The Would-Be Gentleman

A simulation of social mobility set during the life and reign of Louis XIV of France, 1638 -1715

Faculty Author Development Program

Stanford University

User: Tom Maliska, FAD Program

Application: Edit

Document: Source Code:SUNKING/4.1/FINANCE.TEXT

Date: Tuesday, September 30, 1986

Time: 5:01:21 PM

Printer: LaserWriter Plus

```
{$M+}       {mac code}
{$X-}       {no automatic stack expansion}
{$R-}       {no range checking, paslib is buggy}

(*****************************************************************************************)
{The Would-Be Gentleman, Faculty Author Development Program at Stanford University.      }
{Version 4.1, Steve Fisher (version 1.0) 12/20/84 and Tom Maliska (versions to 4.1) 3/12/86, 9/15/86.  }
{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan, Steve Fisher,  }
{and Tom Maliska.                                                                          }
{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford Junior University.  }
{                                                                                          }
{This program simulates social mobility in 17th Century France during the reign of the Sun King  }
{Louis XIV.  It is used in a History seminar, The France of Louis XIV, at Stanford University by  }
{Professor Carolyn Lougee to teach economic aspects of social mobility.  The simulation allows students  }
{to act out the life of a French bourgeois from simple peasant to possible social and political success.  }
{Decision making is based upon a model of French economy first developed by Michael Carter in his  }
{simulation program ANCIEN REGIME and modified to suit the current simulation.             }
{                                                                                          }
{Virtually all procedures and functions use the variable ASSETS in some form to make calculations and  }
{change player status; in most cases, only explicit uses of the entire data structure ASSETS are recorded  }
{in the GLOBALS note.  Familiarity with ASSETS is the key to understanding action in the program.  }
(*****************************************************************************************)

program Finances;

(* This USES statement lists the Macintosh Libraries used by the program to compile Macintosh code.  *)

uses {$U-}
    {$U obj/MemTypes}     MemTypes,
    {$U obj/QuickDraw}    QuickDraw,
    {$U obj/OSIntf}       OSIntf,
    {$U obj/ToolIntf}     ToolIntf,
    {$U Obj/PackIntf}     PackIntf,
    {$U Obj/PasLibIntf}   PasLibIntf;

const  NUMICONS = 5;     (*These constants complement the resource file and procedures using dialogs*)
       BORDER = 3;
       ICONSIZE = 32;
       LANDICON = 1;
       RENTEICON = 2;
       OFFICEICON = 3;
       LEASEICON = 4;
       TEXTILEICON = 5;
       NONE = 0;
       ABOUTITEM = 1;
       INSTRITEM = 2;
       STATUSITEM = 1;
       BUYITEM = 2;
       SELLITEM = 3;
       MANAGEITEM = 4;
       NEXTITEM = 1;
       SAVEITEM = 3;
```

```
LOADITEM = 4;
QUITITEM = 6;
WEALTHITEM = 1;
PERSITEM = 2;
MARRITEM = 1;
PLANITEM = 2;
WILLITEM = 3;
NOBLEITEM = 4;
PROCITEM = 5;
HELPMENU = 1;
PROGRESSMENU = 2;
FINMENU = 3;
DECMENU = 4;
VIEWMENU = 5;
NUMMENUS = 5;
LEFTDIFF = 4;
BARDIFF = 15;


STARTAGE = 30;          (*These constants are used to set up the simulation*)
STARTYEAR = 1639;       (*Has to be one year high, see INITVARS, GoToNext, NextMarriage, *)
RANDDIVIDER = 328;      (*and NextOffice*)


BLIGHTRAND = 4;         (*These constants control land usage and grain yields in the simulation*)
POORRAND = 36;
GOODRAND = 68;
EXCELRAND = 100;
KINDRENT = 5;
BLIGHTYIELD = 0;
POORYIELD = 4;
GOODYIELD = 5;
EXCELYIELD = 6;
FBLIGHTPRICE = 15;
FPOORPRICE = 7;
FGOODPRICE = 5;
FEXCELPRICE = 4;
SBLIGHTPRICE = 25;
SPOORPRICE = 10;
SGOODPRICE = 6;
SEXCELPRICE = 4;
RENTVALUE = 29;
KINDITEM = 8;
CASHITEM = 9;
SHAREITEM = 10;
LANDVAL = 575;
MAXMISCLAND = 2400;
LANDTAX = 3;
GRAINLOSS = 20;
SEIGCOST = 700;
VICOMTECOST = 850;
MARQCOST = 1000;
```

```
SEIGITEM = 9;
VICOMTEITEM = 10;
MARQITEM = 11;
SEIGLOWER = 75;
SEIGUPPER = 150;
VICOMTELOWER = 300;
VICOMTEUPPER = 450;
MARQLOWER = 600;
MARQUPPER = 900;
SEIGRAD = 6;
VIC1RAD = 7;
VIC2RAD = 9;
MARQRAD = 10;
BASECOSTOFL = 600;
VCOSTOFL = 1500;
MCOSTOFL = 2000;


TEXTBUYITEM = 6;        (*These constants return the item number relating to the dialog *)
LANDBUYITEM = 6;        (*for the purchase or sale of investments  *)
RENTEBUYITEM = 6;
RENTESELLITEM = 6;
RENTEYSELLITEM = 6;
LANDSELLITEM = 9;
GRAINSELLITEM = 8;

BTEXT = 20;             (*These constants control the percent return on textiles*)
PTEXT = 1;
GTEXT = 9;
ETEXT = 18;
WEALTHDISP = 6;
PERSDISP = 7;

DEN2PERCENT = 2;        (*These constants relate to rentes*)
DEN7PERCENT = 13;
DEN14PERCENT = 20;
NOPAY = 2;
SOMEPAY = 35;
RENTE1RETURN = 71;      {Scaling factor for Denier 14 Rente Values.  See NextRente, BuyRente}
RENTE2RETURN = 55;      {Scaling factor for Denier 18 Rente Values.  See NextRente, BuyRente}
PARTPAY = 625;
FULLPAY = 1000;
MAXKRENTE = 400;


BRETURN = 10;           (*These constants relate to leases*)
PRETURN = 40;
GRETURN = 65;
ERETURN = 100;
LEASEFILE = 'Lease.Dat';
LEASEMIN = 5000;
```

```
OFFICIAL = 833;

BUYBTNITEM = 1;          (*These constants relate to the office investment dialogs*)
NEXTBTNITEM = 3;
PREVBTNITEM = 4;
BTNINACTIVE = 255;
BTNACTIVE = 0;
RAD1 = 7;
RAD2 = 8;
RAD3 = 9;
RAD4 = 10;
CHECKED = 1;
NOTCHECKED = 0;
NUMOFFPERSCREEN = 4;
OFFICEFILE = 'Office.Dat';
LEVYPERCENT = 3;
RAISEPERCENT = 7;
LEVYTAX = 20;
GLUTPERCENT = 80;
OFFDIVIDER = 4681;
ADDOFFPERCENT = 20;
SELLBTNITEM = 1;
RAISEPAY = 25;
RAISELEVY = 10;

INFOITEM = 3;            (*These constants relate to the Marriage dialog, kids, and family planning*)
COURTITEM = 1;
MARRAD1 = 6;
MARRAD2 = 7;
MARRAD3 = 8;
MARRAD4 = 9;
MARROFFSET = 5;
NUMMARRPERYEAR = 4;
MARRFILENAME = 'Marriage';
FIRSTGEN = 1;
SECONDGEN = 2;
MAXBRIDES = 15;
WAITYEARS = 3;
INELIGIBLE = 4;
TOOOLDFORKIDS = 38;

BIRTHOFFSET = 2;         (*These constants relate to generation switch, death dialog*)
DEATHPERCENT = 100;
OLDDEATHPERCENT = 130;
BEATDEATH = 20;
DEATHYEAR = 1676;
ENDYEAR = 1715;

OLDSONITEM = 6;          (*These constants relate to the will *)
OTHERITEM = 8;
DAUGHTITEM = 10;
KINITEM = 12;
```

```
NONKINITEM = 14;
CHARITYITEM = 16;
CHURCHITEM = 18;
NUMWILLCATEGORIES = 7;
KINMIN = 1;
NONKINMIN = 1;
CHURCHMIN = 5;
CHARMIN = 1;

NOBLECOST = 20000;      (*These constants relate to nobility, prestige, costofliving, and offices*)
SECYKING = 'secretary of the King';
PRESIDENT = 'president in the Chamber of Accounts of Paris';
CHANCELLOR = 'Chancellor';
SECOFSTATE = 'Secretary of State';
STARTPRESTIGE = 40;
AMBITIOUS = 20;
NOOFFPRESTIGE = 30;
COLMARRIAGEFACTOR = 40;
COLKIDSFACTOR = 20;
WILLPRESRATING = 4;
ENDFILE = 'Final Stats';

PRAD1 = 5;              {These constants relate to the protectors and choosing protectors}
PRAD2 = 6;
PRAD3 = 7;
PRAD4 = 8;
PRAD5 = 9;
NOPROCT = 5;
PROCTFAILMAX = 2;
PROCTYEARS = 2;
FOUQCASHMIN = 35000;
MAZCASHMIN = 15000;
PROCTREVOLTGIFT = 5000;
REVOLTPERCLOST = 97;
COLBOFFNUM = 2;
COLBPRESTIGE = 50;
MAINTPRESTIGE = 60;
DOBOFFNUM = 2;
DOBPRESTIGE = 50;
PARTLEASENUM = 3;

                        (*These constants control data file access and system information*)
SAVEFILE = 'Saved simulation';
BEEPDURATION = 3;
DRIVENUM = 1;
INSTRFILE = 'Instructions';
FIRSTBORN = -32768;
DEBTOR = -32767;


type IconRec = record
    IconHdl : Handle;
```

```
    IconRect : Rect;
end; (* IconRec *)

IconArray = array [1..NUMICONS] of IconRec;

IconType = record
    Defs : IconArray;
    Selected : NONE..NUMICONS;
    MenuDisabled,
    IconWasSelected : boolean;
    ChoiceRect : Rect;
end; (* IconType *)

Str13 = String [13];

MenuArray = array [1..NUMMENUS] of MenuHandle;

StrArray = packed array [1..255] of char;

ConvType = (SeigToVic, SeigToMarq, VicToMarq);

DateType = record
    Year : longint;
    Fall : boolean;
end; (* DateType *)

HarvestType = (Blight, Poor, Good, Excellent);

LandType = record
    Yield,
    Price,
    ShareCrop,
    Kind,
    Rent,
    Inherited,
    Bought,
    Seigneurie,
    Vicomte,
    Marquisat,
    Lost,
    Value : longint;
    Local,
    Regional : HarvestType;
end; (* Land *)

RentePtr = ^RenteType;

RenteHandle = ^RentePtr;

RenteType = record
    Year,
    CostDenier : longint;
```

```
    Fall : boolean;
    Next : RenteHandle;
end; (* RenteType *)

RenteRec = record
    FaceDenier,
    CostDenier,
    Return,
    GotThisYear,
    Owe,
    SoldVal,
    Payment : longint;
    IndivRentes : RenteHandle;
end; (* RenteRec *)

LeaseType = record
    FaceValue,
    Offer,
    OldOffer,
    NumBought,
    GotThisYear : longint;
    Title : Str255;
    Hanged,
    Bought : boolean;
end; (* LeaseType *)

DlogOffRec = record
    Title : Str255;
    Value,
    Prestige : longint;
    TitAndNob,
    Nobility : boolean;
end; (* DlogOffRec *)

OffArray = array [1..NUMOFFPERSCREEN] of DlogOffRec;

OfficePtr = ^OfficeRec;

OfficeHandle = ^OfficePtr;

OfficeRec = record
    Salary,
    AmtPaid,
    Value,
    Prestige : longint;
    Title : Str255;
    Next : OfficeHandle;
    Inherited : boolean;
end; (* OfficeRec *)

OfficeType = record
    TotPurchase,
```

```
      Number,
      Salary : longint;
     OfficeList : OfficeHandle;
      Levied,
      Raise : boolean;
end; (* OfficeType *)

BrideRec = record
      Name,
      Father : Str255;
      Age,
      Dowry,
      Group : longint;
end; (* Bride *)

BrideArray = array [1..NUMMARRPERYEAR] of BrideRec;

AvailRec = record
      IsAvail : boolean;
      Year : longint;
end; (* AvailRec *)

MarrType = record
     ThisYear : BrideArray;
     Bride : BrideRec;
      MarrBelow,
      Married : boolean;
      Failures : longint;
      Available : AvailRec;
end; (* MarrType *)

Kidhandle =  ^KidPtr;

KidPtr = ^KidRec;

KidRec = record
      Birth : DateType;
     Next : KidHandle;
end; (* KidRec *)

ChildRec = record
      Number,
      NumBoys,
      NumGirls : longint;
      Boys,
     Girls : KidHandle;
     NextBirth : DateType;
end; (* ChildRec *)

WillArray = array [1..NUMWILLCATEGORIES] of longint;

WillType = record
```

```
    Distribution : WillArray;
    Made,
    WasInAccord,
    InAccord : boolean;
end; (* WillType *)

ProctType = (Cornuel, Mazarin, Particelli, Conde, Fouquet, Colbert, Maintenon, GrandDauphin,
        DukeOfBurgundy, Generic, NoProtector);

ProctArray = array [1..5] of ProctType;

ProctRec = record
    Name : ProctType;
    YearFail,
    NumFailures,
    ThisProctFail : integer;
end; (* ProctRec *)

MailRec = record
    Content : Str255;
    Contact : ProctType;
    Year,
    Cash : longint;
    Fall : boolean;
end; (* MailRec *)

AssetsType = record
    Land : LandType;
    Rente : RenteRec;
    Lease : LeaseType;
    Office : OfficeType;
    Marriage : MarrType;
    Children : ChildRec;
    Will : WillType;
    Protector : ProctRec;
    Mail : MailRec;
    Grain,
    Cash,
    Textiles,
    TotalVal,
    CostOfLiving,
    Debt,
    Taxes,
    Age,
    Generation,
    TooAmbitious,
    OldPrestige,
    Prestige : longint;
    PresFallen,
    Quit,
    Won,
    Noble,
```

```
        ChoseProct,
        BoughtLetter,
        SoldGrain : boolean;
    end; (* AssetsType *)

var FinWindow, AssetWindow, WhichWindow : WindowPtr;
    FWRec, AWRec : WindowRecord;
    Icons : IconType;
    myEvent : EventRecord;
    Code, TopLine, MaxScroll, CorrRefNum, AppResFile, VRefNum : integer;
    Done, Temp : boolean;
    myMenus : MenuArray;
    Letter : char;
    Assets : AssetsType;
    Date : DateType;
    hTE : TEHandle;
    WatchHdl : CursHandle;
    Corrfile : STR255;

{$S}
(*****************************************************************************************)
procedure DebugDelay;
(* CALLED BY: DoPicture                                                                  *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure is used during debugging and pauses the program until the button is pressed.  *)

var ANEVENT : eventRecord;
    TEMP : boolean;

begin (* DebugDelay *)
    repeat
        SystemTask;          (*This allows for screen dumps and other system key interrupts*)
        TEMP := GetNextEvent(everyEvent, ANEVENT);
    until Button;
end; (* DebugDelay *)

(*****************************************************************************************)
procedure DlogManager(var Item : integer);

(* CALLED BY: SellYRente, SellKRente, SellRente, SellLand, BuyLease, BuyRente, BuyTextiles,      *)
(*        BuyMiscLand, BuyTitledLand MakeWill, ManageMLand, PlanFamily, BuyNobility, DispLetter  *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure is used to handle modal dialog boxes.      It repeatedly calls ModalDialog  *)
(* until the user presses either the OK or CANCEL buttons in the dialog box.  It returns (in ITEM) the  *)
(* item number of the button that was pushed.                                            *)

begin (* DlogManager *)
    repeat
        SystemTask;
        ModalDialog(nil, Item);
```

```
    until Item in [OK, Cancel];
end; (* DlogManager *)

(*****************************************************************************************)
procedure GetDText(TheDialog:DialogPtr; ItemNum:integer; var Str:Str255);

(* CALLED BY: SellYRente, Makewill, ProcessManage, BuyMiscLand, BuyTitledLand, buyTextiles,    *)
(* BuyRente, SellKRente, SellLand, Treasury                                                    *)
(* CALLS TO: none                                                                              *)
(* GLOBALS: none                                                                               *)
(* ACTION: This procedure gets the text from the EditText box that is item number ItemNum in dialog *)
(* box TheDialog.  It returns the text in Str.                                                 *)

var  DUMMYTYPE : integer;
    ITEMHDL : Handle;
    DUMMYRECT : Rect;

begin (* GetDText *)
    GetDItem (TheDialog, ItemNum, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    GetIText (ITEMHDL, Str);
end; (* GetDText *)


(*****************************************************************************************)
procedure LightBtn (TheDialog:DialogPtr; Item, Value:integer);

(* CALLED BY: DoConvCheck, InitConvert, DoCheck, BuyOffice, FewOffices, SellOffice, MarrCheck,  *)
(* DoMarriage, BuyNobility                                                                     *)
(* CALLS TO: none                                                                              *)
(* GLOBALS: none                                                                               *)
(* ACTION: This procedure HiLites the dialog button chosen by a mouseclick.                    *)

var  DUMMYTYPE : integer;
    ITEMHDL : Handle;
    DUMMYRECT : Rect;

begin (* LightBtn *)
    GetDItem(TheDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    HiliteControl(Pointer(ITEMHDL), Value);
end; (* LightBtn *)

(*****************************************************************************************)
procedure NumSpecs(Num:longint; var Len:integer; var STR:Str255);

(* CALLED BY: Break, NextLand, SellYRente, CheckDebt, NextMarriage, PutWillItems, EndSimulation, *)
(* DispOldManageValues, DoConvert, ConvertTitles, BuyMiscLand, BuytitledLand, BuyTextiles,     *)
(* BuyRente, AdvanceOffice, BuyOffice, BuyLease, SellKRente, SellAdvance, FewOffices, SellOffice, *)
(* SellLand, DoMarriage, BuyNobility                                                           *)
(* CALLS TO: none                                                                              *)
(* GLOBALS: none                                                                               *)
(* ACTION: This procedure returns the String representation of the number NUM in the variable STR. *)
(* The number of digits is returned through LEN.                                               *)
```

```
var  NEWNUM : String[1];

begin (* NumSpecs *)
    LEN := 0;
    STR := '';
    if Num = 0 then begin
        STR := ' ';
        STR[1] := '0';
        LEN := 1;
    end; (* If *)
    while Num <> 0 do begin
        LEN := LEN + 1;
        NEWNUM := ' ';
        NEWNUM[1] := chr(Num mod 10 + ord('0'));
        STR := Concat(NEWNUM, STR);
        Num := Num div 10;
    end; (* While *)
end; (* NumSpecs *)

(*****************************************************************************************)
function PowerOfTen(Num : integer) : longint;

(* CALLED BY: ConvertNum                                                                 *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This function returns ten to the NUM power.                                   *)

var  I, TEMP : longint;

begin (* PowerOfTen *)
    TEMP := 1;
    for I := 1 to Num do begin
        TEMP := TEMP * 10;
    end; (* For *)
    PowerOfTen := TEMP;
end; (* PowerOfTen *)

(*****************************************************************************************)
procedure ConvertNum(StrNum : Str255; var Num : longint; var ConvOK : boolean);

(* CALLED BY: SellYRente, MakeWill, ProcessManage, BuyMiscLand, BuyTitledLand, BuyTextiles,  *)
(* BuyRente, SellKRente, SellLand, Treasury                                                  *)
(* CALLS TO: none                                                                            *)
(* GLOBALS: none                                                                             *)
(* ACTION: This procedure converts the number in String representation (contained in STRNUM) to its  *)
(* numerical form through NUM. If STRNUM does not represent an unsigned longint, then CONVOK         *)
(* returns false.                                                                            *)

var  I, TEMP : longint;

begin (* ConvertNum *)
```

```
    TEMP := 0;
    if Length(StrNum) <> 0 then begin
       for I := 1 to Length(StrNum) do begin
           if not (StrNum[I] in ['0'..'9']) then begin
              ConvOK := false;
           end; (* If *)
          TEMP := TEMP + (ord(StrNum[I]) - ord('0')) * PowerOfTen(Length(StrNum) - I);
       end (* For *)
    end; (* If *)
    Num := TEMP;
end; (* ConvertNum *)

(*****************************************************************************************)
procedure Break(N : integer);

(* CALLED BY:  Not Called in release version 4.1                                         *)
(* CALLS TO: NumSpecs                                                                    *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure allows you to set numbered breakpoints for debugging.          *)

var LEN : integer;
   BRKNUM : str255;

begin (*Break*)
   NumSpecs(N, LEN, BRKNUM);
   ParamText(BRKNUM, '', '', '');
   Len := StopAlert(337,nil);
end;  (*Break*)

{$S DoPicture}


(*****************************************************************************************)
procedure DoPicture (PicNum : Integer);

(* CALLED BY: Bankrupt, Demographics, Main                                               *)
(* CALLS TO: SetDrawRect (local procedure)                                               *)
(* GLOBALS: AppResFile                                                                   *)
(* ACTION: This procedure reads a picture from a resource and fits it to the window display. *)

var  DrawWindow : windowptr;
    DRAWRECT, TEMPRECT : Rect;
    CopyPicHandle :  pichandle;
    WREC : WindowRecord;
    ExResFile : integer;

  {--------------------local procedure--SETDRAWRECT----------------}

{* CALLED BY: DoPicture                                                                 *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: Centers pictures smaller than the window rectangle in that rectangle          *)
```

```
(* GLOBALS AFFECTED OR USED:none                                    *)
(* CALLED BY: DoPicture                                             *)


  procedure SetDrawRect(WindowRect:rect;
                        OriginalPicRect:rect;
                        var DrawRect:rect);

  var
     WINDOSIZE:integer;
     PICSIZE:integer;

  begin
     WINDOSIZE := WindowRect.right - WindowRect.left;
     PICSIZE := OriginalPicRect.right - OriginalPicRect.left;
     if WINDOSIZE > PICSIZE then
     begin
        DrawRect.left := (WINDOSIZE - PICSIZE) div 2;
        DrawRect.right := DrawRect.left + PICSIZE;
     end else
     begin
        DrawRect.left := WindowRect.left;
        DrawRect.right := WindowRect.right;
     end; {else}

     WINDOSIZE := WindowRect.bottom  - WindowRect.top;
     PICSIZE := OriginalPicRect.bottom - OriginalPicRect.top;
     if WINDOSIZE > PICSIZE then
     begin
        DrawRect.top := (WINDOSIZE - PICSIZE) div 2;
        DrawRect.bottom := DrawRect.top + PICSIZE;
     end else
     begin
        DrawRect.top := WindowRect.top;
        DrawRect.bottom := WindowRect.bottom;
     end; {else}
  end; {procedure SetDrawRect}

begin  (*DoPicture*)

   DrawWindow := GetNewWindow(261, @WRec, Pointer(-1));
     SetPort(DrawWindow);
     PLSetWrPort (DrawWindow);

     HideCursor;

        {set up external resource file}
     ExResFile := OpenResFile('Pictures');
     UseResFile(ExResFile);


     CopyPicHandle := GetPicture(PicNum);
```

```
    TEMPRECT := CopyPicHandle^^.picframe;
    SetDrawRect(DrawWindow^.portRect,TEMPRECT,DRAWRECT);

     ClipRect(DRAWRECT);


     EraseRect(DRAWRECT);
     DrawPicture (CopyPicHandle, DRAWRECT);

     DebugDelay;

        {restore original resfile}
     UseResFile(AppResFile);
     CloseResFile(ExResFile);

  EraseRect(DrawWindow^.PortRect);

  CloseWindow(DrawWindow);
   ShowCursor;

end;   (* DoPicture *)


{$S Seg15}

(*****************************************************************************************)
procedure HidePCtl(ProctDialog:DialogPtr; Item:integer);

(* CALLED BY: SetUpProtector, Main                                                       *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure hides an item choice in a dialog window.                       *)

var  DUMMYTYPE : integer;
   ITEMHDL : Handle;
    DUMMYRECT : Rect;

begin (* HidePCtl *)
   GetDItem(ProctDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
   HideControl(Pointer(ITEMHDL));
end; (* HidePCtl *)

(*****************************************************************************************)
procedure SetUpProtector (ProctDialog:DialogPtr; Date:DateType; var Procts:ProctArray);

(* CALLED BY: ChooseProtector                                                            *)
(* CALLS TO:HidePCtl                                                                     *)
(* GLOBALS: Date                                                                         *)
(* ACTION: This procedure sets up the protector choices available in a given year.       *)

var  DUMMYTYPE : integer;
   ITEMHDL : Handle;
```

```
    DUMMYRECT : Rect;

begin (* SetUpProtector *)
    Procts[NOPROCT] := NoProtector;
    with Date do begin
        if Year < 1640 then begin
            ParamText ('Cornuel','Particelli',",");
            Procts[1] := Cornuel;
            Procts[2] := Particelli;
            HidePCtl(PROCTDIALOG, PRAD3);
            HidePCtl(PROCTDIALOG, PRAD4);
        end else begin
            if Year < 1648 then begin
                ParamText('Mazarin', 'Particelli', 'Condé', '');
                Procts[1] := Mazarin;
                Procts[2] := Particelli;
                Procts[3] := Conde;
                HidePCtl(PROCTDIALOG, PRAD4);
            end else begin
                if Year < 1654 then begin
                    ParamText('Mazarin', 'Condé', '', '');
                    Procts[1] := Mazarin;
                    Procts[2] := Conde;
                    HidePCtl(PROCTDIALOG, PRAD3);
                    HidePCtl(PROCTDIALOG, PRAD4);
                end else begin
                    if Year < 1656 then begin
                        ParamText('Mazarin', 'Condé', 'Fouquet', '');
                        Procts[1] := Mazarin;
                        Procts[2] := Conde;
                        Procts[3] := Fouquet;
                        HidePCtl(PROCTDIALOG, PRAD4);
                    end else begin
                    if Year < 1661 then begin
                    ParamText ('Fouquet', 'Colbert', 'Condé', 'Mazarin');
                            Procts[1] := Fouquet;
                            Procts[2] := Colbert;
                            Procts[3] := Conde;
                            Procts[4] := Mazarin;
                    end else begin
                        if (Year < 1661) or ((Year = 1661) and (not Date.Fall)) then begin
                            ParamText('Fouquet', 'Colbert', 'Condé', '');
                            Procts[1] := Fouquet;
                            Procts[2] := Colbert;
                            Procts[3] := Conde;
                            HidePCtl(PROCTDIALOG, PRAD4);
                        end else begin
                            if Year < 1680 then begin
                                ParamText('Colbert', 'Condé', '', '');
                                Procts[1] := Colbert;
                                Procts[2] := Conde;
                                HidePCtl(PROCTDIALOG, PRAD3);
```

```
                    HidePCtl(PROCTDIALOG, PRAD4);
                end else begin
                    if Year < 1683 then begin
                        ParamText('Colbert', 'Condé', 'Maintenon', '');
                        Procts[1] := Colbert;
                        Procts[2] := Conde;
                        Procts[3] := Maintenon;
                        HidePCtl(PROCTDIALOG, PRAD4);
                    end else begin
                    if Year < 1686 then begin
                        ParamText('Condé', 'Maintenon', 'Duke of Burgundy', 'Grand Dauphin');
                        Procts[1] := Conde;
                        Procts[2] := Maintenon;
                        Procts[3] := DukeOfBurgundy;
                        Procts[4] := GrandDauphin;
                    end else begin
                        if Year < 1711 then begin
                            ParamText('Duke of Burgundy', 'Maintenon', 'Grand Dauphin', '');
                            Procts[1] := DukeOfBurgundy;
                            Procts[2] := Maintenon;
                            Procts[3] := GrandDauphin;
                            HidePCtl(PROCTDIALOG, PRAD4);
                        end else begin
                            if Year < 1712 then begin
                                ParamText('Duke of Burgundy', 'Maintenon', '', '');
                                Procts[1] := DukeOfBurgundy;
                                Procts[2] := Maintenon;
                                HidePCtl(PROCTDIALOG, PRAD3);
                                HidePCtl(PROCTDIALOG, PRAD4);
                            end else begin
                                ParamText('Maintenon', '', '', '');
                                Procts[1] := Maintenon;
                                HidePCtl(PROCTDIALOG, PRAD2);
                                HidePCtl(PROCTDIALOG, PRAD3);
                                HidePCtl(PROCTDIALOG, PRAD4);
                            end; (* If *)
                        end; (* If *)
                    end; (* If *)
                end; (* If *)
                end; (* If *)
            end; (* If *)
        end; (* If *)
    end; (* If *)
    end; (* With *)
    GetDItem(ProctDialog, PRAD5, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    SetCtlVal(Pointer(ITEMHDL), CHECKED);
end; (* SetUpProtector *)

(*****************************************************************************)
```

```
procedure DoProctCheck(ProctDialog:DialogPtr; Item:integer);

(* CALLED BY: ChooseProtector                                                    *)
(* CALLS TO: none                                                               *)
(* GLOBALS: none                                                                *)
(* ACTION: This procedure checks the protector dialog for the choice made by the player.  *)

var  DUMMYTYPE : integer;
     ITEMHDL, RADHDL : Handle;
     DUMMYRECT : Rect;
     VAL : integer;

begin (* DoProctCheck *)
     GetDItem(ProctDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
     VAL := GetCtlValue(Pointer(ord(ITEMHDL)));
     if VAL = NOTCHECKED then begin
          VAL := CHECKED;
          GetDItem(ProctDialog, PRAD1, DUMMYTYPE, RADHDL, DUMMYRECT);
          SetCtlValue(Pointer(RADHDL), NOTCHECKED);
          GetDItem(ProctDialog, PRAD2, DUMMYTYPE, RADHDL, DUMMYRECT);
          SetCtlValue(Pointer(RADHDL), NOTCHECKED);
          GetDItem(ProctDialog, PRAD3, DUMMYTYPE, RADHDL, DUMMYRECT);
          SetCtlValue(Pointer(RADHDL), NOTCHECKED);
          GetDItem(ProctDialog, PRAD4, DUMMYTYPE, RADHDL, DUMMYRECT);
          SetCtlValue(Pointer(RADHDL), NOTCHECKED);
          GetDItem(ProctDialog, PRAD5, DUMMYTYPE, RADHDL, DUMMYRECT);
          SetCtlValue(Pointer(RADHDL), NOTCHECKED);
     end; (* If *)
     SetCtlValue(Pointer(ITEMHDL), VAL);
end; (* DoProctCheck *)

(*****************************************************************************)
procedure ExaProctItem(ProctDialog:DialogPtr; Item, ItemNum:integer; var NewProct:ProctType;
                       Procts:ProctArray);

(* CALLED BY: GetProct                                                          *)
(* CALLS TO: none                                                               *)
(* GLOBALS: none                                                                *)
(* ACTION: This procedure examines the dialog item returned by DoProctCheck and assigns the  *)
(* appropriate protector chosen into NewProct from the enumerated type declared above.       *)  *)

var  DUMMYTYPE, VAL : integer;
     ITEMHDL : Handle;
     DUMMYRECT : Rect;

begin (* ExaProctItem *)
     GetDItem(ProctDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
     VAL := GetCtlValue(Pointer(ITEMHDL));
     if VAL = CHECKED then begin
          NewProct := Procts[ItemNum];
     end; (* If *)
end; (* ExaProctItem *)
```

```
(*****************************************************************************************)
procedure GetProct(ProctDialog:DialogPtr; Procts:ProctArray; var NewProct:ProctType);

(* CALLED BY: ChooseProtector                                                            *)
(* CALLS TO: ExaProctItem                                                                *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure polls the dialog items for choices.                            *)

begin (* GetProct *)
   ExaProctItem(ProctDialog, PRAD1, 1, NewProct, Procts);
   ExaProctItem(ProctDialog, PRAD2, 2, NewProct, Procts);
   ExaProctItem(ProctDialog, PRAD3, 3, NewProct, Procts);
   ExaProctItem(ProctDialog, PRAD4, 4, NewProct, Procts);
   ExaProctItem(ProctDialog, PRAD5, 5, NewProct, Procts);
end; (* GetProct *)

(*****************************************************************************************)
procedure ChooseProtector(var Assets:AssetsType; Date:DateType);

(* CALLED BY: DoCommand                                                                  *)
(* CALLS TO: DoProctCheck, GetProct                                                      *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure sets up the dialog box and keeps track of failures, qualifications, etc.  *)

var  PROCTDIALOG : DialogPtr;
   ITEM : integer;
   PROCTS : ProctArray;
   NEWPROCT : ProctType;
   PROCTOK : boolean;

begin (* ChooseProtector *)
   if Date.Year < Assets.Protector.YearFail + PROCTYEARS then ITEM := StopAlert(312, nil)
   else
   if Assets.ChoseProct then ITEM := StopAlert(329, nil)
   else
   begin
    REPEAT
      PROCTDIALOG := GetNewDialog(277, nil, Pointer(-1));
      SetUpProtector(PROCTDIALOG, Date, PROCTS);
      repeat
          SystemTask;
          ModalDialog(nil, ITEM);
          if ITEM in [PRAD1, PRAD2, PRAD3, PRAD4, PRAD5] then begin
             DoProctCheck(PROCTDIALOG, ITEM);
          end; (* If *)
      until ITEM in [OK, Cancel];
      GetProct(PROCTDIALOG, PROCTS, NEWPROCT);
      PROCTOK := true;
      if ITEM = OK then begin
          with Assets do begin
```

```
case NEWPROCT of
      Particelli : if Lease.NumBought < PARTLEASENUM then begin
                        PROCTOK := false;
                   end; (* If *)
      Mazarin : if Cash < MAZCASHMIN then begin
                        PROCTOK := false;
                   end; (* If *)
      Conde : if Land.Seigneurie + Land.Vicomte + Land.Marquisat = NONE then begin
                        PROCTOK := false;
                   end; (* If *)
      Fouquet : if Cash < FOUQCASHMIN then begin
                        PROCTOK := false;
                   end; (* If *)
      Colbert : if (Office.Number < COLBOFFNUM) or (Prestige < COLBPRESTIGE) then begin
                        PROCTOK := false;
                   end; (* If *)
      Maintenon : if Prestige < MAINTPRESTIGE then begin
                        PROCTOK := false;
                   end; (* If *)
      DukeOfBurgundy : if (Office.Number < DOBOFFNUM) or (Prestige < DOBPRESTIGE) then begin
                        PROCTOK := false;
                   end; (* If *)
      GrandDauphin : if Land.Seigneurie + Land.Vicomte + Land.Marquisat = NONE then begin
                        PROCTOK := false;
                   end; (* If *)
end; (* Case *)
          if not PROCTOK then begin
              ITEM := StopAlert(311, nil);
              Protector.NumFailures := Protector.NumFailures + 1;
              Protector.ThisProctFail := Protector.ThisProctFail + 1;
             Prestige := Prestige - Protector.NumFailures;       {-1 on first, -2 on second}
              if Protector.ThisProctFail >= PROCTFAILMAX then begin
                  Protector.ThisProctFail := NONE;
                  Protector.Name := NoProtector;
                 Protector.YearFail := Date.Year;
                  PROCTOK := true;
                  ITEM := Stopalert(312, nil);
                  ITEM := Stopalert(318,nil);
             end; (* If *)
         end else begin
              Assets.ChoseProct := true;
             Protector.Name := NEWPROCT;
             If NEWPROCT <> NoProtector then begin
                 ITEM := StopAlert(317, nil);
                 Protector.ThisProctFail := NONE;
                 Protector.YearFail := NONE;
               end else begin
                 ITEM := StopAlert(318, nil);
             end; (* If *)
         end;  (* If *)
    end; (* With *)
end; (* If in OK *)
```

```
      DisposDialog(PROCTDIALOG);
    UNTIL PROCTOK;
   end; (* If *)
end; (* ChooseProtector *)

{$S Seg1}

(*****************************************************************************)
procedure GetIcons (var Icons:IconType);

(* CALLED BY: Initialize                                                     *)
(* CALLS TO: none                                                            *)
(* GLOBALS: Icons                                                            *)
(* ACTION: This procedure gets the icon definitions for the financial icons from the resource file and puts *)
(* them into the array in the Defs field of Icons. It then initializes the other fields of Icons and sets the *)
(* rectangles in which the icon is to be displayed. It is these rectangles that are shown when that *)
(* particular icon is selected.                                              *)

var  I : integer;

begin (* GetIcons *)
  for I := 1 to NUMICONS do begin
      Icons.Defs[I].IconHdl := GetIcon(255 + I);
      HNoPurge(Icons.Defs[I].IconHdl);
  end; (* For *)
  Icons.Selected := NONE;
  Icons.IconWasSelected := false;
  Icons.MenuDisabled := true;
  SetRect(Icons.ChoiceRect, 20, 158, 95, 173);
  SetRect(Icons.Defs[1].IconRect, ICONSIZE + 3 * BORDER, BORDER + 3 , 2 * ICONSIZE +
                3 * BORDER, BORDER + ICONSIZE + 3);
  SetRect(Icons.Defs[2].IconRect, BORDER, ICONSIZE + 3 * BORDER + 3, BORDER + ICONSIZE,
                        2 * ICONSIZE + 3 * BORDER + 3);
  SetRect(Icons.Defs[3].IconRect, ICONSIZE + 3 * BORDER, ICONSIZE + 3 * BORDER + 3,
                2 * ICONSIZE + 3 * BORDER, 2 * ICONSIZE + 3 * BORDER + 3);
  SetRect(Icons.Defs[4].IconRect, 2 * ICONSIZE + 5 * BORDER, ICONSIZE + 3 * BORDER + 3,
                3 * ICONSIZE + 5 * BORDER, 2 * ICONSIZE + 3 * BORDER + 3);
  SetRect(Icons.Defs[5].IconRect, ICONSIZE + 3 * BORDER, 2 * ICONSIZE + 5 * BORDER + 3,
                2 * ICONSIZE + 3 * BORDER, 3 * ICONSIZE + 5 * BORDER + 3);
end; (* GetIcons *)

(*****************************************************************************)
procedure DrawRectangles (FinWindow:WindowPtr);

(* CALLED BY: DrawIcons                                                      *)
(* CALLS TO: none                                                            *)
(* GLOBALS: FinWindow                                                        *)
(* ACTION: This procedure draws the rectangles that surround the icons when they are displayed. *)

var  IRECT : Rect;
     I : integer;
```

```
begin (* DrawRectangles *)
  SetPort(FinWindow);
  PLSetWrPort (FinWindow);
  SetRect(IRECT, ICONSIZE + 2 * BORDER + 1, 4, 2 * ICONSIZE + 4 * BORDER - 1,
                 ICONSIZE + 2 * BORDER + 2);
  FrameRect(IRECT);
  for I := 0 to 2 do begin
      SetRect(IRECT, 2 * I * BORDER + I * ICONSIZE + 1, 3 * BORDER + ICONSIZE + 1,
                (I + 1) * ICONSIZE + (I + 1) * 2 * BORDER - 1, 2 * ICONSIZE + 4 * BORDER + 2);
      FrameRect(IRECT);
  end; (* For *)
  SetRect(IRECT, ICONSIZE + 2 * BORDER + 1, 2 * ICONSIZE + 5 * BORDER + 1,
                 2 * ICONSIZE + 4 * BORDER - 1, 3 * ICONSIZE + 6 * BORDER + 2);
  FrameRect(IRECT);
end; (* DrawRectangles *)

(*****************************************************************************************)
procedure DrawIcons (Icons:IconType; FinWindow:WindowPtr);

(* CALLED BY: Initialize, LoadSimulation                                                 *)
(* CALLS TO: DrawRectangles                                                              *)
(* GLOBALS: FinWindow, Icons                                                             *)
(* ACTION: This procedure displays the financial icons on the screen surrounded by bordering rectangles.  *)
(* It also displays the fact that no investment has been selected.                       *)

var  IRECT : Rect;
     I : integer;

begin (* DrawIcons *)
    SetPort(FinWindow);
    PLSetWrPort(FinWindow);
    for I := 1 to NUMICONS do begin
       PlotIcon(Icons.Defs[I].IconRect, Icons.Defs[I].IconHdl);
    end; (* For *)
    DrawRectangles(FinWindow);
    MoveTo(23, 135);
    DrawString('Investment');
    MoveTo(21,150);
    DrawString('  Selected');
    MoveTo(30,170);
    FrameRect(Icons.ChoiceRect);
    if not Icons.IconWasSelected then begin
        DrawString('     None');
    end; (* If *)
end; (* DrawIcons *)

(*****************************************************************************************)
procedure SetUpMenus(var myMenus:MenuArray);

(* CALLED BY: Initialize                                                                 *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: myMenus                                                                      *)
```

```
(* ACTION: This procedure reads in the menus from the resource file and assigns them to the array      *)
(* myMenus. It then inserts them into the menu bar and displays the menu bar. Since no investment is  *)
(* initially selected, the investment menu is disabled.                                                *)

var  I : integer;

begin (* SetUpMenus *)
    for I := 1 to NUMMENUS do begin
        myMenus[I] := GetMenu(I);
        InsertMenu(myMenus[I], 0);
    end; (* For *)
    DisableItem(myMenus[FINMENU], 0);
    DrawMenuBar;
end; (* SetUpMenus *)

{$S}

(*****************************************************************************************)
function OffTooAmbitious(OfficeList:OfficeHandle) : boolean;

(* CALLED BY: CalcPrestige                                                                          *)
(* CALLS TO: none                                                                                   *)
(* GLOBALS: none                                                                                    *)
(* ACTION: This function determines whether or not the player has chosen an office that is not      *)
(* qualified for because only people with old families may buy those offices.                       *)

var  STOP : boolean;

begin (* OffTooAmbitious *)
    STOP := false;
    OffTooAmbitious := false;
    while (OfficeList <> nil) and (not STOP) do begin
        if OfficeList^^.Prestige = AMBITIOUS then begin
            OffTooAmbitious := true;
            STOP := true;
        end else begin
            OfficeList := OfficeList^^.Next;
        end; (* If *)
    end; (* While *)
end; (* OffTooAmbitious *)

(*****************************************************************************************)
procedure AddOffPrestige(var Second:longint; LookUp:longint; OfficeList:OfficeHandle);

(* CALLED BY: CalcPrestige                                                                          *)
(* CALLS TO: none                                                                                   *)
(* GLOBALS: none                                                                                    *)
(* ACTION: This procedure adds in the prestige rating for offices. The player gets an extra point for every  *)
(* office held that is at least rated at his prestige level, excepting the first office that is of that prestige.  *)

begin (* AddOffPrestige *)
    while OfficeList <> nil do begin
```

```
        if OfficeList^^.Prestige >= LookUp then begin
            SECOND := SECOND + 1;
        end; (* If *)
        OfficeList := OfficeList^^.Next;
    end; (* While *)
    SECOND := SECOND - 1;
end; (* AddOffPrestige *)

(*****************************************************************************************)
procedure CalcPrestige(var Assets:AssetsType);

(* CALLED BY: DisplayAssets                                                              *)
(* CALLS TO: OffTooAmbitious, AddOffPrestige                                             *)
(* GLOBALS: Assets                                                                       *)
(* ACTION: This procedure computes the player's prestige index. If the player has a prestige over 70 and  *)
(* buys a lease, the player can never again go into the 70's or beyond. If the player has no offices, then *)
(* he is put into the 30's. Otherwise he starts in the 40's. If he is noble and has a Seigneurie then he   *)
(* goes into the 50's. If he has a Vicomté or Marquisat he goes into the 60's. If he also has a very       *)
(* high office he can go into the 70's or 90's. He can never go into the 80's. He also gets one point for  *)
(* every 100 hectare purchased, and for every office his prestige or higher, except the first one. He also *)
(* gets a point if he's married, and loses two points for unsuccessful marriages or for trying to go into  *)
(* the 80's. If his father's will was not in accord with tradition, he loses 4 points, and he loses one    *)
(* point for every lease bought. These secondary points can never move the player into a higher category,  *)
(* but they can drop him into a lower one. The player, however, can never go below 30 unless he tries to  *)
(* go into the 80's, in which case he can go below 20.                                   *)

var  FIRST, SECOND, PROCTSHAMEPENALTY, TITLED : longint;

begin (* CalcPrestige *)
    with Assets do begin
        if (Prestige >= 70) and (Lease.Bought) then begin
            PresFallen := True;
        end; (* If *)
        {FIRST calculates the ten's digit of the Player's Prestige}
        if Office.OfficeList = nil then begin
            FIRST := NOOFFPRESTIGE;
        end else begin
            if OffTooAmbitious(Office.OfficeList) then begin
                FIRST := AMBITIOUS;
                if (Prestige div 10) <> (AMBITIOUS div 10) then begin
                    OldPrestige := Prestige;
                end; (* If *)
            end else begin
                FIRST := Office.OfficeList^^.Prestige;
            end; (* If *)
        end; (* If *)
        if (FIRST > STARTPRESTIGE) and ((not Noble) or ((Land.Vicomte = NONE) and
        (Land.Marquisat = NONE)) or (Lease.Bought)) then begin
            FIRST := STARTPRESTIGE;
        end; (* If *)
        if FIRST = STARTPRESTIGE then begin
            if (Noble) and (Land.Seigneurie > NONE) then begin
```

