btn\_okLockUnlockClick(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

private

{ Private declarations }

public

{ Public declarations }

end;

var

F\_PanelCaseButtons: TF\_PanelCaseButtons;

implementation

uses

Form\_login, Form\_Juego, Form\_Croupier;

{$R \*.dfm}

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.BitBtn1Click(Sender: TObject);

begin

Form\_Juego.F\_Juego.Panel\_JuegoGral.Enabled:=true;

F\_PanelCaseButtons.Close;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.FormClose(Sender: TObject;

var Action: TCloseAction);

begin

if not (F\_Juego.Enabled) then

F\_Juego.Enabled:= true;

if not (F\_Croupier.Enabled) then

F\_Croupier.Enabled:= true;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.FormCreate(Sender: TObject);

begin

self.Position := poScreenCenter;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.FormShow(Sender: TObject);

begin

if (F\_Juego.Enabled) then

F\_Juego.Enabled:= false;

if (F\_Croupier.Enabled) then

F\_Croupier.Enabled:= false;

MostrarEnGridJugadores();

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.Grid\_ListadoJugadoresClick(Sender: TObject);

begin

if Grid\_ListadoJugadores.Row <> 0

then

begin

if (Panel\_RegalarCredito.Enabled) then //Croupier va a regalar credito

begin

Edit\_Jugador.Text := Grid\_ListadoJugadores.Cells[2,Grid\_ListadoJugadores.Row];

if (Edit\_Jugador.Text<>'') then

begin

Edit\_cantCredito.Enabled:=true;

Edit\_cantCredito.SetFocus;

end

else

Edit\_cantCredito.Enabled:= false;

end

else //Croupier va a bloquear/ desbloquear

Edit\_NickJugador.Text := Grid\_ListadoJugadores.Cells[2,Grid\_ListadoJugadores.Row];

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.Grid\_ListadoJugadoresDrawCell(Sender: TObject;

ACol, ARow: Integer; Rect: TRect; State: TGridDrawState);

begin

with (Sender as TStringGrid) do

begin

if (ARow = 0)

then

begin

Canvas.Brush.Color := clBtnFace;

Canvas.TextRect(Rect, Rect.Left + (Rect.Right - Rect.Left - Canvas.TextWidth(Grid\_ListadoJugadores.Cells[ACol,ARow]) + 1) div 2, Rect.Top + 2, Grid\_ListadoJugadores.Cells[ACol,ARow]);

end

else

begin

Canvas.Font.Color := clblack;

if (ARow mod 2 = 0)

then

Canvas.Brush.Color := $00E1FFF9

else

Canvas.Brush.Color := $00FFEBDF;

Canvas.TextRect(Rect, Rect.Left + 2, Rect.Top + 2, cells[acol, arow]);

Canvas.FrameRect(Rect);

end;

end;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.limpiarGrid();

var

i:integer;

begin

with Grid\_ListadoJugadores do

for i := 0 to RowCount - 1 do

Rows[i].Clear;

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.btn\_okLockUnlockClick(Sender: TObject);

var

sMsj:string;

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR, posJugador, Edit\_NickJugador.Text);

ME\_JUGADORES.CapturarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

if (regJugador.Bloqueado) then

begin

sMsj:= 'desbloquear';

regJugador.Nick:= regJugador.Nick;

regJugador.Contrasenia:= regJugador.Contrasenia;

regJugador.Nombre:= regJugador.Nombre;

regJugador.Apellido:= regJugador.Apellido;

regJugador.Alta:= regJugador.Alta;

regJugador.UltimaConexion:= Now();

regJugador.Bloqueado:= false; // desbloqueo jugador

regJugador.Estado:= regJugador.Estado; // si lo desbloqueo, le doy el estado que ya tiene (desactivo)

end

else

begin

sMsj:= 'bloquear';

regJugador.Nick:= regJugador.Nick;

regJugador.Contrasenia:= regJugador.Contrasenia;

regJugador.Nombre:= regJugador.Nombre;

regJugador.Apellido:= regJugador.Apellido;

regJugador.Alta:= regJugador.Alta;

regJugador.UltimaConexion:= Now();

regJugador.Bloqueado:= true; // bloqueo jugador

regJugador.Estado:= false; // si lo bloqueo, lo desconecto

end;

if (Edit\_NickJugador.Text<>'') then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿Esta seguro que quiere ' +sMsj+ ' a ' +Edit\_NickJugador.Text + ' ?') then

begin

ME\_JUGADORES.BuscarInfoME\_Jugadores(ME\_JUGADOR, posJugador, Edit\_NickJugador.Text);