FAD Project Report

Final Summary

Date: 9/16/86

Program Title: The Would-Be Gentleman

Project Leader: Tom Maliska

Programmer(s): Steve Fisher, Tom Maliska

Professor: Dr. Carolyn Lougee

Course: History 31S, The France of Louis XIV

Development System:

Lisa 2/10 with Lisa Pascal Workshop 3.9

Delivery System:

Macintosh 128K, Macintosh 512K, one disk drive

Project Description:

Category: _X_simulation __tutorial __C.A.I.

Use: __ _primary instruction _X_adjunct

Briefly explain what the program does, who its intended users are, and how it helps these users:

The Would-Be Gentleman models the economic and social life of a French bourgeois during the life and reign of Louis XIV of France, 1638 - 1715. It is intended for undergraduates studying the economic life of the period. This scenario is briefly introduced to the player, who then embarks on an ambitious plan of economic and social decision-making. The player experiences the world of 17th century France by managing income and properties, planning marriages and estates, and seeking influence through official duties and alliances with powerful figures. Historical and personal events are inter-related in the simulation. Significantly, students develop an understanding of economic patterns and make their judgements in terms of 17th century France. As in real life, economic success and social prestige do not follow a fixed pattern. The program features many economic choices clearly laden with prestige, but players who overreach themselves can suffer shame, loss of income, and even, God forbid, bankruptcy. The challenge of *The Would-Be Gentleman* is to keep social status and economic status in balance.

CONTENTS OF THIS NOTEBOOK

*This material comes on Macintosh-formatted diskettes: (1) Source Code, Simulation, (2) Support Programs/Data Files, (3) Documentation and Manuals, (4) Pictures, and (5) *The Would-Be Gentleman* ver 4.1(a).

User: Tom Maliska, FAD Program

Application: MacDraw 1.9

Document: Cover, User's Manual

Date: Friday, October 3, 1986

Time: 6:07:53 PM

Printer: LaserWriter Plus

The Would-Be Gentleman User's Manual
version 4.1 (a)

Created at Stanford University by
the Faculty Author Development Program.

User: Tom Maliska, FAD Program

Application: MacWrite 4.5

Document: User's Manual

Date: Friday, October 3, 1986

Time: 5:33:44 PM

Printer: LaserWriter Plus

The Would-Be Gentleman

A simulation of social mobility set during the life and reign
of Louis XIV of France, 1638 -1715

User's Manual

The Would-Be Gentleman
User's Manual

Tom Maliska
Faculty Author Development Program
Stanford University
September 1986

This User's Manual is intended as a guide to the simulation *The Would-Be Gentleman*, version 4.1(a). It outlines use of the Macintosh computer and decision-making in the simulation, and gives the first-time player the background needed to understand the simulation. This manual is organized into the following sections:

About This Simulation

Purpose

The Would-Be Gentleman models the economic and social life of a French bourgeois during the life and reign of Louis XIV of France, 1638 - 1715. This scenario is briefly introduced to the player, who then embarks on an ambitious plan of economic and social decision-making. The player experiences the world of 17th century France by managing income and properties, planning marriages and estates, and seeking influence through official duties and alliances with powerful figures. Historical and personal events are inter-related in the simulation. Significantly, students develop an understanding of economic patterns and make their judgements in terms of 17th century France. As in real life, economic success and social prestige do not follow a fixed pattern. The program features many economic choices clearly laden with prestige, but players who overreach themselves can suffer shame, loss of income, and even, God forbid, bankruptcy. The challenge of *The Would-Be Gentleman* is to keep social status and economic status in balance.

Playing the Simulation

Insert the simulation diskette into the internal drive of the Macintosh. When you turn on the Macintosh, the simulation will begin automatically. It is advisable to copy the simulation as soon as possible and use only the copy. *The Would-Be Gentleman* is not copy protected.

You will see a window on the screen with a picture of a father and his son. This will go away after a few seconds, and the startup window will appear. This window describes the scenario to you. Click on the mouse button to close this window and start the simulation. You can begin playing by clicking on an investment icon. The usual strategy is to click on the investment icons in turn, using **Status** from the **Investments** menu to learn about the year's financial events.

Overview of Play

The simulation lasts from 1638 until 1715, exactly the lifespan of Louis XIV. Each year is divided into two six month periods marked by the seasons Fall and Spring. Each season, you can modify your investments and personal situation. In this way, you control your destiny in the simulation. Keep in mind the variety of things you can do each season. Among others, you can make a will, get married, view wealth or personal information, make investments, read instructions, and even save the game for later.

Review your investments during each season. To begin, click on the investment icon in the investment window. You can now use the Investment menu to make decisions about that investment. The types of investment available are Land, Rentes, Offices, Leases, and Textiles. It is important to know them, so review them all. Textiles are available in the Fall for one year investment; leases are available in the Spring for one year investment. Land, Offices, and Rentes are active during both Fall and Spring. See the section **Investment Decision-making** below for more details about investment planning.

All of the investments, except leases, allow you to change your mind on a purchase during the same season (i.e., until you give the **Next Interval** command on the **Progression** menu). Buying Leases and, in personal affairs, getting married are irrevocable, so don't purchase a lease or choose a bride unless you are certain of your commitment.

These are the first windows you will see when you start the simulation.

Startup Screen with copyright information.

THE WOULD-BE GENTLEMAN

HISTORY 318

PROF. CAROLYN LOUGEE

Created by the Faculty Author Development Program, Stanford University

The scenario for the simulation.

It is September 1638. The kingdom celebrates the birth of Louis le Dieudonné (the gift of God), first son of Louis XIII and Anne of Austria. Circumstances everywhere are not so joyous, however. In Normandy you, Denis Marin, have just lost your father, a bourgeois of Rouen. At age 30, you assume leadership of the Marin family, which for two generations has been moving away from its peasant origins toward the notability.

From your father you inherit an office of auditeur in the Chamber of Accounts of Rouen, which is valued at £21,000. You also inherit £5054 in cash and 42 hectares of cultivable land near Chateauvalion. Use these assets to the best of your ability. If you are shrewd and manage your assets well, you may increase your family's wealth and prestige during the lifetime of Louis le Dieudonné.

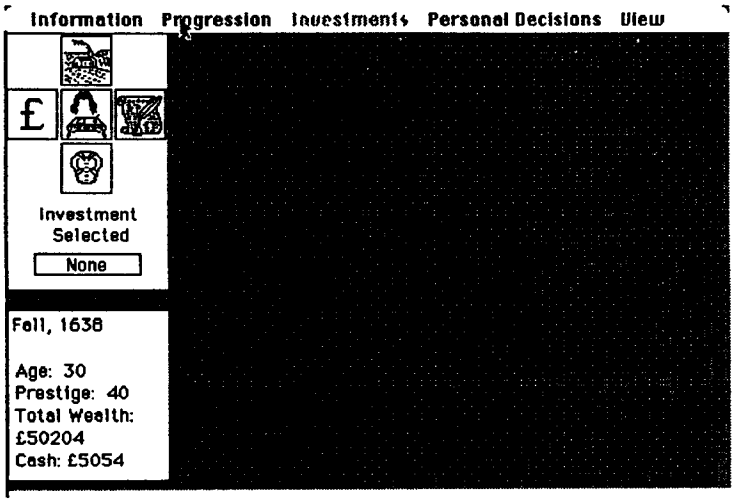
As you set out in the world to make your fortune, we wish you well, recalling the words penned two years ago by Pierre Corneille, friend and compatriot, for LE CID:

"A vaincre sans peril, on triomphe sans gloire."
"When there is no peril in the fight, there is no glory in the triumph."

{To start the simulation, press the button on the mouse.}

The simulation.

Note menus, investment icons, and status window.



Measuring Success (with an explanation of livres (£) and prestige points):

Success in the simulation is measured in wealth and prestige. Total wealth and accumulated prestige are always on display in the summary window. Wealth is a measure of financial success. The coin of the realm is the livre, or French pound (symbol = £). It is not important to know the historical derivation of the livre, only to think of it as a unit of money. To see your holdings in different investments, choose the **Wealth** item on the **View** menu.

The King's prestige is rated at 100; accordingly, your prestige is measured on a scale set from 0 to 99. Almost all events in the simulation affect your prestige. It will vary according to the investments and the personal decisions you've made, such as buying land, disposing of your wealth in a will, or success and failure in finding a protector. One of the challenges of *The Would-Be Gentleman* is to find investments and personal choices that bring prestige.

Investments in Brief

Investments are made using the icons in the investment window and the **Investments** menu. You can click on the investment icon and then select **Status** from the **Investments** menu. **Status** provides information about the market for the investment you selected and allows you to review your holdings in that form of investment. You can then select **Buy, Sell,** or **Manage** from the **Investments** menu.

All investments are seasonal. Land, and grain grown on it, can be bought or sold in any season. You can buy and sell rentes in any season, and earn half of an annuity on rente each season. Offices pay you an annual salary and are yours until you sell them or go bankrupt. You can buy leases in the Spring and earn a return on them in the following Spring. You can buy textiles in the Fall and earn a return the following Fall.

You will be paid immediately for rentes, offices, leases, and textiles when the associated investment annuity or salary is due. To find the return on these investments, click on an investment icon and check **Status** or choose **Wealth** on the **View** menu.

Land and grain must be sold using **Sell** on the **Investments** menu.

Read on for more information about investments, including information about financial calculations and units of measure for land, rentes, offices, leases, and textiles.

Land (with an explanation of quintels and hectares):

The Would-Be Gentleman models its financial decisions on the 17th century French economy. It is important to understand the conventions and units of that economy to learn from the simulation. Your family's fortunes are in your hands!

Land management relies on units of quintels and hectares. Quintels, like bushels in English measure, are volume measurements for dry goods. Hectares are measures of land area, equal to roughly 2.5 acres English measure. A harvest is measured in terms of the number of quintels per hectare, and a price per quintel, as the following tables show. The better the harvest, the less likely scarcity will drive the price higher before the next harvest.

The *regional* harvest determines the price of a quintel of grain.

Quality		Fall price	Spring price
blight	=	15£	25£
poor	=	7	9
good	=	5	6
excellent	=	4	4

The *local* harvest determines the yield in quintels of grain per hectare of land. This is the amount of grain that you receive for each hectare cultivated by sharecropping or renting in kind.

Quality		Amount of grain
blight	=	0 quintels per hectare
poor	=	4 " " "
good	=	5 " " "
excellent	=	6 " " "

The profit you obtain by selling grain is determined by comparing the relation between the local and the regional harvest. In general, when the local harvest is better than the regional harvest, you will have larger amounts of grain which, at a higher price, bring in more profits.

You can manage land for the harvest in three ways: Renting in Kind, Renting for Cash, and Sharecropping. These choices vary in the stability and range of their returns. Renting in Kind returns grain at a rate of 5 quintels per hectare and this yield is paid regardless of the quality of the local harvest; the value of the grain is dependent on the price determined by the regional harvest.

Renting for Cash pays you £29 per hectare and, as the least speculative form of management, it is paid regardless of the quality of the harvests. Profits from sharecropping equal the price of grain at the time of sale multiplied by the yield of grain in the harvest, and are thus dependent on both local and regional harvests.

Grain is lost to rats, fungus, rot, and damp storage. The amount is shown in **Status** each Fall.

Peasant revolts can occur in the transition from Fall to Spring. If a sufficient quantity of grain is not sold in the Fall to provide for winter food, then some grain holdings can be lost to a revolt.

Titled Land (with an explanation of titled lands of types Siegneurie, Vicomté, Marquisat):

Titled land may be purchased using the **Buy** command on the **Investments** menu and clicking on the **Titled** button in the window that appears. Another window will appear listing the types of titled land and their prices. Titled land is more expensive than miscellaneous land and must be bought in large quantities. It is included in the management of land for the harvest.

Titled land is land set aside for estates owned by noblemen and wealthy merchants. It carries a title for the owner and it is a sign of great prestige for the owner.

Once purchased, titled land (as with inherited miscellaneous land) cannot be resold.

Land Tax:
If you are not a nobleman, taxes will be assessed in the Spring at 3 livres per hectare for all of your holdings in miscellaneous and titled lands.

Rente (with an explanation of denier):

Rente is the term used for a form of loan of £1000. Rentes are made either to the King (King's rente) or borrowed for personal use (Personal rente). Rentes can be treated as a speculative investment with the strategy "buy King's rente at a low price and sell at a high price" (King's rente), as a means of acquiring cash (Personal rente), or as a steady investment income (annuity, or interest returned on King's rente). Denier is a measure of a rente's market value or price; one denier is 1/14 of the value of one rente, or 1/14 x £1000. There are five market values that occur with different frequencies during the simulation, and these values can change in any season. The base price at the start of the simulation is denier 14, but rentes can be devalued to a base price of denier 18. At this price, one denier is 1/18 of the value of one rente, or 1/18 x £1000.

The following table outlines denier and associated prices in livres.

Denier or market price	At denier 14, price of one rente in livres	At denier 18, price of one rente in livres
2	£143	£111
7	£500	£389
11	£786	£611
14	£1000	£778
18	n/a	£1000

King's rente:

You can buy up to 400 King's rentes at a given time. The King's rente can be resold for cash. The market price when you sell the rente compared to the market price for which you bought it determines profit or loss.

Personal rente:

Personal rentes are loans you can take. In some situations you might lose enough money on investments to be forced to pay with a personal rente. You can borrow cash in any quantity, but you must pay interest at the market price at the time that the loan is made. If a low price is in effect when you sell a personal rente, you'll pay interest at a higher rate than you would for a loan obtained at the full price of denier 14. At denier 2, for instance, you'll pay back seven times the interest rate of a loan made at the denier 14. Choose **Status** on the **Investments** menu or **Wealth** on the **View** menu to get information about payments on personal rente sold.

Annuity:

The King's rentes also return an annuity in cash each year they are held, equal to one denier per rente purchased. At rente equal to denier 14, one denier equals 1/14 of £1000, or £71; at rente equal to denier 18, one denier equals 1/18 of £1000, or £55. Payment of the annuity may be voided or reduced based on the King's decree. This can be a great boon or a catastrophe financially, depending on how much cash you have invested in rentes and when you made the investment. The following table shows the payment, the frequency of that payment, and the annuity paid for each rente or £1000 loaned.

Payment (as % of annuity)	Return in £ at denier 14	Return in £ at denier 18 (after 1660)
0	0	0
62.5	44	34
100	71 (full annuity)	55 (full annuity)

Special cases:
Personal rentes always result in a cash return at the market rate. You are essentially giving a note for £1000 in return for the cash value of the rente at current denier. These rentes cannot be reclaimed and interest on the loan will be assessed each year as an expense.

Some protectors provide warning signals about the price of rentes.
Some protectors do not allow you to purchase the King's rentes.

Offices:

Offices pay a salary each year. Your salary can be seen by choosing **Wealth** on the **View** menu. Individual office salaries can be inspected by clicking on the **Office** icon, then selecting **Buy** from the **Investments** menu.

Leases:

Leases, like offices, are purchased from the government. You can purchase a lease in Spring, and collect a fee the following Spring. Leases are speculative, in that you buy them at a discount and reap large profits or losses on your purchase. Select **Status** on the **Investments** menu in Spring to find out what the profit or loss is on a lease.

Textiles:

Textiles return cash to you each Fall on investments from the previous Fall. The amount of the return is given in a percentage of the investment in the status window for the Textiles investment. Click on the Textiles icon, then select **Status** on the **Investments** menu. Your return on investments in Textiles depends upon the health of the economy of the region, which is directly related to the quality of the harvest.

Personal Decision-making

Marriage

Choose **Marriage** from the **Personal** menu to consider a marriage. It is important to marry in the simulation, since the time span of the simulation covers two generations. The menu choice displays a window with a list of brides. For each bride, there is a button to select her. Use the **Info** button to get information about her age, her father, his office, and her dowry. Use the **Court** button to attempt a courtship. The simulation keeps track of success and failure. Best wishes!

Family Planning

Having and supporting children is an important task in the simulation. In the first generation, a successful marriage to the right bride will always produce an heir. In the second generation, the player has to use the **Family Planning** choice on the **Personal** menu to plan for children. The firstborn son is the inheritor of the player's wealth, and the will determines how much of the father's wealth is passed on (see the section following named **Make a Will**).

Generations mark progress in the simulation. Control of the family fortune is passed to the eldest son. In the first generation, Denis Marin takes up the reins of the Marin family at the age of 30. At the time of his death, the eldest son Jean-François Marin takes on the responsibility of the family fortune. Hyacinth-Florent Marin de Montville carries on the family name in the third generation, at which point the simulation ends. Your skill at managing the Marin family fortune will determine the status Denis, Jean-François, and Hyacinth-Florent will achieve in their lifetimes.

Make a Will

A will allows the player to pass an inheritance on to family and other beneficiaries. Use the **Make a Will** choice on the **Personal** menu to leave percentages of wealth to the eldest son, other sons, other daughters, other kin, non-kin, charity, and the Church. The simulation will tell you if the will is in accord with tradition, and, if not, allow you to try a different distribution of wealth. You can keep a will that is not in accord with tradition, but it will cost you prestige!

Buy a Letter of Nobility

A Letter of Nobility costs £20,000 and, in combination with titled land, gives a higher prestige rating. Higher prestige is important for marriages and can have an influence on protectors; it can also mean a higher cost of living, so one must be careful to plan for this purchase.

The Letter of Nobility confers two additional prestige points on the buyer. You should see the prestige added right after buying the Letter of Nobility.

Choose a Protector

When the game begins, you inherit a friend at court named M. Cornuel. At his death, no one is left to protect your interests. It is advisable to seek admission to the circle of associates, or coterie, of a powerful courtier, financier, or nobleman. The **Choose Protector** item on the **Personal** menu allows you to consider some of the leading figures of the time and seek admission to their coterie. Requirements for acceptance differ with each protector, and the protectors change throughout the simulation. Protectors offer financial benefits when you belong to their coterie, and each keeps you informed of events at the palace through correspondence. Choose well, and try not to overreach yourself, because rejection by these powerful protectors can cost you both prestige and money!

Protectors and Correspondence:
Correspondence is a means for your protector and others to communicate with you. These messages can bring good news or bad tidings. Correspondence can include cash benefits or tell you of costs related to social events and political tides. The correspondence you see will vary with different protectors, so don't expect to see the same messages each time you run the simulation.

During a season, you'll be able to read each mail message one time only. When you click on the **Ready** button, the message disappears.

Hints for using the Simulation

A Quick Review of the Menus

Information	Use this menu to get background on the simulation.
About the Authors	View author and copyright information.
Instructions	Get hints on using *The Would-Be Gentleman.*
Progression	Use this menu to control progress in the simulation.
Next Interval	Move the game ahead one turn, or six months, and calculate the outcome of financial decisions.
Save Game	Save current progress for later use.
Restore Game	Restore the last simulation saved.
Quit	End the simulation and eject the diskette.
Investments	Use this menu to make investment decision
Status	View holdings in the current investment.
Buy	Make an investment.
Sell	Sell Land, Grain, or Offices for cash.
Manage	Allocate miscellaneous land for the harvest or convert lesser titled lands into greater titled lands.
Personal	Use this menu to manage affairs for your family.
Marriage	Observe and court the brides of the season.
Family Planning	In the second generation, plan for a new birth.
Make a Will	Plan inheritance of your fortune.
Buy a Letter of Nobility	Become a nobleman.
Choose a Protector	Seek the protection of powerful figures in government.
View	Use this menu to examine your socioeconomic status.
Wealth	View the state of your finances.
Personal Information	View the state of your personal affairs.

Instructions for The Would-Be Gentleman

A brief overview of *The Would-Be Gentleman* can be reviewed at any time during play by selecting **Instructions** from the **Information** menu. A window appears with an overview of *The Would-Be Gentleman.* The User's Manual contains more explicit information about using the simulation effectively.

Using the Game with Different Macintoshes (MFS vs. HFS)

The Would-Be Gentleman comes on a 400K diskette formatted in the Macintosh File System (MFS). It is ready to run on any Macintosh model. Those who use the simulation diskette on Macintosh-512K E and -Plus computers can copy the **Louis XIV** application and **Game Files** folder to any 800K diskette or hard disk drive with System 2.3 and Finder 5.3 installed. If the Hierarchical File System (HFS) is used on the 800K diskette or hard disk, then the application **Louis XIV** must be placed in the **Game Files** folder holding the simulation data files. The simulation data files must be removed from their separate folders and placed in the **Game Files** folder.

Preparing a Backup of the Simulation Diskette

Copy the simulation diskette to a newly formatted, blank diskette.

Set *The Would-Be Gentleman* simulation diskette aside for a moment. Start your Macintosh with a System diskette. In the external drive, insert a new diskette and initialize it (or choose **Erase diskette** from the **Special** menu to re-use an old diskette). Name that diskette *Backup of The Would-Be Gentleman.* When the icon for *Backup...* appears on the desktop, click the mouse on the System diskette and eject it from the internal drive. Insert the original simulation diskette in the internal drive, and when it appears on the desktop, drag its icon on top of *Backup of The Would-Be Gentleman,* the newly formatted diskette icon. This begins the diskette copy procedure. When all the files have been copied to the newly formatted diskette, select **Shut Down** from the **Special** menu and label the new diskette *Backup of The Would-Be Gentleman.*

Use only the copy of the simulation and put the original in a safe place, away from heat, magnetic fields, liquids, and dirt. You can make copies for your own use, but please do not distribute the simulaton to others. Refer to the Academic Courseware Exchange catalog available at Kinko's copy centers for information about new versions of *The Would-Be Gentleman.*

Unlock the Simulation Diskette

Make sure the simulation diskette is unlocked before use. If you forget to unlock the diskette, you will be unable to successfully save and restore the simulation you are working on. If this happens while you're working on the simulation, eject the diskette with a straightened paperclip in the small hole next to the diskette opening or press the Command-Shift-1 keys. Check the tab in the upper right corner of the diskette. This tab protects the diskette from changes when it is in the locked or open position. Make sure it is unlocked or closed, and reinsert the diskette in the Macintosh.

Using the Macintosh

This section is an introduction to the Macintosh mouse and windows.

The Mouse Pointer

The mouse is a pointer to things on the Macintosh screen. At different times, the mouse pointer changes shape. Most of the time it looks like an *arrow*. Use this pointer with menus, buttons, and windows that appear on the screen. You move the mouse to point at something, and press the mouse button to see the result.

Sometimes the pointer turns into a *watch*. This just means that the computer is working and you need to wait a few seconds. Don't be alarmed when this happens! As soon as the arrow returns, continue to mouse around.

Clicking and Dragging on Menus

Take some time to get used to the mouse. It generally acts as a pointer for you. The mouse button is used ("clicked" or "pressed") to signify your choice in a command situation, or to select menu items ("dragging," or moving the mouse with the button down).

The most important use of the mouse is on the *menu bar*. You select items from the menu by *pointing* at the menu bar, *clicking* on the menu name, *dragging* through the menu items, and you activate commands by releasing the mouse button on the appropriate item. To see a menu, point the mouse at the menu bar (the white area at the top of the screen with the menu names **Information, Progression, Investments, Personal,** and **View**) and click the mouse button. The menu under the pointer will "pull down" to show you menu items. Each item represents a command that you can give to the simulation.

Drag the mouse down the menu (don't let the mouse button up yet!). Menus, and commands in them, darken to show you when they are selected. If you release the mouse button when an item is darkened, the Macintosh will execute that command.

You can leave a menu without choosing a command by dragging the pointer away from the list of menu items and letting go of the mouse button.

About Windows

Windows are your way of communicating choices to the program, such as making investments. Windows in this simulation also allow you to get information about your investments and life.

Making Decisions

In some windows you make decisions simply by clicking the mouse in specified areas and typing in your responses. The **Tab** key will move you from one information field to another. The **Enter** key should be used carefully, as it often means you are finished making commands (see the description of the **OK** button in the next section).

Leaving a Window

Windows that offer choices have a **CANCEL** button. Clicking the mouse here allows you to exit a window without making choices. If you do make decisions, the **OK** button or the **Enter** key puts them into action for you. Most windows that give information have only an **OK** option or simply disappear when you click the mouse button. This allows you to exit a window without making changes or reading everything in that window. Correspondence, or mail messages, appear in a window with a message "This is the only time you will be able to read this message" and a **Ready** button.

If a window appears without a **CANCEL, OK,** or **Ready** button, read the information on the window and click on the mouse button to make it disappear.

Areas of the Screen

There are four visual areas in the simulation: the menu bar, the information window, the investment window, and the player summary window.

The *menu bar* is located at the top of the screen. Menus, as discussed earlier, are used to give commands to the simulation.

The text window that precedes the simulation is an example of an *Information Window*. During the simulation, other information windows appear to display mail, investment information, and other results of your decison-making. You can ask for different types of information using the **Status** choice on the **Investments** menu or any choice on the **View** menu. You can choose **Instructions** from the **Information** menu to display an overview of the simulation.