ME\_JUGADORES.ModificarInfoME\_Jugadores(ME\_JUGADOR, regJugador, posJugador);

//Elimino las apuestas efectuadas del jugador recientemente bloquado

EliminarApuestasUnJugador(Edit\_NickJugador.Text);

MostrarEnGridJugadores();

end;

end

else

MessageDlg('Debe seleccionar un jugador de la lista!!',mtWarning , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.btn\_okRegalarCreditoClick(Sender: TObject);

begin

if (Edit\_Jugador.Text<>'') then

begin

if (Edit\_cantCredito.Text<>'') then

begin

if (strtoint(Edit\_cantCredito.Text) >= ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA)) and (strtoint(Edit\_cantCredito.Text) <= ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA)) then

begin

if Lib\_Auxiliar.Mensaje\_Confirmacion('¿Seguro que quiere obsequiar $'+ Edit\_cantCredito.Text +' a ' + Edit\_Jugador.Text + ' ?') then

begin

regCtaCte.Nick:= Edit\_Jugador.Text;

regCtaCte.FechaHora:= Now();

regCtaCte.Concepto:= Type\_ALMACEN.tipoConceptos.Credito\_Regalado;

regCtaCte.Debe:= strToint(Edit\_cantCredito.Text);

regCtaCte.haber:= 0;

regCtaCte.saldo:= regCtaCte.Debe + ME\_CTACTE.SaldoAcumulado(ME\_CUENTACORRIENTE, Edit\_Jugador.Text);

ME\_CTACTE.InsertarAlmacen(ME\_CUENTACORRIENTE, RegCtaCte);

messagedlg('Crédito obsequiado con éxito!', mtInformation, [mbOk], 0);

Edit\_Jugador.Clear;

Edit\_cantCredito.Clear;

MostrarEnGridJugadores();

end;

end

else

begin

MessageDlg('El crédito minimo debe ser de $ ' +intTostr(ME\_JUEGO.ApuestaMinima(ME\_JUEGORULETA))+

', y el crédito máximo debe ser de $ ' +intTostr(ME\_JUEGO.ApuestaMaxima(ME\_JUEGORULETA)),mtWarning, [mbOK], 0);

Edit\_cantCredito.Clear;

Edit\_cantCredito.SetFocus;

end;

end

else

begin

MessageDlg('Debe ingresar crédito a obsequiar!',mtError , [mbOK], 0);

Edit\_cantCredito.SetFocus;

end;

end

else

MessageDlg('Debe seleccionar un jugador de la lista!',mtError , [mbOK], 0);

end;

{--------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.Edit\_cantCreditoKeyPress(Sender: TObject;

var Key: Char);

const

CARAC\_HABILITADOS = ['0'..'9', #0..#27];//solo numero y tecla borrar

begin

if not (Key IN CARAC\_HABILITADOS) then

Key:=#0; //la tecla vale null si presiona caracteres no habilitados

end;

{------------------------------------------------------------------------------}

procedure TF\_PanelCaseButtons.MostrarEnGridJugadores();

begin

F\_PanelCaseButtons.Show;

Lib\_Auxiliar.limpiarGrid(Grid\_ListadoJugadores);

Lib\_Auxiliar.EncabezadoDetalladoJugadores(Grid\_ListadoJugadores);

Grid\_ListadoJugadores.RowCount:=1;

Lib\_AuxJuego.ListadoGeneral\_JugadoresEnGrid(ME\_JUGADORES.Raiz(ME\_JUGADOR), Grid\_ListadoJugadores);

Grid\_ListadoJugadores.FixedRows:=1;

end;

END.