The *Investment Window* has icons for your various investment opportunities. These icons represent types of investment that work with commands (**Status, Buy, Sell,** or **Manage**) from the **Investments** menu.

Lastly, you have a *Player Summary Window* that displays the current date, your age, prestige, total wealth, and free (uninvested) cash. This window is updated by the computer each time you make an investment decision, and serves as a quick look at how you're doing.

User: Tom Maliska, FAD Program

Application: MacWrite 4.5

Document: Programmer's Manual

Date: Thursday, October 2, 1986

Time: 6:36:25 PM

Printer: LaserWriter Plus

The Would-Be Gentleman
Programmer's Manual

Tom Maliska
Faculty Author Development Program
Stanford University
September 1986

This document contains technical information about *The Would-Be Gentleman*, organized into the following sections:

Program Information

The Would-Be Gentleman was developed in Lisa Pascal using Workshop 3.9. It consists of two parts in the Lisa code: **Sunking/4.1/Finance**, which is the Pascal program, and **Sunking/4.1/Financer**, which is the resource file containing information about icons, dialogs, alerts, and windows. The compiled code is renamed **Louis XIV** and has a SunKing icon on the Macintosh diskette. Descriptions of the coding and data structures follow in the sections **List of Procedures** and **Description of Segments.**

A number of program and data files are required by *The Would-Be Gentleman*, and are created on the Macintosh using support programs. The data files are described briefly here and in more detail in the section **Support Programs and Data Files.**

Program	Created using
Louis XIV (main program)	Sunking/4.1/Finance.text on Lisa Pascal Workshop 3.9. Compiled with Exec.text executive program.

File	Created using
FirstMail.1.dat	Mail editor
FirstMail.2.dat	Mail editor
SecondMail.1.dat	Mail editor
SecondMail.2.dat	Mail editor
Office.dat	Office editor
Lease.dat	Lease editor
Marriage1.dat	Marriage1 editor
Marriage2.dat	Marriage2 editor
Final Stats	Created during execution; readable by Examine program
Instructions	File or another Text editor/SetFile
StartText	File or another Text editor/SetFile
StartupScreen	Thunderscan/MacPaint/MacDraw/ScreenMaker
Pictures	Thunderscan/MacPaint/MacDraw/Scrapbook
Saved Simulation	Created during execution; readable by main program.

System Requirements and Compatibility

The software comes on a 400K diskette in MDS format and runs on Macintosh-128K, -512K, -XL, -512E, and -Plus computers. In addition to the **Louis XIV** application and data files, a Macintosh System 2.0, MiniFinder and ImageWriter driver are installed on the diskette in a System Folder. MiniFinder replaces Finder 4.1 to conserve space on 400K diskettes, since the only executable application is the Louis XIV simulation itself. Those who use the simulation diskette on Macintosh-512E and -Plus computers can copy the **Louis XIV** application and **Game Files** folder to an 800K diskette or hard disk drive with System 2.3 and Finder 5.3 installed. If the Hierarchical File System (HFS) is used on the 800K diskette or hard disk, then the application Louis XIV should be placed in the **Game Files** folder holding the simulation data files.

List of procedures

The following list of procedures and functions defines the simulation program **Sunking/4.1/Finance**. Numbers represent the different program segments that organize related procedures. Some segments are not declared as contiguous blocks, but all procedures marked by the same number are in the same segment. Segment zero is special in that it is the main segment that always remains in memory. See the **Source Code** listing for more complete documentation of procedures and functions, and see the sections **Description of Segments** and **Data Structures** for more information about the organization of the simulation program.

List of Procedures

```
7 ----------
        Bankrupt
        CalcHarvest
        NextLand
        NextTextiles
        GetCost

        LoseRentes
        CalcPayment
        NextRente
0 ----------
        SellYRente
7 ----------
        CheckDebt
        RaiseSalary
        NextOffice
        NextMarriage
        NextLease
        CalcExpenses
        KillKid
        CheckSexDeath
        CheckDeaths
        CheckWill
        AddChild
        DemoGraphics
13 ----------
        PutWillItems
        DisCancel
        MakeWill
4 ----------
        SwitchGen
        Displetter
        NextCorr
        NextProct
        EndSimulation
7 ----------
        GoToNext
16 ----------
        DispWealth
        DispSex
        DispPersonal
12 ----------
        HarvValue
        DispLand
        DispText
        DispLease
        DispOffice
        DispRente
        ContButton
        DisplayStatus
```

8 ----------
DispOldManageValues
ProcessManage
ManageMLand
DoConvCheck
InitConvert
DoConvert
ItemChecked
ConvertTitles
ManageLand
9 ----------
BuyMiscLand
BuyTitledLand
BuyLand
BuyTextiles
AddRente
NumKRente
BuyRente
DoCheck
AdvanceOffice
0 ----------
ExaOffItem
GetOffBought
9 ----------
AddOffice
Credentials
OwnOffice
BuyOffice
BuyLease
Purchase
10 ----------
LoseLast
SellKRente
SellRente
SellAdvance
HideCtl
FewOffices
LoseOffice
SellOffice
SellLand
Sell
2 ----------
SetUpWindow
SetUpControls
SetUp
ScrollBits
Increase
Decrease
PageScroll
DoScroll
SetScrollMax
0 ----------
Treasury
2 ----------
ReadText

3 ----------
MarrCheck
ExaMarrItem
GetMarrChecked
DoMarriage
14 ----------
PlanFamily
AboutProgram
BuyNobility
5 ----------
SaveRente
SaveOffice
SaveKid
SaveSimulation
QuitHandler
ReadRente
ReadOffice
ReadKid
LoseKids
LoadSimulation
0 ----------
DoCommand
6 ----------
FinalStats
StopSimulation
0 ----------
Main

Description of Segments

The simulation program Sunking/4.1/Finance is divided into eighteen segments (labeled 0-16 and DoPicture). Segments allow the program to work with limited memory, using the segment loading features of the Macintosh Toolkit. Segments also serve as a documentation aid, since they are divided up in a logical manner and serve specific purposes.

Segment 0: This segment is special in that all its procedures are always in memory. The procedures in this segment are the low-level utilities (such as procedures to convert numbers to string, light radio buttons, and get text from dialog boxes). Contained in this segment are also procedures called regularly in the simulation, such as the section that manages the selection of investment icons, the section that manages the information window, and the main event loop. Finally, there are a few special purpose procedures that are called by procedures from more than one of the other segment (such as SellYRente).

Segment DoPicture: This segment opens a resource file containing pictures and displays the appropriate picture in a window. The picture is resized automatically to fit the window.

Segment 1: This segment is the initialization segment. Its procedures are called at the beginning of each generation to set up the various windows and menus that make up the display, initial settings for the player, and system variables like filenames and resource ids.

Segment 2: This segment reads and displays the instructions from an external file. The window created has scroll bars and a go-away box.

Segment 3: This segment is the marriage handler. Its procedures display the names of the brides, give information about brides, and manage courtships.

Segment 4: This segment handles transition between generations. One procedure (SwitchGen) transfers the person's posessions from the player to his heir according to the will, and another procedure (EndSimulation) handles transition into the third generation resulting in the conclusion of the simulation.

Segment 5: This segment controls the saving and restoring of the current state of the simulation. The procedure SaveSimulation is the overall manager for the saving process, while procedure

LoadSimulation manages the restoration process. Procedure QuitHandler is called when the player wishes to Quit the simulation and, if the player wishes, saves the simulation before quitting.

Segment 6: These procedures are called when the simulation is about to end, either by the menu choice **Quit**, a player who completes a generation without an heir, a player who is hanged through an injudicious purchase of a lease, or a player who successfully reaches the third generation. These procedures save the player's position (if the player didn't just quit) and then stop the simulation. The Final Stats file is updated and the Macintosh ejects the diskette. A text window describes the next steps for the user. If finished, the player can simply remove the diskette and turn off the computer. To replay the simulation, the player must reinsert the diskette and click twice on the icon named **Louis XIV**.

Segment 7: This segment is called when the player chooses the **Next Interval** selection of the **Progression** menu. New investment prices and opportunities are created, and old ones evaluated according to the player's choices. The procedure GoToNext moves the player forward six months and calculates the player's income and expenses for that period. Other procedures calculate demographics (births and deaths) for that period.

Segment 8: This segment is called when **Manage** is chosen from the **Investments** menu. Since only land is managed, these procedures handle the allocation of land for the harvest and the conversion of titled lands from one title to another. Please note that land can only be allocated in the Spring for harvest in the Fall, and that conversion of titled lands is only possible when titled lands are owned in sufficient quanitities (as a Marquisat is larger than a Vicomté is larger than a Siegneurie).

Segment 9: This segment is called when the player chooses to buy an item. Its procedures handle the acquisition of miscellaneous and titled lands, investment in textiles, and purchase of the King's rente, offices, and leases. The procedure Purchase is the overall manager of this segment.

Segment 10: This segment is called when the player chooses to sell an item. Its procedures allow the player to sell either the King's rente or rente of his/her own, to sell offices, and to sell miscellaneous land. The procedure Sell is the overall manager of this segment.

Segment 11: This segment is used to read text from an external file into a text edit record. It is declared as a separate segment because it is called by both Segment 1 (to read in the initial text) and by Segment 2 (to read in the instructions).

Segment 12: This segment handles displaying the player's status in the various categories (financial and personal). Its procedures display a player's investment status for land and grain, leases, offices, textiles, and rentes (by choosing the appropriate investment icon and then **Status** from the **Investments** menu).

Segment 13: This segment handles the creation of the player's will. It is called when the player chooses **Make a Will** from the **Personal** menu. It allows the player to enter values for the will, and determines whether or not it is in accord with traditional practice.

Segment 14: This segment consists of three miscellaneous procedures that don't really belong in any of the other segments. One procedure handles family planning in the second generation, one procedure tells the player about the authors of the program, and the third procedure allows the player to purchase a letter of nobility.

Segment 15: This segment manages the selection of court protectors for the player. It is called to set up the list of protectors available in a given year and to manage the selection of a protector by the player through the **Choose Protector** selection on the **Personal** menu.

Segment 16: This segment calls two procedures that give the player a summary of the player's financial and personal status through the **Wealth** and **Personal** selections on the **View** menu. A third procedure, DispSex, figures out the number and sexes of children for the Personal status window.

Data Structures

The program has two main data structures, the **DateType** and the **AssetsType**.
DateType holds the year (as longint) and the season (as a boolean variable FALL).
AssetsType is a structured data type, consisting of many levels of records, arrays, linked lists, and simple types (integer, longint, char). The following is a complete description of **AssetsType**:

Land:

```
HarvestType = (poor, fair, good, excellent);

Landtype =      record

                    Yield, Price, ShareCrop, Kind, Rent, Inherited, Bought, Seigneurie, Vicomte,
                    Marquisat, Lost, Value : longint;

                    Local, Regional : HarvestType
```

This record tracks the player's grain yield for each harvest, grain price, allocation of lands for the harvest, amount of land inherited (cannot be sold), amount of miscellaneous land bought by the player (during each generation, not each season), titled lands owned by type, amount of grain lost in storage, total value of all lands owned, and the quality of the local and regional harvests.

Rente:

```
RentePtr :      ^RenteType;

RenteHandle :   ^RentePtr;

RenteType =     record
                    Year, CostDenier : longint;

                    Fall : boolean;

                    Next : RenteHandle;

RenteRec =      record
                    FaceDenier, CostDenier, Return, GotThisYear, Owe, SoldVal, Payment : longint;

                    IndivRentes : RenteHandle;
```

These records track the current face value of rente, the discount cost of rente, the amount of return from rente, the amount owed from personal rente, the value of personal rente sold, the seasonal payment due on personal rente sold, and a list of all rente purchased from the King, including year of purchase.

Office:

```
OfficePtr =     ^OfficeRec;

OfficeHdl =     ^OfficePtr;

OfficeRec =     record
                Salary, AmtPaid, Value, Prestige : longint;

                Title : Str255;

                Next : OfficeHdl;

                Inherited : boolean;

OfficeType =    record
                TotPurchase, Number, Salary : longint;

                OfficeList : OfficeHandle;

                Levied, Raise : boolean;
```

OfficeRec stores information about individual offices in a linked list. Information includes the salary of the office, the amount paid for the office, the full value of the office, the prestige associated with the office, the office title, a pointer to the next office in the list, and if the office was inherited.

OfficeType stores information about the offices purchased, such as number purchased, total purchase price, and total salary. It keeps a linked list of all offices purchased. It tracks taxes levied against office value and royal gratuities (such as a raise for a fee).

Lease:

```
LeaseType =     record
                FaceValue, Offer, OldOffer, NumBought, GotThisYear : longint;

                Title : str255;

                Hanged, Bought : boolean;
```

This record stores the current (this year's) lease's face value, the offered price, the last offered price, the total number of leases purchased, the return on a lease held one year, the title of the lease. It also records if the lease that causes hanging of the collector was purchased and if a lease was purchased in the current year.

Marriage:

```
BrideRec =      record
                Name, Father : str255;

                Age, Dowry, Group : longint;

BrideArray =    array [1..NumMarrPerYear] of BrideRec;

AvailRec =      record
                IsAvail :boolean;

                Year : longint;

MarrType =      record
                ThisYear : BrideArray;

                Bride : BrideRec;

                MarrBelow, Married : boolean;

                Failures : longint;

                Available : AvailRec;
```

These records store the current selection of brides. When a player gets married, the record stores the bride's name, her father's name, her dowry, her prestige level, and her age. It records the prestige related to a marriage, the player's marital status, the number of refusals from prospective brides, and a player's availablility/eligibility according to the date of the last refusal (three years must pass before eligibility is restored).

Children:

```
KidHandle =     ^Kid Ptr;

KidPtr =        ^KidRec;

KidRec =        record
                Birth : DateType;

                Next : KidHandle;

ChildRec =      record
                Number, NumBoys, NumGirls : longint;

                Boys, Girls : KidHandle;

                NextBirth : DateType;
```

These records store the total number of children by sex, the number of boys, and the number of girls. They store a record for each child of a given sex in a linked list for demographic calculations.

Will:

```
WillArray =   array [1..NUMWILLCATEGORIES] of longint;

WillType =    record
              Distribution : WillArray;

              Made, WasInAccord, InAccord : boolean;
```

This record tracks the player's distribution of his assets in the will, if a will was made in each generation, concordance with tradition prior to a new birth, and concordance with traditional distribution of wealth.

Protector:

```
ProctType =   (Cornuel, Mazarin, Particelli, Conde, Fouquet, Colbert, Maintenon,
              GrandDauphin, DukeOfBurgundy, Generic, NoProtector);

ProctArray =  array [1..5] of ProctType;

ProctRec =    record
              Name : ProctType;

              YearFail, NumFailures, ThisProctFail : integer;
```

This record tracks the name of the protector associated with the player. It also tracks rejection from a coterie by the year of the failed approach, the number of such rejections, and the number of rejections in a season.

Mail:

```
MailRec =     record
              Content : str255;

              Contact : ProctType;

              Year, Cash : longint;

              Fall : boolean;
```

This record tracks historical correspondence read from external files. It stores mail messages according to their content, protector associated with the message, the year the message is to be sent, the amount of gain or penalty in cash associated with the message, and the season that the message is displayed.

Other variables:

The **AssetsType** data structure controls these variables:

<u>Pascal type longint</u>

Grain:	The amount of grain held.
Cash:	The amount of cash held.
Textiles:	The amount the player has invested in textiles for the coming year.
TotalVal:	The total value of the player's financial assets.
CostOfLiving:	The player's annual cost of living.
Debt:	The player's current indebtedness.
Taxes:	Annual taxes due.
Age:	The player's age.
Generation:	The current family generation (first, second, third).
TooAmbitious:	The number of times the player has tried and failed to purchase an office (e.g. an office with a prestige rating of 80 cannot be purchased).
OldPrestige:	If the player is penalized for purchasing too prestigious an office, AssetsType keeps track of the previous prestige.
Prestige:	The player's prestige level.

<u>Pascal type boolean</u>

PresFallen:	If the player has purchased a lease after he has achieved a prestige of 70 or greater, this is noted.
Quit:	If the player quits, successfully reaches the third generation, or is hung.
Won:	If the player has completed the simulation into the third generation.
Noble:	If the player has achieved nobility.
ChoseProct:	If the player has chosen a protector during the current season.
BoughtLetter:	If the player has bought a letter of nobility.
SoldGrain:	If the player has sold grain during the season in quantity adequate to stop dissatisfaction among the peasants.

Internal Data Management

The simulation also uses a variety of data included in the main program source code. Some data is used to set prices for investments, while others describe parameters of family and protectors. Although this data exists as program code, it can best be described as attributes of investment, prestige, cost of living, wills, taxes, protectors, and family. See the previous section for more complete information about data structures.

Investment

Land

Miscellaneous land represents the amount of arable land in the local area. Miscellaneous land can be purchased during any season and in any quantity up to the total of 2400 hectares. It can be resold in any quantity, unless inherited. The standard price of miscellaneous land is £575. Some protectors benefit the player with a discount on the price of miscellaneous land.

Grain returns and pricing are calculated on the basis of the local and regional harvests. The quality of the regional harvest determines the price of grain in livres per quintel, while the quality of the local harvest determines the yield in quintels per hectare. The price increases from Fall to Spring, as supplies diminish over the winter. The simulation returns the quality of a harvest as blight, poor, good, and excellent with varying frequencies.

Quality of Harvest	Frequency	Yield (local harvest)	Price (regional harvest) Fall - Spring
Blight	4%	0 quintels	15 - 25£
Poor	32	4	7 - 10
Good	32	5	5 - 6
Excellent	32	6	4 - 4

Land can be managed for the harvest in three ways: Renting in Kind, Renting for Cash, and Sharecropping. These choices vary in the stability and range of their returns. Renting in Kind returns grain at a rate of 5 quintels per hectare and this yield is paid regardless of the quality of the local harvest; the value of the grain is dependent on the price determined by the regional harvest. Renting for Cash pays £29 per hectare and, as the least speculative form of management, it is paid regardless of the quality of the harvests. Profits from sharecropping equal the price of grain at the time of sale multiplied by the yield of grain in the harvest, and are thus dependent on both local and regional harvests.

Grain is lost to rats, fungus and rot, and damp storage at a rate of 20% each Fall.

Peasant revolts can occur in the transition from Fall to Spring during years when both Regional and Local harvests are less than Good. If a sufficient quantity of grain (10% of holdings) is not sold in the Fall to provide for winter food, then 97% of grain holdings are lost to a revolt.

Titled lands can be purchased in lots and have great bearing on the prestige of the player. They can be upgraded from one title to another, provided that the amount of land owned under one title is in sufficient quantity for a lot of the higher title and the difference in price per hectare is paid. Titled land is included in the management of lands for the harvest. Titled land cannot be sold.

Title	Size in hectares	Price per hectare
Siegneurie	75-150	£700
Vicomté	300-450	£850
Marquisat	600-900	£1000

Rente

Rente is the term for a loan of £1000. Rente are made either to the King (King's rente) or borrowed for personal use (Personal rente). Rente can be treated as a speculative investment with the strategy "buy King's rente at a low price and sell at a high price" (King's rente), as a means of acquiring cash (Personal rente), or as a steady investment income (annuity, or interest returned on King's rente). Denier is a measure of a rente's market value or price; one denier is 1/14 of the value of one rente, or 1/14 X £1000. There are four market values that occur with different frequencies during the simulation, and these values can change in any season. The following table outlines denier, prices, and frequency of prices.

Denier or market price	Price in Livres	Frequency of this price for rente
2	£143	4%
7	£500	22%
11	£786	60%
14	£1000	14%

King's rente:
A maximum of 400 King's rente can be purchased at a given time. King's rente can be resold for cash. The current price compared to the price at time of purchase determines profit or loss.

Personal rente:
Personal rente are loans taken out by the player. A player can borrow cash in any quantity, but the player must pay interest at the market price at the time that the loan is made. If a low price is in effect when a personal rente is sold, the player pays interest at a higher rate than for a loan obtained at the full price of denier 14. At denier 2, for instance, the player pays back seven times the interest rate of a rente sold at denier 14.

In some situations the player might lose enough money on investments to be forced to take a personal rente in order to pay debts. The dialog for sale of personal rente appears automatically.

Annuity:
King's rente also return an annuity in cash each year they are held, equal to one denier per rente purchased. At rente equal to denier 14, one denier equals 1/14 of £1000, or £71; at rente equal to denier 18, one denier equals 1/18 of £1000, or £55. Payment of the annuity may be voided or reduced based on the King's decree. The following table shows the payment, the frequency of that payment, and the annuity paid for each rente of £1000 purchased from the King.

Payment (as % of annuity)	Frequency	Return in £ at denier 14	Return in £ at denier 18
0	2%	0	0
62.5	33%	44	34
100	65%	71 (full annuity)	55(full annuity)

Special cases:

Personal rente always result in a cash return at the market rate. A player is essentially giving a note for £1000 in return for the cash value of the rente at current denier. These rente cannot be reclaimed and interest on the loan will be assessed each year as an expense.
In 1664, all rente purchased since 1656 are declared void by the King.
In 1660, rente are devalued from denier 14 to denier 18 by order of the King.
In 1648, the King made a payment of 62.5% on the annuity for King's rente.
Some protectors do not allow their associates to purchase the King's rente.
Some protectors provide warning signals about the price of rente.

Offices

Offices can be bought and sold during any season. From 1642 to 1652 office prices are lower by 20 percent.

Special case:
In the Spring, levies may be raised against offices owned by the player. The King can decree a 20% fee on the value of each office or raise salaries by 25% while charging a one-time fee of 10% of total salaries. This happens by chance, with the frequency of each levy set.

Frequency (yearly)	Levy
3%	20% of total value of offices held
4%	10% of total salaries for offices held

Leases

Leases, like offices, are purchased from the government. They are fees paid to the player for acting as tax collector; in essence, they are speculative investments. Leases have three prices, determined by the market, called Official Value, Face Value, and Offered Price. The player uses the Official Value and Face Value to judge the worth of the lease compared to the Offered Price.

A lease is bought in the Spring at the Offered Price; the return is paid the following Spring as a percent of Face Value.

Face Value is calculated by generating a price randomly until it exceeds a minimum of £5000. The Offered Price, which a player must pay to secure the lease, is calculated each Spring as a percentage of the Face Value.

Offered Price (as % Face Value)	Frequency
28	20% (one chance in five)
48	20%
58	20%
68	20%
78	20%

The player makes money on a lease when the discount price paid is lower than the percentage returned the Spring after the lease was purchased. The percent return on the Face Value of a lease is based on the regional harvest.

Regional harvest	Frequency	Percent return of Face Value
Blight	4%	10
Poor	32%	40
Good	32%	65
Excellent	32%	100

Special cases:
Some protectors do not allow their associates to purchase leases.
Some protectors offer financial benefits to associates in the form of discount leases.
For more information, see the section on **Protectors** below.

In 1639, the lease is automatically set to "Royal toll on herring and salmon in the Carenton District." This lease, if purchased, leads to hanging in the following year.

Textiles

Textiles are purchased in the Fall and return their value in the Fall following the purchase. Profits in the textile market depend upon the regional harvest, the quality of which sets the price of grain and the return on meney invested in textiles. The following table shows the relationship between the quality of the regional harvest, the frequency of that kind of harvest, and the percentage return on an investment in textiles.

Regional harvest	Frequency	Percentage return on investment in textiles
Blight	4%	-20
Poor	32%	1
Good	32%	9
Excellent	32%	18

Prestige

The procedure CalcPrestige computes the player's prestige index with the following rules. The first set of rules determine the tens digit of the prestige index. The second set determines other single digit penalties and additions.

Primary prestige points (ten's digit)

If the player begins the Spring with a prestige over 70 and buys a lease during the season, the player can *never again* go into the 70's or beyond.

If the player has no offices, then prestige begins at 30.

If the player owns an office, prestige begins at 40.

If the player owns an office that is too ambitious for his standing, prestige begins in the 20's.

If the player is a nobleman and owns a Seigneurie then prestige begins at 50. If the player is a noble man and owns either a Vicomté or Marquisat prestige begins at 60.

If the player owns both titled land and a very high office then prestige begins at 70 or 90.

Prestige cannot drop below 30 unless a player tries to enter the 80's, in which case prestige is lowered to 20. It can go lower than 20, depending on other penalties.

Prestige can never go into the 80's.

Secondary prestige points (one's digits)

These secondary points can never move the player into a higher category, but they can drop him into a lower one.

A player loses one point for every lease bought.

One point is given for every 100 hectares purchased.

A point is given if the player is married.

Two points are lost for each unsuccessful courtship.

Three points are lost for marriage to a woman below one's prestige.

Two points are lost for each purchase of an office that is too ambitious for one's prestige.

Two points are given for the purchase of a letter of nobility.

One point is subtracted for rejection from a protector's coterie; three points are subtracted if a player has been rejected more than once.

One point is given for every office rated equal to or above the starting prestige.

Nine points are given if a Marquisat is owned.

If the father's will was not in accord with tradition, a player loses four points.

If the sum of the above is greater than 9, then the one's digit is set to 9.

If the prestige is less than 20, then it is set to 20.

If the base prestige rating is greater than 20 but the calculated prestige ends up less than 30, then it is set to 30. This preserves the 20's rating for the overly ambitious player.

Cost of Living

The procedure CalcCostofLiving computes the player's cost of living.

It first calculates the base cost based on the player's offices. If the player owns no office, the base cost is £600 with no titled land, £1500 with a Vicomté, or £2000 with a Marquisat.

If offices are owned, procedure GetFactors is called to get the multipliers for the offices. The multipliers are based on the highest prestige rating of the offices owned.

Highest prestige rating	Multiplier	Percent
20, 30, 40	1	25%
50	2	50%
60	3	50%
70	4	50%
80, 90	8	50%

The base cost is then computed by multiplying the highest salary by MULT and then adding in PERCENT percent of the rest of the salaries. If the player is not married, then the cost of living is 40% of the result.

If the player is married, the cost of living is increased 20% by each child.

If the cost of living is less than £600, then it is set to £600 as the minimum cost of living.

Wills

A dialog to create a will appears on the screen when the player selects the **Make a Will** choice on the **Personal** menu. It also appears when the death of the simulation's character is imminent and a will has not been made. The formula for accord with tradition follows:

Party to the Will	Minimum Percent of Total Wealth
The eldest son	20% plus (80 times (1/n)) where n is the number of children
Other children	100/2n% apiece
Non-Kin	1%
Other Kin	1%
Charity	1%
The Church	5%

If the player's first will is not in accord with traditional practice, it will cost the next generation 4 prestige points.

Taxes

A tax is assessed each Spring equal to £3 per hectare for all land owned.

Protectors

This list gives information about protectors including their years in power, conditions for acceptance to their coterie, financial benefits of association, and theconsequences of association at their death. Cornuel is automatically assigned as protector to the first generation. Costs related to having no protector are delivered by historical correspondence and are always deductions in a player's cash framed as taxes, gratuities paid, and other penalties.

Name: Cornuel
Years: 1638-1640
Acceptance Conditions: Automatic
Financial Benefits: None
Consequences of association at Protector's Death: No Protector

Name: Particelli
Years: 1638 -1648
Acceptance Conditions: Must Purchase at least 3 leases
Financial Benefits: None
Consequences of association at Protector's Death: No Protector, Bankruptcy

Name: Mazarin
Years: 1640 - 1661
Acceptance Conditions: £15,000 in cash.
Financial Benefits: Leases at 3/4 price
Consequences of association at Protector's Death: No Protector

Name: Condé
Years: 1640 - 1686
Acceptance Conditions: Must own some titled land.
Financial Benefits: Misc. land at 1/2 price 'til 1653, 2/3 price 1653 - on; £5000 gift if peasant revolt destroys your stored grain and barns.
Consequences of association at Protector's Death: Pass to Grand Dauphin's coterie.

Name: Fouquet
Years: 1654 - 1661, FALL
Acceptance Conditions: Must have cash holdings greater than £35000.
Financial Benefits: Leases at 3/4 price,
Consequences of association at Protector's Arrest: No Protector, Bankruptcy

Name: Colbert
Years: 1656 - 1683
Acceptance Conditions: Must hold at least 2 offices and have 50+ prestige.
Financial Benefits: Informed of Rentes at 2
Consequences of association at Protector's Death: Pass to Duke of Burgundy's coterie.

Name: Mme. de Maintenon
Years: 1680 - 1715 (end of game)
Acceptance Conditions: Prestige 60+.
Financial Benefits: None
Consequences of association at Protector's Death: None; Maintenon lasts to end.

Name: Grand Dauphin
Years: 1683 - 1711
Acceptance Conditions: Must own titled land.
Financial Benefits: Misc. land at 1/2 price
Consequences of association at Protector's Death: No Protector

Name: Duke of Burgundy
Years: 1683 - 1712
Acceptance Conditions: Must own two offices and have 50+ Prestige.
Financial Benefits: Informed of Rentes at 2, CANNOT buy leases
Consequences of association at Protector's Death: No Protector

All protectors have additional costs and benefits given by mail messages.

The game will be changed to show a penalty of 1 prestige point if rejected once in the protector dialog, and a penalty of 3 points and a two year wait if rejected twice.

Marriages and Family

Marriage

To qualify for marriage to a bride, a prospective suitor must meet two requirements. First, his prestige group must be greater than or equal to the prestige group of the bride. Second, his total assets must be at least twice as large as the bride's dowry.

A failed courtship costs two prestige points.

A failed courtship results in three years of ineligibility for the suitor.

Four prospective brides are available each season, and are selected randomly from the marriage1 or marriage2 data files.

Family

The first generation ends in 1676, the second generation ends in 1715 at the death of Louis XIV.

In the first generation, a son is born automatically in the first generation at the end of the first year of marriage. This son is not allowed to die, protecting the first generation of the simulation against failure at the death of the father. Other children are born at random intervals until the mother reaches 36.

In the second generation, children must be planned using the **Family Planning** item on the **Personal** menu. As in the first generation, the eldest son is protected.

A bride who is 38 years of age or older cannot bear children.

10 percent of children die before age one, 13 percent die before age 20. After age 20, children are protected against death.

Creating a Program Diskette for Players

Complete information on the contents of diskettes is included in the **Table of Contents** of this binder. *The Would-Be Gentleman* diskette should be constructed to include the main program **Louis XIV** and a **Game Files** folder containing required external data files. If these files are not included, or not accessible (as is the case with HFS when the main program and the data are not in the same folder), the simulation will display an empty window where external data is required.

Compilation

Compilation of the source code for the main program and support programs requires the application source code, its resource file, and the exec.text executive program. Naming conventions are followed:

Main program	Sunking/Version/Application.text
Resource	Sunking/Version/ApplicationR.text

Exec.text is invoked by the Run command from the Lisa Workshop shell as follows:
R(un)
Program to Run: <exec(Sunking/Version/Application)

Compilation with Workshop 3.9 is very reasonable for speed. Earlier versions compile extremely slowly and should not be used with *The Would-Be Gentleman* code. The source code, once compiled, is written to the Macintosh diskette as Finance. It should be renamed Louis XIV.

Releases of *The Would-Be Gentleman*

Releases

1.0	1/85	First release, Macintosh interface, used in seminar at Stanford University
2.0	3/7/85	New economic model under development
3.0	8/85	Pictures added, improved rente management
4.0	10/85	Economic model and correspondence improved
4.1	3/86	Release version, testing completed, economic model debugged
4.1(a)	10/86	Release version, appended User's Manual

By date	**Event**
1/10/1985	Course used version 1.0
August 1985	EDUCOM used ver 3.0 in Integrating Software into the Unversity Curriculum
1/10/1986	Apple Computer handed out version 4.0a at the Apple University Consortium
3/2/1986	EDUCOM handed out demonstration copies of pre-release version 4.0b
3/13/1986	Academic Courseware Exchange released version 4.1
10/3/1986	Academic Courseware Exchange released version 4.1(a)

User:	Tom M.
Application:	MacWrite 4.5
Document:	Programmer's Manual Appendix, v
Date:	Tuesday, June 16, 1987
Time:	17:45:23
Printer:	Bullwinkle

The Would-Be Gentleman
Programmer's Manual Appendix for version 4.3

Tom Maliska
Faculty Author Development Program
Stanford University
September 1986

The Turbo Pascal compiler:
The Would-Be Gentleman version 4.3 source code compiles using the Turbo Pascal v. 1.0 compiler from Borland International. This compiler was chosen as a low-cost and readily available language product.

Compilation:
The source code is divided into a main program (SK.Pas) and a number of units (SKSeg0.Pas, SKSeg1.Pas, SKSeg7.Pas, SKSeg11.Pas, and SKSeg15.Pas). SK is shorthand for Sun King, Louis XIV's nickname. You may wish to change these names to suit your own application or make them generic (Unit0.Pas, etc.) for simplicity's sake. The segment numbers in these names correspond to the segments described in the programmer's manual. Segments must be compiled to disk before the main program and installed in the Turbo Pascal library using the UnitMover utility program. There are dependencies between units, so recompilation of one unit may involve recompilation of others in the appropriate order. While this sounds tiresome, the speed with which Turbo Pascal compiles the individual units makes this a fairly reliable and simple process. The order in which the units must be compiled is the following:

=== (dependent only on Turbo Pascal unit libraries)
SKSeg0.Pas

=== (dependent on Turbo Pascal unit libraries and SKSeg0.Pas)
SKSeg15.Pas
SKSeg11.Pas

=== (dependent on Turbo Pascal unit libraries and multiple SK units)
SKSeg1.Pas (dependent on SKSeg0.Pas and SKSeg11.Pas)
SKSeg7.Pas (dependent on SKSeg0.Pas, SKSeg11.Pas, and SKSeg1.Pas)

=== (dependent on Turbo Pascal unit libraries and all SK units)
SK.Pas (Units)

Customization:
Resources for the program contain much of the interactive text for dialogs. It is advisable when designing your program to convert these dialog boxes to suit your own simulation strategy and topics. Resources are contained in the file SK.R; they can be edited with Turbo Pascal and compiled with the Resource Compiler.

ResEdit and other tools can be used to change resource files of the type Myfile.Rsrc, but do not make changes at the source level. It is advisable to make changes with SK.R and the Resource Compiler rather than "on the fly" with ResEdit or other tools.

Much of the rest of the textual data is contained in external data files. The Programmer's Manual for The Would-Be Gentleman describes how to modify the external data files using the Editors, or support programs, and how the main data structures in the main program source, SK.Pas (Units), are maintained. You may wish to read through the source code to familiarize yourself with the many procedures in the simulation; the source code is documented at each procedure and function header. Many changes can be made to the program merely by understanding the main data structure, ASSETS, and changing related constants and If..Then rules in the source.

All output windows must be customized for your simulation. Since output to these windows involves many formatting steps, I have left it to you to design your own writeln statements and make them display properly. The Windows and Dialogs have been provided and are initialized and ready for text display. The original Gentleman output, consisting mostly of writelns, works but has many glitches as I have not completed my own customization with formatting in mind. You may want to explore the use of DrawString instead of writeln; see the source code for the lease Editor in Lease.Pas for an example of this and for source to convert integer values to strings.

Note that some of the output routines are located in the body of the units rather than in the main program; in this case, you must modify the unit code and recompile the appropriate units to test your changes to window and dialog output.

User: Tom M.

Application: MacWrite 4.5

Document: License for Gentleman Source Co

Date: Tuesday, June 16, 1987

Time: 17:46:28

Printer: Bullwinkle

Interested Party 6/2/87

Dear Interested Party,

I am writing to confirm receipt of the source code for *The Would-Be Gentleman* and set forth our guidelines for the use of this source code. Please sign this letter and return it to me as acknowledgement of receipt and agreement to these guidelines, namely:

1. That no software products developed from this source code be distributed for profit, royalty, or pecuniary gain by any party.

2. That you acknowledge Stanford's contibution to any product developed from this source code by incorporating this text in the **About...** dialog:

> "This product is based on source code designed at Stanford University for
> The Would-Be Gentleman, version 4.3.
>
> Faculty Author Development team, Stanford University :
> Carolyn Lougee, Steve Fisher, Michael Carter, Ed McGuigan and Tom Maliska.
>
> Copyright 1985 Carolyn Lougee & the Board of Trustees of the Leland Stanford Junior University."

3. That a copy of any product developed from this source code be sent as a courtesy to:

IRIS, Stanford University
c/o *The Would-Be Gentleman*
Sweet Hall 3rd Floor
Stanford, CA 94305-3091

4. That no distribution of this source code be made to other parties for use other than cooperative work with the addressee. Inquiries from other parties about the source code [redacted] should be directed to the address indicated in #3 above.

Thank you for your interest in our software and source code. Until August 1987, I (Tom Maliska) will be available for limited technical support on this product. Please contact me by phone at (415) 723-1055 or via bitnet at **maliska%portia@stanford.bitnet.**

If you are interested in distributing products based on this source code, please make sure to contact us first. As a point of information, *The Would-Be Gentleman* is available for $7.50 from the Academic Courseware Exchange, a service of Kinko's Copies, Inc. A catalog of ACE products and ordering information can be obtained by calling 1-800-235-6919.

Sincerely,

Tom Maliska
Stanford University IRIS

==========

I hereby acknowledge receipt of the source code for The Would-Be Gentleman and agree to the provisions for its use listed above:

_____ _____
SIGNED DATE

ADDRESS:_____

User: Tom Maliska, FAD Program

Application: MacWrite 4.5

Document: Instructor's Manual

Date: Thursday, October 23, 1986

Time: 7:19:17 PM

Printer: Boris Badinov

Instructor's Manual
The Would-Be Gentleman

Professor Carolyn C. Lougee,
Faculty Author Development Program
Stanford University
September 1986

A Historical Simulation of the France of Louis XIV

"The Would-Be Gentleman" is a simulation of social mobility in seventeenth-century France, designed for use by students in undergraduate classes on the Old Regime. It has been tested in a sophomore/junior-level seminar at Stanford University; the amount of technical information it teaches and the fact that its use should extend over a number of weeks probably make it inappropriate for freshman European survey courses, but it could appropriately be used in graduate-level courses on the seventeenth century.

The simulation begins (at logon) in September 1638, when the future Sun King Louis XIV has just been born. It continues until September 1715, when Louis dies. In the intervening 77 years, two generations of the Marin family attempt to raise the fortunes and status of their family. At the outset, the player receives the following message:

It is September 1638. The kingdom celebrates the birth of Louis le Dieudonné (the gift of God), first son of Louis XIII and Anne of Austria. Circumstances everywhere are not so joyous, however. In Normandy you, Denis Marin, have just lost your father, a bourgeois of Rouen. At age 30, you assume leadership of the Marin family, which for two generations has been moving away from its peasant origins toward the notability.

From your father you inherit an office of auditeur in the Chamber of Accounts of Rouen, which is valued at £21,000. You also inherit £5054 in cash and 42 hectares of cultivable land near Chateauvallon. Use these assets to the best of your ability. If you are shrewd and manage your assets well, you may increase your family's wealth and prestige during the lifetime of Louis le Dieudonné.

As you set out in the world to make your fortune, we wish you well, recalling the words penned two years ago by Pierre Corneille, friend and compatriot, for LE CID:

"A vaincre sans péril, on triomphe sans gloire."
"When there is no peril in the fight, there is no glory in the triumph."

{To start the simulation, press the button on the mouse.}

What follows are 154 decision points (fall and spring of each calendar year) at which Denis Marin (and after 1676 his son Jean-François Marin de Merinville) make investment, management, and personal decisions that are appropriate for the times. The aim is to maximize prestige over the two generations of the simulation and attain the highest possible social standing in 1715. A constant stream of correspondence to the player identifies economic or political opportunities, warns of risks, and informs of windfall gains or unexpected losses stemming from circumstances beyond the player's control.

The investment decisions permit one to buy or sell land, venal offices (e.g., for 25,000 livres Denis can become honorary secretary of the king), textile shares, leases (the term for contracts to collect indirect taxes), or rentes (annuities). The probabilities of making a profit rather than a loss and the size of the profit or loss vary for the different investment types to reflect the economic realities of the seventeenth century. Thus, for example, leases are the riskiest and land the safest investment, but the potential profit on a lease transaction if one is lucky is far higher than the monetary return on land will ever be. Textiles are less volatile than leases but vulnerable to occasional market collapses, and rentes are fairly steady performers but subject both to market fluctuations and to the vagaries of royal penury which can from time to time cause the king to place a costly surtax on them. In playing, the student discovers the short-term and long-term potential of various investments and learns which investments make sense at any particular juncture in the simulation.

Management decisions concern land-rental and sales of the grain that they accrue as landowners. The Marin persona can let their acreage in return for cash rents, rents in kind, or sharecropping. They can store or sell the grain they receive as rent in kind or crop shares. Each fall's harvest is volatile in quantity, depending upon the weather, so the amount of grain received as rent and its market value fluctuates. These fluctuations determine the relative profitability at a given time for the landowner of the three options for renting. Over the course of the 77 years the profitability of the three options evens out, but in a given fall the profit accruing from the three can be very different and the student player benefits or suffers accordingly from the choices made. Grain can be sold immediately or stored for sale at a time when supplies may become scarce and drive the price up. Speculation can reap a handsome profit if prices rise before spoilage reduces one's stocks, but prices can also fall and erase the certain gain that immediate sale would have brought.

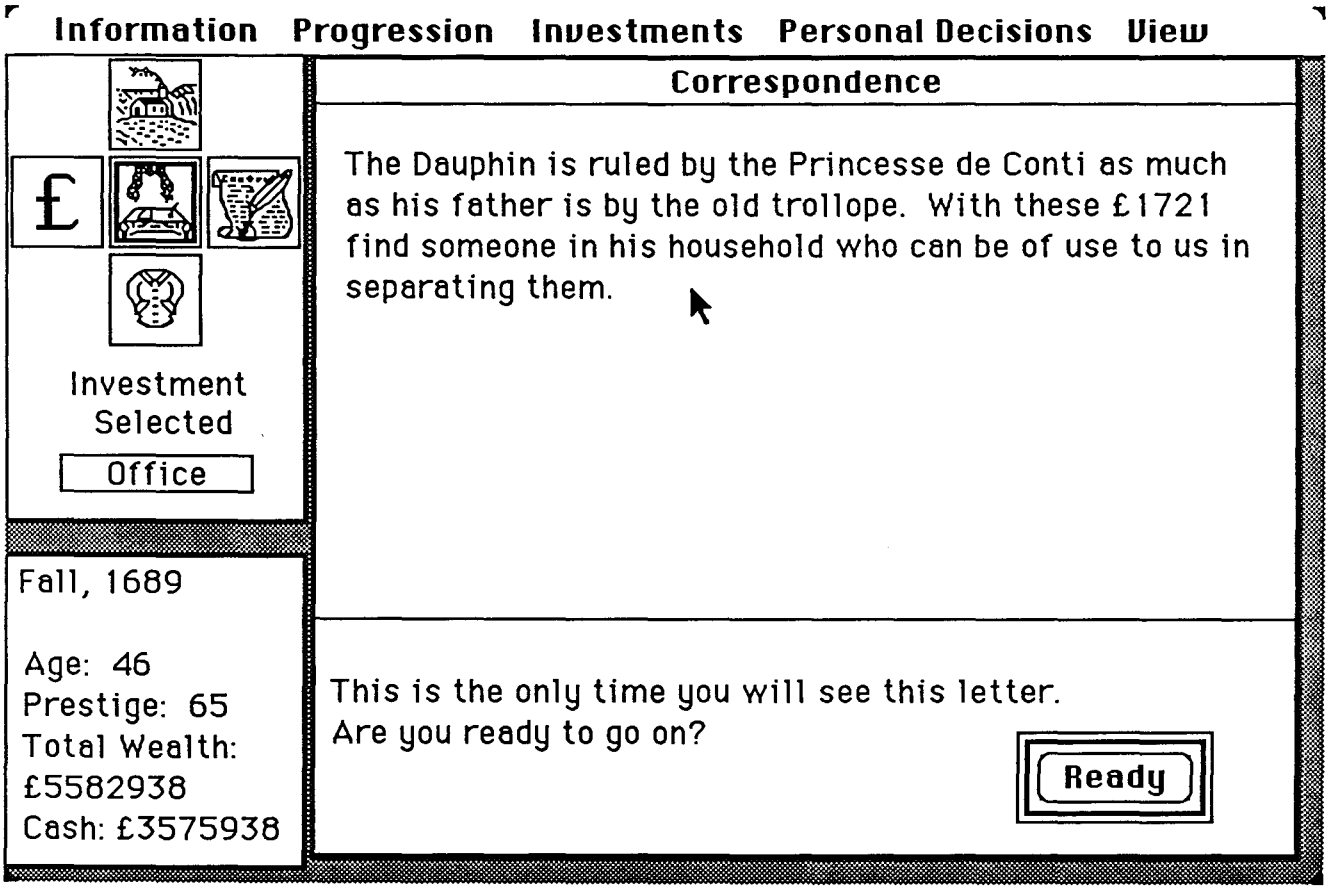
Personal decisions include choosing a wife, having children, finding a protector, making a will, and seeking titles of nobility. Whenever one chooses to enter the marriage market Denis, or later Jean-François, is presented with information on the personal characteristics, family status, and dowry size of four available young ladies. Marin's objective is to choose the wife who will bring him the most tangible benefits in terms of fortune and connections, but he must be careful to

observe the proprieties of the age; should he be so foolhardy as to court a woman whose status is superior to what he has to-date attained, he will be humiliated with a refusal, and as a result so shamed that no family will entertain his courtship proposal again for several years. Once married, Denis and Jean-François will begin to have progeny, which affects both their annual cost of living and their family's future. A special feature of the simulation is the difference in reproduction experience between the first and second of these seventeenth-century generations. In the first generation children begin to arrive one year after the marriage and arrive regularly at 24-30 month intervals. In the second generation children do not arrive automatically at all but only if requested, since this was the period when (as recent demographic studies have revealed) the French aristocracy began practicing family limitation (birth control). In both generations, children die at old regime rates (25% in the first year, another 25% by the age of 20).

Finding a protector is as critical a move as marriage and procreation, for the security of one's investments and social standing over the course of the 77 years depend directly upon whether one has a powerful protector and who that protector is. In the first generation, the available protectors are the financial-political figures who dominated the first half of the reign of Louis XIV: Cornuel, Particelli, Mazarin, Fouquet, Colbert. If Marin does not attain the prerequisites for acceptance into the clientele network of one of these men (usually a certain success with some form of investment, or a wealth minimum, or a certain status ranking), then he will be hit each year with a heavy liability (a monetary fine, a harvest failure, a confiscation). Once accepted into a clientele network, the would-be gentleman will enjoy a series of windfall profits, but must beware the fall of his protector and bail out into another coterie before that happens. Bailing out and joining another coterie are, however, tricky to accomplish, as changes of allegiance were in the seventeenth century. In the second Marin generation, the available coteries are the factions at court: those centering on Madame de Maintenon, the Duke of Burgundy, and the Dauphin. Each faction has its own prerequisites for inclusion and its own rewards of membership. As in the first generation, not belonging to one or the other coterie has severe consequences on finances and prestige.

Seeking titles of nobility is the heart of the exercise, since the objective of the simulation is to raise prestige. Investments yield only money, which does not translate directly into prestige under the Old Regime. Converting money into land and then land into nobility is the means by which fortune can be translated into status in the simulation, as it was in the seventeenth century. Status, the objective of the exercise, is measured on an artificial index of 0-100; the most powerful means for advancing on the index is acquiring titles such as vicomte, comte, and marquis. The simulation displays two indices, wealth (expressed in Livres) and status (expressed on the 100-point scale). Status alone, not wealth, is the measure of success at the end of the game.

The simulation, which sounds so dry and technical in the above description, is also filled with whimsy, which makes it fun for the students to play. Chateauvallon, the location of the Marin estate mentioned on the opening screen, is the name of the current French television show that imitates "Dallas." The names of the three successive heads of the Marin family progress from plain to extravagant, suggesting the increasing refinement and even frivolousness of the higher reaches of the social hierarchy: Denis Marin, Jean-François Marin de Merinville, Hyacinthe-Florent Marin de Merinville. Much of the correspondence reproduces verbatim some of the more colorful letters actually sent by members of Louis' court, for example:



Some of the benefits of using the simulation in class come from the fun students find in it. In my class it was an incomparable ice-breaker, stimulating dialogue among students independent of the professor. So, for example, as I walked to the second class meeting (the first after students had begun to work with the simulation) I could hear from far down the hall the students' boisterous sharing of their various vicarious experiences in Louis XIV's France. One student arrived a few minutes late, saying he had been delayed in printing out his homework screen because a group of his friends in the dorm were using his Macintosh to "play the game." Repeatedly in class someone would interject "Hey, I know her!" when the name of a potential bride or a person mentioned in the letters came up in discussion. Students, then, through this amusing and interactive computerized exercise, established a stronger personal identification with the subject of study than they typically do through reading alone; and the simulation made their classwork in a sense a living historical experience.

Beyond enticing students into an engagement with the past, the simulation was very effective in teaching students historical concepts and abstract problems that are normally very difficult to interest students in. Students' attention can seldom be fastened on the way interest rates were calculated in the seventeenth century, how patronage networks were formed and operated, how endogamy was defined, or the technicalities of investments. But the fact that these all had to be mastered in order to succeed at the exercise transformed what would otherwise be thought boring into a motivational challenge.

The simulation was distributed to students at the first meeting of the ten-week course, with no instructions other than how to use a Macintosh. Students were expected to work on the simulation each week and turn in their best 1715 screen at each class meeting. The readings in the course (see syllabus below) proceeded week by week through the topics integral to the simulation: rural economics, demography, the structure of the court, royal finance, status representations, popular revolts, and so on. As they learned from primary and secondary readings about each aspect of the France of Louis XIV, students became increasingly successful at the simulation. They could apply the readings to understand why the available choices had the consequences they had, especially why these consequences were radically different from what they would be in twentieth-century America.

Indeed, understanding the difference between past and present consequences of a similar action was the overall intent of this teaching tool. When they acted from twentieth-century motivations rather than from seventeenth-century assumptions, students fell nicely into the traps set for them. A common way of losing the game was to arrive at 1715 with loads of cash and high status but without any heir to pass them on to. The students explained in these cases that they had not had

any children because "kids are so expensive!" Words to reflect upon in 1986, surely, but not an attitude that any responsible family-builder in the seventeenth century would have had. Or again, students were often slow to grasp the central lesson of the simulation: that wealth was not directly connected to status/prestige then as it typically is in American society today. In the second week of the course one especially earnest student came to my office to tell me that "there is a bug in your program." I was delighted to have his help in identifying flaws to correct. So he proceeded to explain the "bug": his wealth kept going up but his status index kept going down. This he considered "unfair." I explained to him that that might look like a bug or unfairness to twentieth-century eyes, but that he had to learn to think as a seventeenth-century person in order to understand why that was happening. (He had in fact been making money from leases, which are lucrative but socially disreputable forms of activity in the seventeenth century. Each purchase of a lease therefore costs the player status points in the simulation.) Throughout the course, students were required to identify patterns of consequences of a given game choice, then learn to understand them and use them to interpret a society far different from our own.

The students' immersion in twentieth-century realities had another unexpected outcome in the early weeks of the simulation's use, which we had to move swiftly to correct. Several students "broke the bank" very quickly, attaining status 99 and posts as First Gentleman of the King's Bedchamber. This clearly was not faithful to seventeenth-century experience, so I asked them how they did it. Simple: they knew that if they waited long enough harvests would fail and grain prices would skyrocket, so they simply hoarded their grain until that happened -- 40, 50, 60 years if necessary. Knowing seventeenth-century conditions well, I had never thought of this strategy or provided historically accurate obstacles to this outlandish outcome. So we added rats (a spoilage factor that accelerates with the number of seasons one holds onto one's grain) and peasant rebellions (popular confiscations of grain hoards when harvest failure deprives the local community of sustenance). By the time students did their readings on peasant rebellions they understood why long-term speculation on grain was impossible in the seventeenth century.

The simulation is surely not yet perfectly calibrated; future students will no doubt surprise us further with their twentieth-century ingenuity. But it is now in at least an eminently workable form.

The simulation has some value for the humanities beyond what I have sketched above. It was a very effective means of attracting history students to the computer, which a surprising number of them were unfamiliar with and even resistent to, and convincing them to use the machine not only for the simulation itself but also for writing their papers. The results in improved writing were simply astonishing. These students now feel comfortable with the computer, and I would expect them to be positively inclined toward using it in the future for more complex tasks such as data

analysis. Furthermore, the simulation showed that humanistic subjects can be enhanced by computerized learning. Social status and prestige are highly qualitative and imprecise historical notions. Rendering them into the distinct categorization, even mathematical representation, upon which programming rests was an awesome intellectual challenge and an example of what might be accomplished elsewhere in the humanities with an investment of imagination and time.

SOURCES OF TECHNICAL INFORMATION

General Information

Robert and Elborg Forster, eds., European Society in the Eighteenth Century

Pierre Goubert, The Ancien Regime: French Society 1600-1750

Orest Ranum, ed., The Century of Louis XIV

Economic Data

Micheline Baulant and Jean Meuvret, Prix des cereales extraits de la Mercuriale de Paris (1520-1698)

C.-E. Labrousse, Le Prix du froment en France au temps de la monnaie stable

C.-E. Labrousse, Histoire economique et sociale de la France

Offices, Officers, and their Families

Roland Mousnier, La Venalite des offices

Carolyn C. Lougee, Le Paradis des femmes: Women, Salons, and Social Stratification in Seventeenth-Century France

The World of Finance

Julian Dent, Crisis in Finance: Crown, Financiers, and Society in Seventeenth-Century France

Julian Dent, "The Role of Clienteles in the Financial Elite of France Under Cardinal Mazarin," in J.F. Bosher, ed., French Government and Society, 1500-1850

Yves Durand, Les Fermiers generaux au XVIIe siecle

The Court

Various memoirs and letters from the period, especially Elborg Forster, ed., A Woman's Life in the Court of the Sun King: Letters of Liselotte von der Pfalz

Emmanuel Le Roy Ladurie, "Versailles Observed: The Court of Louis XIV in 1709," in The Mind and Method of the Historian

SYLLABUS USED IN THE COURSE

The France of Louis XIV: Grand Siècle or Tragedy?

This course is one of the History Department's sophomore-level seminars, the first course required for majors. These seminars are intended to teach students "something about the process by which archival material becomes historical description and explanation, and, secondly, what it is about the process of interpretation itself that makes it open to disagreement and revision." The courses "contain a series of exercises that introduce undergraduates to the problems of interpreting sources, of constructing narrative descriptions and historical explanations, and of making sense of differing analytic strategies and interpretations." In the case of French history, we will study the Annales historians' particular "menage à trois" for transforming the past into history: source, problem, technique. Students are assumed to have already completed a year-long survey course in European history.

Week 1: In Search of The France of Louis XIV

Readings: Commentaries on Louis' reign by Pellisson, Bossuet, Leibniz, La Bruyere, Voltaire, G.P. Gooch, Boris Porchnev, Alain Peyrefitte.

Together, these readings serve to introduce the principal lines of conflicting interpretation, identify the key issues they involve, form questions in students' minds that they will carry to the original sources, and make students self-conscious about their own biases, values, and epistemological limitations.

Week 2: Social Structure in the Ancien Regime

Readings: Pierre Corneille, "Le Cid"
Moliere, "The Bourgeois Gentleman"
Charles Loyseau, "Treatises of the Orders and Dignities"
Selected short documents on nobility from Pierre Goubert's Ancien Regime, vol. 1 (letters of ennoblement, nobles' revenues and expenses, views of nobility)

Paper topic :
What social values, what assumptions about the bases of social status and prestige underlie "Le Cid" and "The Bourgeois Gentleman"? Do the two plays espouse the same values and assumptions?

Week 3: Rural Society

Readings: John Locke's Journal of his travels in France 1675-78
Maps, Charts, and Tables extracted from Pierre Goubert's Beauvais et les Beauvaisis (intendants reports, rent rolls, taille rolls, land surveys, censuses, harvest reports, market prices)

Documents on rural life and poverty from Pierre Goubert's Ancien Regime, vol. 1

Paper topic: A "History problem set" based upon a visit to the Beauvaisis, a region North of Paris. Your task is to understand as well as you can the economic situation of the peasantry. What features of the rural economy can you see from the documents provided? Specifically, see if you can calculate what percentage of the population in the Beauvaisis had enough land to feed their own families. In calculating this you will have to make a few assumptions or "best guesses." Do you get a different answer for the two halves of the province? As a simulation, once you've reached your first conclusion, figure what the percentage is if the taille doubles, if all taxes disappear, or if the size of the family changes. Or assuming that holdings remain constant, figure whether the percentage changes across the course of the 17th century. Render your calculations into prose.

Week 4: Popular Revolts and the Fronde

Readings: Pierre Goubert, "The French Peasantry of the Seventeenth Century"
Precis of the conflicting views of Boris Porchnev and Roland Mousnier on the revolts
Selected contemporary documents on the revolts (eyewitness accounts, ordinances, legends, petitions, police reports)

Paper topics:
1) How sound is Goubert's own calculation from last week's documents?
2) Which interpretation, Porchnev's or Mousnier's, does the available evidence best support? Why?

Week 5: Louis Comes to Power

Readings: On the Fouquet Affair, 1661
Louis' Memoirs, Voltaire, Guizot, Madame de Sevigne, photos of Vaux-le-Vicomte, inventory of Fouquet's wealth
On the early reign: Louis' Memoirs, documents reprinted in Orest Ranum, The Century of Louis XIV
On Colbertism: documents reprinted in Ranum

Paper topic:
Formulate the case for the prosecution or for the defense in the Fouquet trial.

Week 6: Religious Sensibility in the Age of Louis XIV

Readings :Pascal, Pensees and Provincial Letters #1-3
Voltaire, "On the Pensees of Mr Pascal"
Documents on the Revocation of the Edict of Nantes and on Louis' religious views

Paper topics:
1) Write Pascal's response to Voltaire, OR
2)What do you see in Pascal's writings that might have inclined Louis to endorse or oppose them?

Week 7: The Court at Versailles as Instrument of Absolutism

Readings :Accounts of the court by Saint-Simon, De la Force, Locke, Lister, Sevigne, Wren, Spanheim, Louis XIV.
Modern Analyses: Le Roy Ladurie, "Versailles Observed" and Norbert Elias, "The Sociogenesis of French Court Society"

Project (mandatory) : Using materials in the Art Library, compare the iconography of Francis I and Fontainebleau with that of Louis XIV and Versailles. What does the contrast suggest about the different "politics" of the two kings? (On Versailles, pay particular attention to the Apollo and Latona themes.)

Paper topics:
1)Write up the above contrast, OR
2)What political functions of the court do the documents suggest?

Week 8: Demographic Crisis

Readings :Three statistical case studies:
Crulai, a Norman village (fertility patterns)
The High Nobility (fertility patterns)
The Beauvaisis (mortality patterns)
Contemporaries speaking about birth and death

Paper topics:
1) According to Hippolyte Taine, "Preindustrial populations had no ability whatever to escape the inexorable grip of their own biology and the caprice of their environment.... The peasant was like a man walking in a pondwith water up to his chin. The least depression in the bottom or the least ripple of a wave, he loses his footing and is submerged." Do the materials available verify or force a revision in Taine's view?, OR
2) What new data does the demographic record offer for an evaluation of how well-designed Colbertism was to deal with the economic problems of seventeenth-century France?

Week 9: Overt Opposition: The Late Reign

Readings: Critical commentaries by La Bruyere, Vauban, Jurieu, Saint-Simon, anonymous songwriters and poets.

User: Tom Maliska, FAD Program

Application: MacWrite 4.5

Document: Support Programs and Data Files

Date: Thursday, October 2, 1986

Time: 6:30:33 PM

Printer: LaserWriter Plus

The Would-Be Gentleman
Support Programs and Data Files

Tom Maliska
Faculty Author Development Program
Stanford University
September 1986

This document contains technical information about *The Would-Be Gentleman*, organized into the following sections.

The Would-Be Gentleman requires a number of program and data files that are created on the Macintosh using support programs. The support programs and data files are described here in detail.

Source Code for Support Programs

Source code to be compiled includes the following items.

All of the following are compiled with Exec.text executive program on Lisa Pascal Workshop 3.9. All run on Macintosh and have File Type APPL and File Creator SIMU.

Program	Pascal Source Code	Resource Code
Louis XIV (main program)	Sunking/4.1/Finance.text	Sunking/4.1/Financer.text
Mail editor	Sunking/4.0/Mail.text	Sunking/4.0/Mailr.text
Office editor	Sunking/4.0/Office.text	Sunking/4.0/Officer.text
Lease editor	Sunking/4.0/Lease.text	Sunking/4.0/Leaser.text
Marriage1 editor	Sunking/4.0/Marriage1.text	Sunking/4.0/Marriage1r.text
Marriage2 editor	Sunking/4.0/Marriage2.text	Sunking/4.0/Marriage2r.text
Examine	Sunking/4.0/Examine.text	Sunking/4.0/Examiner.text

Data Files

Data files required on the Macintosh simulation diskette and used by Louis XIV (main program) include the following items. File Type and Creator are specified as they must match for the main program to be able to read the files. In most cases, the support program used to create the data file sets File Type and Creator automatically. The File Creator "????" is literal and does not indicate an omission.

File	Created by	:Type	:Creator
FirstMail.1.dat	Mail editor	MAIL	????
FirstMail.2.dat	Mail editor	MAIL	????
SecondMail.1.dat	Mail editor	MAIL	????
SecondMail.2.dat	Mail editor	MAIL	????
Office.dat	Office editor	LEAS	????
Lease.dat	Lease editor	LEAS	????
Marriage1.dat	Marriage1 editor	MAR1	????
Marriage2.dat	Marriage2 editor	MAR1	????
Final Stats	execution; readable by Examine program	ENDS	????
Instructions	A text editor/SetFile	CORR	JUNK
StartText	A text editor/SetFile	CORR	JUNK
StartupScreen	Thunderscan/MacPaint/MacDraw/ScreenMaker	SCAN	NONE
Pictures	Thunderscan/MacPaint/MacDraw/Scrapbook	ZSYS	MACS
Saved Simulation	execution; readable by main program.	SAVE	????

Creating Data Files with Support Programs

The following is a list of the support programs and information about the data files they create:

Mail

Support Program Information:

Name mail editor, **Type** APPL, **Creator** SIMU.

Compiled using: Sunking/4.0/exec.text

Source code: Sunking/4.0/mail.text (Pascal code)

Resource code: Sunking/4.0/mailr.text (resource file).

Runs on: Macintosh

Action: The mail editor creates a file mail.dat that holds correspondence for the simulation. The correspondence is shown on the screen when the appropriate date and season arrive, and the protector of the player matches that of the recorded mail message.

DataFile Information:

Name mail.dat, **Type** MAIL, **Creator** ????.

Notes: The mail editor reads and writes only to the file **mail.dat.** Since the mail editor does not read other files, they must be renamed to mail.dat for editing. To make readable files for the simulation, mail.dat must be renamed on the desktop after editing. The names are arranged according to generation (First or Second) and number, as follows: FirstMail.1.dat, FirstMail.2.dat, SecondMail.1.dat, SecondMail.2.dat.

The Would-Be Gentleman requires four files for correspondence since Pascal does not allow a data structure to exceed 32K. To stay within this limit, the list of files in the mail editor should not be allowed to exceed 115 in number or an error will occur.

Messages are stored by number in the list, not sorted chronologically and by protector, so it is the programmer's responsibility to keep the list in correct sequence. Since mail data is extensive and requires frequent changes, an editing feature allows the insertion of messages at any point in the list by message number. The list is automatically renumbered when the new message is inserted. Programmers can add, delete, edit, save, quit, and list mail messages to the screen. The save command must be given explicitly. Quit *does not* save the file automatically!

Offices

Support Program Information:

Name Office editor, **Type** APPL, **Creator** SIMU.

Compiled using: Sunking/4.0/exec.text

Source code: Sunking/4.0/office.text (Pascal code)

Resource code: Sunking/4.0/officer.text (resource file).

Runs on: Macintosh

Action: This program generates a list of offices. Included are the names of the offices, their cost, their prestige level (either 20 (for the 80's), 40, 70, or 90), if nobility is required to own the office, and if nobility *and* a title is required to own the office.

DataFile Information:

Name office.dat, **Type** LEAS, **Creator** ????.

Note: The office list is read in order by number, and it is the programmer's responsibility to keep the list in order from lowest priced office to highest. Offices *are not* arranged by prestige or salary, as both can fluctuate in the simulation. It is important to organize your work first on paper, as the office program does not have edit and insert commands, only a delete command. New offices are added to the *end* of the list.

Leases

Support Program Information:

Name Lease editor, **Type** APPL, **Creator** SIMU.

Compiled using: Sunking/4.0/exec.text
Source code: Sunking/4.0/lease.text (Pascal code)
Resource code: Sunking/4.0/leaser.text (resource file).
Runs on: Macintosh

Action: Generates a list of lease names. Other information about leases (price, return on investment, etc.) is generated during play by the simulation program.

DataFile Information:

Name Lease.dat, **Type** LEAS, **Creator** ????.

Note: Leases are listed in the order added. Since the simulation picks a lease randomly each spring, order in the list is not important.

Marriages

Support Program Information:

Name Marriage1 editor, **Type** APPL, **Creator** SIMU.

Compiled using: Sunking/4.0/exec.text
Source code: Sunking/4.0/marriage1.text (Pascal code)
Resource code: Sunking/4.0/marriage1r.text (resource file).
Runs on: Macintosh

Action: This program generates a list of brides for the first generation. Included are the names of the brides, their age at the start of simulation in Fall 1638, their prestige category, and their father's office.

DataFile Information:

Name marriage1.dat, **Type** MAR1, **Creator** ????.

Note: Marriages are listed in the order added. Since the simulation picks a group of brides randomly each spring, order in the list is not important.

Support Program Information:

Name Marriage2 editor, **Type** APPL, **Creator** SIMU.

Compiled using: Sunking/4.0/exec.text
Source code: Sunking/4.0/marriage2.text (Pascal code)
Resource code: Sunking/4.0/marriage2r.text (resource file).
Runs on: Macintosh

Action: This program generates a list of brides for the second generation. Included are the names of the brides, their age at the start of the second generation in Spring 1676, their prestige category, and their father's office.

DataFile Information:

Name marriage2.dat, **Type** MAR1, **Creator** ????.

Note: Marriages are listed in the order added. Since the simulation picks a group of brides randomly each spring, order in the list is not important.

<u>Final Statistics</u>

Support Program Information:

Name examine, **Type** APPL, **Creator** SIMU.

Compiled using: Sunking/4.0/exec.text
Source code: Sunking/4.0/examine.text (Pascal code)
Resource code: Sunking/4.0/examiner.text (resource file).
Runs on: Macintosh

Action: This program reads from the file **Final Stats**, which is updated by the simulation each time a game is successfully finished or quit. This file holds standard information for each game played with the diskette. This information includes the date on which the simulation was played, the final prestige attained, the final date in the game, and the student's final wealth. It also tells whether or not the game was completed, i.e., if the student reached the third generation.

DataFile Information:

Name Final Stats, **Type** ENDS, **Creator** ????.

Note: A game can be listed **not complete** if the student did not have a son in either the first or second generation, and thus did not leave an heir, or if the student bought a lease that led to peasant unrest and an untimely hanging.

Text files

Support Program Information:

A Macintosh text editor, Set File

Compiled using: n/a

Source code: n/a

Resource code: n/a

Runs on: Macintosh

Action: Programmers should use a text editor and the utility program Set File to generate the text for the files **Instructions** and **StartText**. These are text only files. If MacWrite or MS Word are used, the file must be saved as text only (also called ASCII only). If Edit (a programmer's text editor) or File (from the Lisa Workshop example listings) are used, the document will be saved as text only. When the text has been saved to diskette, the document holding the files should be modified with Set File or another resource editor to change the File Type to **CORR** and File Creator to **JUNK**. The main program will read only text files with these settings.

DataFile Information:

Name Instructions, **Type** CORR, **Creator** JUNK.

Name StartText, **Type** CORR, **Creator** JUNK.

Pictures

Support Program Information:

Thunderscan™, MacPaint, MacDraw, and ScreenMaker.

Compiled using: n/a

Source code: n/a

Resource code: n/a

Runs on: Macintosh

Action: Two data files provide pictures for display during the simulation. They are digitized using Thunderscan™, then cleaned up with MacPaint and MacDraw. The **StartupScreen** is created with the programmer's utility ScreenMaker, which changes a MacPaint picture into a screen display that appears when the Macintosh is powered on. The file **Pictures** contains two images, a picture of a Father and Son, and a picture of a Beggar. These are MacDraw objects pasted into an empty System Scrapbook. The **Scrapbook File** is renamed **Pictures** on the Macintosh desktop and the pictures are accessed as external resources by the main program.

DataFile Information:

Name StartupScreen, **Type** SCAN, **Creator** NONE.

Name Pictures, **Type** ZSYS, **Creator** MACS.

Notes: The **Pictures** file is referenced by resource ID. In the main program, the Father and Son image is opened by the procedure DoPicture with a call to the constant FIRSTBORN (-32768), which is the resource ID number of that image. Similarly, the Beggar image is opened by the procedure DoPicture with a call to the constant DEBTOR (-32767), which is the resource ID number of that image. These will be the default IDs assigned if the images are pasted into an empty scrapbook from MacDraw; paste Father and Son first, then Beggar, as the first image pasted into the scrapbook gets the lower ID number.

Images are digitized from woodcuts by Jacques Callot (1592-1635 A.D.) published with accompanying English text in a collection of "Callot's Etchings" (ed. Howard Daniel, Dover Publications, Inc., New York, 1974).

Saved Games

Support Program Information:

No support program required.

Compiled using: n/a
Source code: n/a
Resource code: n/a
Runs on: Macintosh

Action: The file **Saved Simulation** is created when the player chooses to Save the game. The state of the simulation is stored using variables DATE and ASSETS. This file can be read by the program to restore a previously saved simulation.

DataFile Information:

Name Saved Simulation, **Type** SAVE, **Creator** ????.

Notes: This file is created during main program execution, and is readable by the main program using the **Restore** command on the **Progression** menu.

Support Program Technical Details

Getting Started

Source code for the support programs is on the diskette **Data Files and Editors**, along with the compiled applications and associated data files. Double clicking on an editor executes it and opens the data file associated with it.

Structure

All the support programs have the following basic general structure: a data structure of type record defines the attributes of the particular thing to be stored (e.g., the name of the office, its cost, its prestige). This record also has a boolean field named DELETED. Data is stored in an array of records. Some initialization procedures set up the window and menu.

Operation

The menu allows the user to ADD, DELETE, SAVE, and QUIT. All the existing data is then loaded from the appropriate data file, the DELETED fields of these records are set to false, and the data are displayed in the window.

The ADD procedure allows the user to add more data to the end of the list. It creates a dialog window for input and then adds data to the array.

The DELETE procedure allows an entry in the data list to be deleted. It brings up a dialog window asking for the number of the item to be deleted. Each item has a number that is displayed in the main window. The DELETED field of that record in the array is then set to true. Other items are renumbered when deletion is complete, and the list redisplayed.

The SAVE procedure writes all the undeleted records (excluding the DELETED field) into a file. The main event loop works as in other Macintosh programs to process menu choices.

If the user chooses to QUIT, then the program ends. Data is not saved by the QUIT command!

The Mail Editor differs in that it allows editing of existing data items, the insertion of new data items at any point in the list, and does not redisplay the (relatively long) list of data unless prompted.

Protecting data files

The support programs need not (and *should not*) appear on the student's diskette. The editors are meant for programmers' use to generate data and should not be given to the students. The data files they generate (four mail files, office.dat, lease.dat, marriage1.dat, marriage2.dat), however, are *required* for the program to function properly. Early versions of the simulation made these files **invisible** using the **Set File** program so that the students were not confused or able to delete them by accident. Later versions, including the Academic Courseware Exchange version 4.1, locked the data files in desktop folders to prevent accidental erasure or removal of the files.

```
            FIRST := 50;
          end; (* If *)
        if (Noble) and ((Land.Vicomte > NONE) or (Land.Marquisat > NONE)) then begin
            FIRST := 60;
        end; (* If *)
      end; (* If *)
      if (Prestige >= 70) and (PresFallen) then begin
          FIRST := 60;
      end; (* If *)
      {SECOND calculates the one's digit of the Player's Prestige}
      SECOND := NONE;
      SECOND := SECOND - Lease.NumBought;
      SECOND := SECOND + (Land.Bought div 100);
      if Marriage.Married then begin
          SECOND := SECOND + 1;
      end; (* If *)
      if Marriage.MarrBelow then begin
          SECOND := SECOND - 3;
      end; (* If *)
      SECOND := SECOND - 2 * Marriage.Failures;
      SECOND := SECOND - 2 * TooAmbitious;
      if BoughtLetter then SECOND := SECOND + 2;
      if Protector.NumFailures <= 1 then PROCTSHAMEPENALTY := Protector.NumFailures
                                    else PROCTSHAMEPENALTY := 3;
      SECOND := SECOND - PROCTSHAMEPENALTY;
      if (FIRST >= STARTPRESTIGE) and (Prestige < 70) then begin
        AddOffPrestige(SECOND, STARTPRESTIGE, Office.OfficeList);
      end else begin
          if FIRST >= 70 then begin
            AddOffPrestige(SECOND, FIRST, Office.OfficeList);
          end; (* If *)
      end; (* If *)
      if Land.Marquisat > NONE then begin
          SECOND := SECOND + 9;
      end; (* If *)
      if (not Will.WasInAccord) then begin
          SECOND := SECOND - WILLPRESRATING;
      end; (* If *)
      if SECOND > 9 then begin
          SECOND := 9;
      end; (* If *)
      Prestige := FIRST + SECOND;
      if Prestige < AMBITIOUS then begin
          Prestige := AMBITIOUS;
      end; (* If *)
      if (FIRST > AMBITIOUS) and (Prestige < NOOFFPRESTIGE) then begin
          Prestige := NOOFFPRESTIGE;
      end; (* If *)
    end; (* With *)
end; (* CalcPrestige *)

(*****************************************************************************)
```

```
function CalcRenteVal (Rente:RenteRec) : longint;

(* CALLED BY: CalcTotalVal, DispWealth                                          *)
(* CALLS TO: none                                                               *)
(* GLOBALS: Assets                                                              *)
(* ACTION: This function counts the number of King's Rentes held by the player and multiplies by  *)
(* 1000 for total value.                                                        *)

var  NUM : longint;
   MARKER : RenteHandle;

begin (* CalcRenteVal *)
   with Assets do begin
      MARKER := Rente.IndivRentes;
      NUM := 0;
      while MARKER <> nil do begin
         NUM := NUM + 1;
         MARKER := MARKER^^.Next;
      end; (* While *)
      CalcRenteVal := NUM * 1000;
   end; (* With *)
end; (* CalcRenteVal *)

(*****************************************************************************)
procedure CalcTotalVal(var Assets:AssetsType);

(* CALLED BY: DisplayAssets                                                     *)
(* CALLS TO: CalcRenteValue                                                     *)
(* GLOBALS: Assets                                                              *)
(* ACTION: This procedure computes the total wealth of the player. It computes the number of Rentes  *)
(* the player own and returns the face value of King's Rentes held. It then calculates the value of the  *)
(* player's land. To these values are added the player's cash, grain value, amount invested in textiles, and  *)
(* the amount paid for offices.                                                 *)

var  RENTEVAL : longint;

begin (* CalcTotalVal *)
   with Assets do begin
      RENTEVAL := CalcRenteVal(Rente);
      Land.Value := ((Land.Inherited + Land.Bought) * LANDVAL) + (Land.Seigneurie * SEIGCOST) +
                  (Land.Vicomte * VICOMTECOST) + (Land.Marquisat * MARQCOST);

      TotalVal := Cash + (Grain * Land.Price) + Land.Value + Textiles + Office.TotPurchase + RENTEVAL;
   end; (* With *)
end; (* CalcTotalVal *)

(*****************************************************************************)
procedure GetFactors(var Mult, Percent:longint; Prestige:longint);

(* CALLED BY: CalcCostofLiving                                                  *)
(* CALLS TO: none                                                               *)
(* GLOBALS: none                                                                *)
```

```
(* ACTION: This procedure is used when computing the cost of living. Mult is the amount by which the   *)
(* player's highest salary is mutiplied to give an initial cost of living. Percent gives the percentage of the   *)
(* other salaries that are added in to give rest of the basic cost of living.  NOTE:  these percentages   *)
(* change as the prestige rate climbs, which changes cost of living dramatically!                         *)

begin (* GetFactors *)
   case (Prestige div 10) of
      2, 3, 4 : begin
                  Mult := 1;
                 Percent := 25;
               end; (* 4 *)
      5 : begin
            Mult := 2;
           Percent := 50;
         end; (* 5 *)
      6 : begin
            Mult := 3;
           Percent := 50;
         end; (* 6 *)
      7 : begin
            Mult := 4;
           Percent := 50;
         end; (* 7 *)
      8, 9 : begin
            Mult := 8;
           Percent := 50;
         end; (* 9 *)
   end; (* Case *)
end; (* GetFactors *)

(*****************************************************************************************)
procedure GetHighestOff(OfficeList:OfficeHandle; var Off:DlogOffRec);

(* CALLED BY: CalcCostofLiving                                                            *)
(* CALLS TO: none                                                                         *)
(* GLOBALS: none                                                                          *)
(* ACTION: This procedure returns the salary and the title of the office with the highest salary.   *)

begin (* GetHighestOff *)
   Off.Value := NONE;
   while OfficeList <> nil do begin
      if OfficeList^^.Salary > Off.Value then begin
         Off.Value := OfficeList^^.Salary;
         Off.Title := OfficeList^^.Title;
       end; (* If *)
      OfficeList := OfficeList^^.Next;
   end; (* While *)
end; (* GetHighestOff *)

(*****************************************************************************************)
procedure CalcCostOfLiving(var Assets:AssetsType);
```

```
(* CALLED BY: DisplayAssets, SellOffice                                          *)
(* CALLS TO: GetFactors, GetHighestOffice                                        *)
(* GLOBALS: Assets                                                               *)
(* ACTION: This procedure computes the player's cost of living. It first calculates the base cost. This is  *)
(* based on the player's offices. If the player has no offices, the base cost is BASECOSTOFL with no  *)
(* titled land, VCOSTOFL with a Vicomté, or MCOSTOFL with a Marquisat. If offices are owned,  *)
(* procedure GetFactors is called to get the multipliers for the offices. The base cost is then computed  *)
(* by multiplying the highest salary by MULT and then adding in PERCENT percent of the rest of the  *)
(* salaries. If the player is not married, then the cost of living is 40% of this. The cost of living is then  *)
(* increased 20% by each kid.                                                    *)

var  MULT, PERCENT, KIDSPERCENT : longint;
   OFF : DlogOffRec;
   MARKER : OfficeHandle;
   I : integer;

begin (* CalcCostOfLiving *)
   with Assets do begin

        (* Sets base cost of living*)
          CostOfLiving := BASECOSTOFL;

          if land.vicomte <> 0 then begin
          Costofliving := VCOSTOFL;
         end; (* If *)

         if land.marquisat <> 0 then begin
          Costofliving := MCOSTOFL;
         end;

       (* Calculates the prestige factors and adds in percentage of offices to c. o. l. *)
         if (Prestige div 10) = (AMBITIOUS div 10) then begin
             GetFactors(MULT, PERCENT, OldPrestige);
           end else begin
             GetFactors(MULT, PERCENT, Prestige);
         end; (* If *)
         GetHighestOff(Office.OfficeList, OFF);
          CostOfLiving := MULT * OFF.VALUE;
         MARKER := Office.OfficeList;
          while MARKER <> nil do begin
             if MARKER^^.Title <> OFF.Title then begin
                CostOfLiving := CostOfLiving + ((MARKER^^.Salary * PERCENT) div 100);
             end; (* If *)
             MARKER := MARKER^^.Next;
         end; (* While *)

      (* Calculates the number of children and adds a percentage to c. o. l. for each child *)
      if not Marriage.Married then begin
          CostOfLiving := ((CostOfLiving * COLMARRIAGEFACTOR) div 100);
      end; (* If *)
      KIDSPERCENT := COLKIDSFACTOR * Children.Number;
      CostOfLiving := CostOfLiving + ((CostOfLiving * KIDSPERCENT) div 100);
```

```
        if CostOfLiving <= BASECOSTOFL then begin
            CostOfLiving := BASECOSTOFL;
        end; (* If *)
    end; (* With *)
end; (* CalcCostOfLiving *)

(*****************************************************************************************)
procedure DisplayAssets (var Assets:AssetsType; Date:DateType);

(* CALLED BY: Bankrupt, CheckDebt, NextCorr, EndSimulaton, ManageLand, BuyOffice, Treasury,    *)
(*                  DoCommand, Main                                                            *)
(* CALLS TO: CalcTotalVal, CalcPrestige, CalcCostOfLiving                                      *)
(* GLOBALS: AssetWindow, Assets, Date                                                          *)
(* ACTION: This procedure displays the date, the season, the player's status and the player's total wealth.  *)

var  DISPRECT : Rect;

begin (* DisplayAssets *)
    CalcTotalVal(Assets);
    CalcPrestige(Assets);
    CalcCostOfLiving(Assets);
    SetPort(AssetWindow);
    PLSetWrPort(AssetWindow);
     SetRect(DISPRECT, 0, 0, 114, 120);
    EraseRect(DISPRECT);
     MoveTo(0,30);
     if Date.Fall then begin
         write(' Fall, ');
    end else begin
         write(' Spring, ');
    end; (* If *)
    writeln(Date.Year:0);
     writeln;
     writeln('Age:  ',Assets.Age:0);
     writeln('Prestige:     ',Assets.Prestige:0);
     writeln('Total Wealth: ');
     writeln('£', Assets.TotalVal:0);
     write('Cash: £',Assets.Cash:0);
end; (* DisplayAssets *)

{$S Seg1}

(*****************************************************************************************)
procedure InitVars (var Assets:AssetsType; var Date:DateType);

(* CALLED BY: Initialize, Bankrupt                                                             *)
(* CALLS TO: none                                                                              *)
(* GLOBALS: Assets, Date                                                                       *)
(* ACTION: This procedure initializes program variables as required.                           *)

var  TEMPOFFICE : OfficeRec;
    I : integer;
```

```
begin (* InitVars *)
    Assets.Land.ShareCrop := 0;
    Assets.Land.Kind := 0;
    Assets.Land.Rent := 0;
    Assets.Land.Inherited := 42;
    Assets.Land.Seigneurie := NONE;
    Assets.Land.Vicomte := NONE;
    Assets.Land.Marquisat := NONE;
    Assets.Land.Lost := 0;
    Assets.Generation := FIRSTGEN;
    Date.Fall := false;
    Assets.Grain := 0;
    Assets.Cash := 5054;
    Assets.Land.Bought := 0;
    Date.Year := STARTYEAR - 1;
    Assets.Textiles := 0;
    Assets.Rente.IndivRentes := nil;
    Assets.Debt := 0;
    Assets.Lease.FaceValue := 0;
    Assets.Lease.Hanged := false;
    Assets.Rente.Owe := 0;
    Assets.Rente.SoldVal := 0;
    Assets.Rente.CostDenier := 11;
    Assets.Lease.GotThisYear := 0;
    Assets.Lease.NumBought := NONE;
    Assets.Office.Salary := 2000;
    Assets.Office.TotPurchase := 21000;
    Assets.Office.Number := 1;
    TEMPOFFICE.Title := 'Auditeur en la Chambre des Comptes de Rouen';
    TEMPOFFICE.Value := 21000;
    TEMPOFFICE.Salary := 2000;
    TEMPOFFICE.AmtPaid := 21000;
    TEMPOFFICE.Prestige := STARTPRESTIGE;
    TEMPOFFICE.Next := nil;
    Assets.Office.OfficeList := Pointer(NewHandle(SizeOf(TEMPOFFICE)));
    Assets.Office.OfficeList^^ := TEMPOFFICE;
    Assets.Office.Levied := false;
    Assets.Office.Raise := false;
    Assets.Age := STARTAGE;
    Assets.Marriage.Married := false;
    Assets.Marriage.Available.IsAVail := true;
    Assets.Marriage.Failures := 0;
    Assets.Marriage.MarrBelow := false;
    Assets.Children.Number := 0;
    Assets.Children.NumBoys := 0;
    Assets.Children.NumGirls := 0;
    Assets.Children.Boys := nil;
    Assets.Children.Girls := nil;
    Assets.Children.NextBirth.Year := 0;
    Assets.Will.Made := false;
    Assets.Will.WasInAccord := true;
```

```
    for I := 1 to NUMWILLCATEGORIES do begin
        Assets.Will.Distribution[I] := NONE;
    end; (* For *)
    Assets.Noble := false;
    Assets.TooAmbitious := NONE;
    Assets.PresFallen := false;
    Assets.CostOfLiving := NONE;
    Assets.Won := false;
    Assets.Quit := false;
    Assets.SoldGrain := false;
    Assets.BoughtLetter := false;
    Assets.ChoseProct := true;
    Assets.Protector.Name := Cornuel;
    Assets.Protector.YearFail := NONE;
    Assets.Protector.NumFailures := NONE;
    Assets.Protector.ThisProctFail := NONE;
end; (* InitVars *)

{$S Seg11}

(*****************************************************************************************)
procedure SetUpTextEdit(var hTE:TEHandle; TextWindow:WindowPtr);

(* CALLED BY: BeginText, Setup, Main                                                     *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: hTE                                                                          *)
(* ACTION: This procedure sets us the text portion of dialog windows.                    *)

begin (* SetUpTextEdit *)
    hTE := TENew(TextWindow^.portRect, TextWindow^.portRect);
    hTE^^.ViewRect.Left := hTE^^.ViewRect.Left + LEFTDIFF;
    hTE^^.ViewRect.Right := hTE^^.ViewRect.Right - BARDIFF;
    hTE^^.DestRect.Left := hTE^^.DestRect.Left + LEFTDIFF;
    hTE^^.DestRect.Right := hTE^^.DestRect.Right - BARDIFF;
    hTE^^.ViewRect.Bottom := hTE^^.ViewRect.Bottom - BARDIFF;
    hTE^^.DestRect.Bottom := hTE^^.DestRect.Bottom - BARDIFF;
end; (* SetUpTextEdit *)

(*****************************************************************************************)
procedure GetText(FileName:Str13; var hTE:TEHandle; VRefNum:integer);

(* CALLED BY: BeginText, ReadText                                                        *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: VRefNum, hTE                                                                 *)
(* ACTION: This procedure reads text from a file on disk.                                *)

Var  REFNO, IO : integer;
    LOGEOF: LongInt;

begin (* GetText *)
    IO := FSOpen (FileName, VRefNum, REFNO);
    IO := GetEOF (REFNO, LOGEOF);
```

```
    SetHandleSize (hTE^^.hText, LOGEOF);
    IO := FSRead (REFNO, LOGEOF, hTE^^.hText^);
    IO := FSClose (REFNO);
    hTE^^.teLength := LOGEOF;
    TESetSelect(0, 0, hTE);
    TECalText(hTE);
end; (* GetText *)

{$S Seg1}

(*****************************************************************************************)
procedure BeginText (VRefNum:integer);

(* CALLED BY: Initialize                                                                 *)
(* CALLS TO:GetText, SetUpTextEdit                                                       *)
(* GLOBALS: VRefNum                                                                      *)
(* ACTION: This procedure reads text from the StartText file and displays it in a dialog window.  *)

var  WREC : WindowRecord;
    TEXTWINDOW : WindowPtr;
    hTE : TEHandle;
    TEMP : boolean;
    ANEVENT : EventRecord;

begin (* BeginText *)
    TEXTWINDOW := GetNewWindow(260, @WREC, Pointer(-1));
    HideCursor;
    SetPort(TEXTWINDOW);
    PLSetWrPort(TEXTWINDOW);
    SetUpTextEdit(hTE, TEXTWINDOW);
    hTE^^.ViewRect.Bottom := hTE^^.ViewRect.Bottom + 1;
    hTE^^.DestRect.Bottom := hTE^^.DestRect.Bottom + 1;
    GetText('StartText', hTE, VRefNum);
    hTE^^.txFont := NewYork;
    hTE^^.txSize := 10;
    TESetJust(teJustCenter, hTE);
    TECalText(hTE);
    TEUpdate (TEXTWINDOW^.portRect, hTE);
    repeat
        SystemTask;
       TEMP := GetNextEvent(everyEvent, ANEVENT);
    until button;
    CloseWindow(TEXTWINDOW);
    TEDispose(hTE);
    ShowCursor;
end; (* BeginText *)

(*****************************************************************************************)
procedure Initialize (var FinWindow, AssetWindow:WindowPtr; var Icons:IconType; var myMenus:MenuArray;
                var Assets:AssetsType; var Date:DateType; var WatchHdl:CursHandle; var hTE:TEHandle;
                  var Corrfile:STR255; var CorrRefNum, VRefNum:integer);
```

```
(* CALLED BY: Main                                                              *)
(* CALLS TO: GetIcons, DrawIcons, SetupMenus, Initvars                          *)
(* GLOBALS: FinWindow, AssetWindow, FWRec, AWRec, Icons, CorrRefNum, AppResFile,  *)
(* VRefNum, myMenus, Assets, Date, hTE, WatchHdl, Corrfile                      *)
(* ACTION: This procedure initializes the various system managers, sets up the menus and windows,  *)
(* draws the financial icons, and initializes the various program variables.    *)

var  ERR, DUMMYINT : integer;
     SIZE : longint;
     VOLNAME : Str255;

begin (* Initialize *)
     MaxApplZone;
     MoreMasters;
     MoreMasters;
     MoreMasters;
     MoreMasters;
     InitGraf(@thePort);
     AppResFile := CurResFile;
     randSeed := TickCount;
     InitFonts;
     FlushEvents(everyEvent, 0);
     InitWindows;
     TEInit;
     InitDialogs(nil);
     InitMenus;
     InitCursor;
     WatchHdl := GetCursor(4);
     HNoPurge(Pointer(WatchHdl));
     ERR := GetVol (@VOLNAME, VRefNum);
     BeginText (VRefNum);
     FlushEvents(everyEvent, 0);
     FinWindow := GetNewWindow(256, @FWRec, Pointer(-1));
     AssetWindow := GetNewWindow(258, @AWRec, Pointer(-1));
     SetUpMenus(myMenus);
     GetIcons(Icons);
     DrawIcons(Icons, FinWindow);
     InitVars(Assets, Date);
     CORRFILE:='FirstMail.1.dat';
     ERR := FSOpen(CORRFILE, 0, CorrRefNum);
     SIZE := SizeOf(integer);
     ERR := FSRead(CorrRefNum, SIZE, @DUMMYINT);
     SIZE := SizeOf(MailRec);
     ERR := FSRead(CorrRefNum, SIZE, @Assets.Mail);
end; (* Initialize *)

{$S}

(*****************************************************************************************)
procedure HiliteIcon (Icons:IconType; IconNum:integer);

(* CALLED BY: SelectIcon                                                        *)
```

```
(* CALLS TO: none                                                                    *)
(* GLOBALS: Icons                                                                   *)
(* ACTION: This procedure hilightes an icon when the player selects that icon by clicking it. It does this  *)
(* by erasing the icon, plotting the icon, and then drawing the rectangle that the icon fits in.            *)                    *)

begin (* HiliteIcon *)
   EraseRect(Icons.Defs[IconNum].IconRect);
   PlotIcon(Icons.Defs[IconNum].IconRect, Icons.Defs[IconNum].IconHdl);
    PenSize(2,2);
   FrameRect(Icons.Defs[IconNum].IconRect);
    PenNormal;
end; (* HiliteIcon *)

(*****************************************************************************************)
procedure UnHiliteIcon (Icons:IconType; IconNum:integer);

(* CALLED BY: SelectIcon                                                            *)
(* CALLS TO: none                                                                   *)
(* GLOBALS: Icons                                                                   *)
(* ACTION: This procedure unhilights an icon when the player has selected another investment choice.  *)
(* It does this by erasing the rectangle that the icon sits in and then plotting the icon.           *)

begin (* UnHiliteIcon *)
   EraseRect(Icons.Defs[IconNum].IconRect);
   PlotIcon(Icons.Defs[IconNum].IconRect, Icons.Defs[IconNum].IconHdl);
end; (* UnHiliteIcon *)

(*****************************************************************************************)
procedure PrintChoice(Choice:integer; SelRect:Rect);

(* CALLED BY: SelectIcon                                                            *)
(* CALLS TO: none                                                                   *)
(* GLOBALS: none                                                                    *)
(* ACTION: This procedure prints out the name of the investment represented by the selected icon.  It  *)
(* erases the old name and prints the correct one, as determined by the parameter Choice.             *)

begin (* PrintChoice *)
   EraseRect(SelRect);
   FrameRect(SelRect);
   MoveTo(30, 170);
   case Choice of
      LANDICON : DrawString('  Land');
      RENTEICON : DrawString('  Rente');
      OFFICEICON : DrawString('  Office');
      LEASEICON : DrawString('  Lease');
      TEXTILEICON : DrawString(' Textile');
   end; (* Case *)
end; (* PrintChoice *)

(*****************************************************************************************)
function BoughtRente(Rentes:RenteHandle; Date:DateType) : boolean;
```

```
(* CALLED BY: Not called in this version of the Gentleman; reserved for more complex transactions      *)
(* at later date.                                                                                      *)
(* CALLS TO: none                                                                                      *)
(* GLOBALS: Date                                                                                       *)
(* ACTION: This function determines whether or not the player bought a Rente this year. It goes to the *)
(* end of the list of Rentes and checks to see if that Rente was bought at the current date. If so, the *)
(* function returns TRUE. Otherwise it returns FALSE.                                                  *)

begin (* BoughtRente *)
   BoughtRente := false;
   if Rentes <> nil then begin
      while Rentes^^.Next <> nil do begin
         Rentes := Rentes^^.Next;
      end; (* While *)
      if (Rentes^^.Year = Date.Year) and (Rentes^^.Fall = Date.Fall) then begin
         BoughtRentes := true;
      end; (* If *)
   end; (* If *)
end; (* BoughtRente *)

(*****************************************************************************************)
procedure En_Disable (Date:DateType; FinMenu:MenuHandle; var Icons:IconType; var Assets:AssetsType);

(* CALLED BY: SelectIcon, GoToNext                                                                     *)
(* CALLS TO: none                                                                                      *)
(* GLOBALS: Icons, Assets, Date                                                                        *)
(* ACTION: This procedure controls which items in the investment menu are to be enabled, and thus      *)
(* hilighted, based on which investment choice has been selected. It first enables all the item in the *)
(* investment menu, and then enables the menu itself, if necessary. It then unhilights the appropriate *)
(* choices, depending on whether it is Spring or Fall, and on which investment was selected. For example, *)
(* leases can never be sold, and so the SELL choice is disabled whenever leases are chosen.            *)

var I : integer;

begin (* En_Disable *)
   EnableItem(FinMenu, STATUSITEM);
   EnableItem(FinMenu, MANAGEITEM);
   EnableItem(FinMenu, BUYITEM);
   EnableItem(FinMenu, SELLITEM);
   if (Icons.MenuDisabled) then begin
      EnableItem(FinMenu, 0);
      DrawMenuBar;
      Icons.MenuDisabled := false;
   end; (* If *)
   case Date.Fall of
      true :
         case Icons.Selected of
            LANDICON : if (Assets.Land.Bought = 0) and (Assets.Grain = 0) then begin
                           DisableItem(FinMenu, SELLITEM);
                       end; (* If *)

            RENTEICON : DisableItem(FinMenu, MANAGEITEM);
```

```
            OFFICEICON : begin
                           DisableItem(FinMenu, MANAGEITEM);
                           if Assets.Office.OfficeList = nil then begin
                              DisableItem(FinMenu, SELLITEM);
                           end; (* If *)
                         end; (* OFFICEICON *)

            LEASEICON : begin
                           DisableItem(FinMenu, MANAGEITEM);
                           DisableItem(FinMenu, SELLITEM);
                           DisableItem(FinMenu, BUYITEM);
                         end; (* LEASEICON *)

            TEXTILEICON : begin
                           DisableItem(FinMenu, MANAGEITEM);
                           DisableItem(FinMenu, SELLITEM);
                         end; (* TEXTILEICON *)
          end; (* case *)

        false :
          case Icons.Selected of
            LANDICON : if (Assets.Land.Bought = 0) and (Assets.Grain = 0) then begin
                           DisableItem(FinMenu, SELLITEM);
                         end; (* If *)

            RENTEICON : DisableItem(FinMenu, MANAGEITEM);

            OFFICEICON : begin
                           DisableItem(FinMenu, MANAGEITEM);
                           if Assets.Office.OfficeList = nil then begin
                              DisableItem(FinMenu, SELLITEM);
                           end; (* If *)
                         end; (* OFFICEICON *)

            LEASEICON : begin
                           DisableItem(FinMenu, MANAGEITEM);
                           DisableItem(FinMenu, SELLITEM);
                           if Assets.Lease.Bought then begin
                              DisableItem(FinMenu, BUYITEM);
                           end; (* If *)
                         end; (* LEASEICON *)

            TEXTILEICON : begin
                           DisableItem(FinMenu, BUYITEM);
                           DisableItem(FinMenu, MANAGEITEM);
                           DisableItem(FinMenu, SELLITEM);
                         end; (* TEXTILEICON *)
          end; (* Case *)
      end; (* Case *)
  end; (* En_Disable *)
```

```
(*****************************************************************************************)
procedure SelectIcon(var Icons:IconType; myEvent:EventRecord; myMenus:MenuArray; Date:DateType; var Assets:AssetsType);

(* CALLED BY: Main                                                                       *)
(* CALLS TO: UnHiliteIcon, PrintChoice                                                   *)
(* GLOBALS: Icons, MyEvent, myMenus, Assets, Date                                        *)
(* ACTION: This procedure is called when the player pushes the button in the window that contains the  *)
(* financial icons. It determines which icon was selected by finding which icon's rectangle contains the  *)
(* point of the mouse-down event. If an icon was selected, if then unhilights the previously selected icon,  *)
(* if there was one, and hilights the new icon, if it wasn't the currently selected icon. If this was the  *)
(* first time an icon was selected, it enables the investment menu and displays the choice.  *)

var I : integer;

begin (* SelectIcon *)
   GlobalToLocal(myEvent.Where);
   I := 1;
   while (not PtInRect(myEvent.Where, Icons.Defs[I].IconRect)) and (I < NUMICONS) do begin
      I := I + 1;
   end; (* While *)
   if PtInRect(myEvent.Where, Icons.Defs[I].IconRect) then begin
      if not (Icons.Selected in [NONE, I]) then begin
         UnHiliteIcon(Icons, Icons.Selected);
         PrintChoice(I, Icons.ChoiceRect);
      end; (* if *)
      if Icons.Selected <> I then begin
         HiliteIcon(Icons, I);
      end; (* If *)
      if not Icons.IconWasSelected then begin
         Icons.IconWasSelected := true;
         Icons.MenuDisabled := false;
         PrintChoice(I, Icons.ChoiceRect);
         EnableItem(myMenus[FINMENU], 0);
         DrawMenuBar;
      end; (* If *)
      Icons.Selected := I;
      En_Disable(Date, myMenus[FINMENU], Icons, Assets);
   end; (* If *)
end; (* SelectIcon *)

{$S Seg7}

(*****************************************************************************************)
procedure Bankrupt(var Assets:AssetsType; Date:DateType; Really, ShowPic:boolean);

(* CALLED BY: CheckDebt, NextProct, LoadSimulation                                       *)
(* CALLS TO: DoPicture, DisplayAssets, InitVars                                          *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure is called when the player is thrown into bankruptcy. Every financial asset  *)
(* the player owns is removed except for miscellaneous land gained through inheritance.  *)

var OLDDATE : DateType;
```

```
    OLDAGE, OLDGEN, OLDAMBITION, OLDLEASENUM, OLDSHARE, OLDRENT,
            OLDKIND, OLDLAND : longint;
    RMARKER, RDISPMARK : RenteHandle;
    OMARKER, ODISPMARK : OfficeHandle;
    OLDPROCT : ProctRec;
    LEN : integer;
    KIDS : ChildRec;
    WIFE : MarrType;
    OLDFALLEN, OLDNOBLE, OLDLETTER : boolean;
    OLDWILL : WillType;

begin (* Bankrupt *)
    if Really then begin
        LEN := StopAlert(271, nil);
    end; (* If *)
    with Assets do begin
        OLDDATE := Date;
        OLDNOBLE := Noble;
        OLDPROCT := Protector;
        OLDAGE := Age;
        KIDS := Children;
        WIFE := Marriage;
        OLDGEN := Generation;
        OLDAMBITION := TooAmbitious;
        OLDLEASENUM := Lease.NumBought;
        OLDSHARE := Land.ShareCrop;
        OLDRENT := Land.Rent;
        OLDKIND := Land.Kind;
        OLDLAND := Land.Inherited;
        OLDFALLEN := PresFallen;
        OLDLETTER := BoughtLetter;
        OLDWILL := Will;
        RMARKER := Rente.IndivRentes;
        while RMARKER <> nil do begin
            RDISPMARK := RMARKER;
            RMARKER := RMARKER^^.Next;
            DisposHandle(Pointer(RDISPMARK));
        end; (* While *)
        OMARKER := Office.OfficeList;
        while OMARKER <> nil do begin
            ODISPMARK := OMARKER;
            OMARKER := OMARKER^^.Next;
            DisposHandle(Pointer(ODISPMARK));
        end; (* While *)
        InitVars(Assets, Date);
        Date := OLDDATE;
        Noble := OLDNOBLE;
        Protector := OLDPROCT;
        Age := OLDAGE;
        Children := KIDS;
        Marriage := WIFE;
        Cash := 0;
```

```
        Generation := OLDGEN;
        TooAmbitious := OLDAMBITION;
        Lease.NumBought := OLDLEASENUM;
        Land.ShareCrop := OLDSHARE;
        Land.Rent := OLDRENT;
        Land.Kind := OLDKIND;
        Land.Inherited := OLDLAND;
        PresFallen := OLDFALLEN;
        BoughtLetter := OLDLETTER;
        Will := OLDWILL;
        Office.Salary := 0;
        Office.TotPurchase := 0;
        Office.Number := 0;
        DisposHandle(Pointer(Office.OfficeList));
        Office.OfficeList := nil;
        DisplayAssets(Assets, Date);
    end; (* With *)
    if ShowPic then begin
        DoPicture(DEBTOR);
    end; (* If *)
end; (* Bankrupt *)

(*****************************************************************************************)
procedure CalcHarvest(var Harvest:HarvestType);

(* CALLED BY: GoToNext, Main                                                             *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure determines what type of harvest there was. It calculates a random number  *)
(* from one to 100 and then uses that number to determine what the harvest is, based on percentages  *)
(* defined in constants.                                                                 *)

var  TEMP : integer;

begin (* CalcHarvest *)
    TEMP := abs(Random) div RANDDIVIDER;
    if TEMP <= BLIGHTRAND then Harvest := Blight
    else if TEMP <= POORRAND then Harvest := Poor
    else if TEMP <= GOODRAND then Harvest := Good
    else if TEMP <= EXCELRAND then Harvest := Excellent;
end; (* CalcHarvest *)

(*****************************************************************************************)
procedure NextLand(var Assets:AssetsType; Date:DateType);

(* CALLED BY: GoToNext                                                                   *)
(* CALLS TO: NumSpecs                                                                    *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure calculates the cash value of a quintel of grain. In the fall, it finds the yield on  *)
(* the land a player is renting in kind and sharecropping, and how much cash was gained by renting for  *)
(* cash.                                                                                 *)
```

```
var  TOTLAND, GIFT : longint;
    LEN : integer;
    GIFTSTR : STR255;

begin (* NextLand *)
   with Assets.Land, Date do begin
      case Local of
         Blight : Yield := BLIGHTYIELD;
         Poor : Yield := POORYIELD;
         Good : Yield :=  GOODYIELD;
         Excellent : Yield := EXCELYIELD;
      end; (* Case *)
      case Fall of
         true :
            case Regional of
               Blight : Price := FBLIGHTPRICE;
               Poor : Price := FPOORPRICE;
               Good : Price := FGOODPRICE;
               Excellent : Price := FEXCELPRICE;
            end; (* Case *)
         false :
            case Regional of
               Blight : Price := SBLIGHTPRICE;
               Poor : Price := SPOORPRICE;
               Good : Price := SGOODPRICE;
               Excellent : Price := SEXCELPRICE;
            end; (* Case *)
      end; (* Case *)
      if Fall then begin
         Assets.SoldGrain := false;
         TOTLAND := Bought + Inherited + Seigneurie + Vicomte + Marquisat;
         Assets.Cash := Assets.Cash + (((Rent * TOTLAND) div 100) * RENTVALUE);
         Lost := (Assets.Grain * GRAINLOSS) div 100;
         Assets.Grain := Assets.Grain + (Yield * ((ShareCrop * TOTLAND) div 100))
                              + (KINDRENT * ((Kind * TOTLAND) div 100)) - Lost;

      end; (* If *)
      if not Fall then begin
         Case Regional of
            Blight, Poor :
               case Local of
                  Blight, Poor : if not Assets.SoldGrain then
                   begin
                    if (Assets.Grain <> 0) and ((Sharecrop <> 0) or (Kind <> 0)) then
                       begin
                       LEN := StopAlert(325, nil);
                      Assets.Grain := Assets.Grain - ((REVOLTPERCLOST * Assets.Grain) div 100);

                      If Assets.Protector.Name = Conde then
                        begin
                        GIFT := PROCTREVOLTGIFT;
                        NumSpecs(GIFT, LEN, GiftStr);
```

```
                                ParamText(GiftSTR,",",");
                                LEN := StopAlert(326, nil)
                               end; (* If *)
                             end; (* If *)
                           end; (* If *)
                        end;  (* Case Local *)
                end; (* Case Regional *)
            end; (* If *)
        end; (* With *)
    end; (* NextLand *)

(*****************************************************************************************)
procedure  NextTextiles (var Assets:AssetsType; Date:DateType);

(* CALLED BY: GoToNext                                                                   *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Assets , Date                                                                *)
(* ACTION: This procedure, in the fall, calculates the return from investments in textiles the previous  *)
(* fall. The return is based on the harvest, since the better the harvest, the richer the peasantry, and the  *)
(* more cash available to spend for clothing.                                            *)

begin (* NextTextiles *)
    if Date.Fall then begin
        with Assets do begin
            case Land.Regional of
                Blight : Cash := Cash + Textiles - ((BTEXT * Textiles) div 100);
                Poor : Cash := Cash + Textiles + ((PTEXT * Textiles) div 100);
                Good : Cash := Cash + Textiles + ((GTEXT * Textiles) div 100);
                Excellent : Cash := Cash + Textiles + ((ETEXT * Textiles) div 100);
            end; (* Case *)
            Textiles := 0;
        end; (* With *)
    end; (* If *)
end; (* NextTextiles *)

(*****************************************************************************************)
procedure GetCost(var Cost:longint);

(* CALLED BY: NextRente                                                                  *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure calculates at what rate Rentes are to be sold during the next year.  It  *)
(* computes a number from one to 100 and then, based on probabilities, determines whether the King is  *)
(* selling for denier 2, denier 7, denier 11, or denier 14.                              *)

var  TEMP : integer;

begin (* GetCost *)
    TEMP := (abs(Random) div RANDDIVIDER) div 2;
    if TEMP <= DEN2PERCENT then begin
        Cost := 2;
    end else begin
```

```
        if TEMP <= DEN7PERCENT then begin
            Cost := 7;
        end else begin
            if TEMP <= DEN14PERCENT then begin
                Cost := 14;
            end else begin
                Cost := 11;
            end; (* If *)
        end; (* If *)
    end; (* If *)
end; (* GetCost *)

(*****************************************************************************************)
procedure LoseRentes(var IndivRentes:RenteHandle);

(* CALLED BY: NextRente                                                                  *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure is used in 1664, when all Rentes since 1656 are made void.  It goes through   *)
(* the list of Rentes, deleting all those bought between 1656 and 1664.                  *)

var  MARKER, DISPMARK : RenteHandle;

begin (* LoseRentes *)
    MARKER := IndivRentes;
    while MARKER^^.Next <> nil do begin
        if MARKER^^.Next^^.Year > 1656 then begin
            DISPMARK := MARKER^^.Next;
            MARKER^^.Next := MARKER^^.Next^^.Next;
            DisposHandle(Pointer(ord(DISPMARK)));
        end else begin
            MARKER := MARKER^^.Next;
        end; (* If *)
    end; (* While *)
    if IndivRentes^^.Year > 1656 then begin
        DISPMARK := IndivRentes;
        IndivRentes := IndivRentes^^.Next;
        DisposHandle(Pointer(ord(DISPMARK)));
    end; (* If *)
end; (* LoseRentes *)

(*****************************************************************************************)
function CalcPayment(Date:DateType) : integer;

(* CALLED BY: NextRente                                                                  *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Date                                                                         *)
(* ACTION: This function determines how much the King pays on his Rentes. The King either pays   *)
(* nothing (rarely), a partial payment, or full payment. In 1648 he only made a partial payment.  *)

var  TEMP : integer;
```

```
begin (* CalcPayment *)
   if Date.Year = 1648 then begin
      CalcPayment := PARTPAY;
   end else begin
      TEMP := abs(Random) div RANDDIVIDER;
      IF TEMP <= NOPAY then begin
         CalcPayment := 0;
      end else begin
         if TEMP <= SOMEPAY then begin
            CalcPayment := PARTPAY;
         end else begin
            CalcPayment := FULLPAY;
         end; (* If *)
      end; (* If *)
   end; (* If *)
end; (* CalcPayment *)

(*****************************************************************************************)
procedure NextRente(var Assets:AssetsType; Date:DateType);

(* CALLED BY: GoToNext                                                                   *)
(* CALLS TO: GetCost, LoseRentes                                                         *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure calculates how much money the player receives from Rentes for the previous  *)
(* half-year. It checks to see what the year is and sets the face value accordingly. It then determines the  *)
(* rate for Rentes for the coming year and, if it is 1664, deletes all the Rentes purchased since 1656.  *)
(* If it is Spring, the program then determines how much the King will pay on his Rentes for the  *)
(* coming year.  It finally calculates how much the player will receive from Rentes from the coming year  *)
(* and adds half this amount to the player's cash. Only half is added since this procedure is called twice  *)
(* a year.                                                                               *)

var  MARKER : RenteHandle;
   LEN : integer;

begin (* NextRente *)
   with Assets.Rente do begin
      if Date.Year < 1660 then begin
         FaceDenier := 14;
         Return := RENTE1RETURN;  {RENTE1RETURN is 71}
      end else begin
         FaceDenier := 18;
         Return := RENTE2RETURN;  {RENTE2RETURN is 55}
      end; (* If *)
      if (Date.Year = 1660) and (not Date.Fall) then begin
         LEN := StopAlert(303, nil);
      end; (* If *)
      GetCost(CostDenier);
      if (Date.Year = 1664) and (IndivRentes <> nil) and (not Date.Fall) then begin
         LoseRentes(IndivRentes);
         LEN := StopAlert(302, nil);
      end; (* If *)
      MARKER := IndivRentes;
```

```
        GotThisYear := 0;
        if not Date.Fall then begin
            Payment := CalcPayment(Date);
        end; (* If *)
        while MARKER <> nil do begin
            GotThisYear := GotThisYear + Return * Payment div 1000;
            MARKER := MARKER^^.Next;
        end; (* While *)
        Assets.Cash := Assets.Cash + (GotThisYear div 2);
    end; (* With *)
end; (* NextRente *)

{$S}

(*********************************************************************************************)
procedure SellYRente(var Assets:AssetsType; Date:DateType);

(* CALLED BY: CheckDebt, SellRente                                                           *)
(* CALLS TO: NumSpecs, SellText, GetDText, ConvertNum                                        *)
(* GLOBALS: Assets, Date                                                                     *)
(* ACTION: This procedure is used to borrow money by selling personal Rentes. The best rate the player  *)
(* can get is the going rate for the King's Rente. The player can get this rate if borrowing less than half of  *)
(* his total wealth. From that point the rate increases linearly up to the player's total wealth, at which  *)
(* point the rate is denier 2 (50%). The player cannot borrow more than total wealth. When total wealth  *)
(* is calculated, money that was gained through Rentes previously sold is not counted.       *)

var  RENTEDIALOG, SELLDIALOG : DialogPtr;
    ITEM, LEN, PERCENT : integer;
    CONVOK : boolean;
    AMT, SOLDSTR, DENVALSTR, CASHVALSTR : Str255;
    TEMP, HALF, SELLDEN : longint;

begin (* SellYRente *)
    with Assets.Rente do begin
        RENTEDIALOG := GetNewDialog(265, nil, Pointer(-1));

        repeat

            NumSpecs(CostDenier, LEN, AMT);
            NumSpecs(SoldVal,LEN, SOLDSTR);
            ParamText(AMT, SOLDSTR, '', '');
            CONVOK := true;
            SELLDEN := CostDenier;
            SellText(RENTEDIALOG, RENTEYSELLITEM, 0, 255);
            DlogManager(ITEM);

            if (ITEM = OK) then begin
                GetDText(RENTEDIALOG, RENTEYSELLITEM, AMT);
                ConvertNum(AMT, TEMP, CONVOK);

                if not CONVOK then begin
                    LEN := StopAlert(256, nil);
```

```
        end else begin

            if TEMP <> NONE then begin

                if TEMP > Assets.TotalVal - (2 * SoldVal) then begin
                    LEN := CautionAlert(266, nil);
                    CONVOK := false;
                end else begin
                    SELLDIALOG := GetNewDialog(266, nil, Pointer(-1));
                    HALF := (Assets.TotalVal - (2 * SoldVal)) div 2;
                    PERCENT := 100 div CostDenier;

                    if TEMP <= HALF then begin
                        SELLDEN := CostDenier;
                    end else begin
                        if HALF <> 0 then begin

                            SELLDEN := 10000 div ((((TEMP - HALF) * 100) div HALF) * (50 -
                                    PERCENT) + (PERCENT * 100));
                        end else begin
                            SELLDEN := 2;
                        end; (* If *)
                    end; (* If *)
                    NumSpecs(SELLDEN, LEN, AMT);
                    ParamText(AMT, '', '', '');
                    DlogManager(ITEM);
                    DisposDialog(SELLDIALOG);

                    if ITEM = Cancel then begin
                        CONVOK := false;
                    end; (* If CANCEL*)

                end; (* If Assets Allow sale*)

            end; (* If TEMP <> 0*)

        end; (* If not CONVOK*)

    end; (* If ITEM = OK*)

until CONVOK;

if (ITEM = OK) and (TEMP <> 0) then begin

    Assets.Cash := Assets.Cash + TEMP;
    SoldVal := SoldVal + TEMP;
    Owe := Owe + ((TEMP * (100 div SELLDEN)) div 100);

    NumSpecs(SELLDEN, LEN, DENVALSTR);
    NumSpecs(TEMP, LEN, CASHVALSTR);
    ParamText( DENVALSTR, CASHVALSTR, '', '');
    LEN := NoteAlert(336, nil);
```

```
        end; (* If *)

    end; (* With *)

    DisposDialog(RENTEDIALOG);
end; (* SellYRente *)

{$S Seg7}

(*****************************************************************************************)
procedure CheckDebt(var Cash, Debt:longint; var Assets:AssetsType; Date:DateType);

(* CALLED BY: CalcExpenses                                                              *)
(* CALLS TO:DisplayAssets, NumSpecs, Bankrupt, SellYRente                               *)
(* GLOBALS: Assets, Date                                                                *)
(* ACTION: This procedure checks whether or not the player has gone into debt, i.e. has negative *)
(* cash-flow. If the debt is greater than the player's total value, then Rente cannot be sold to pay off *)
(* the debt and therefore is thrown into bankruptcy. Otherwise the player is forced to sell a personal Rente *)
(* to payoff his debt.                                                                  *)

var  LEN : integer;
     STOP : boolean;
     AMT : Str255;

begin (* CheckDebt *)
    repeat
        Cash := Cash - Debt;
        DisplayAssets(Assets, Date);
        if Cash < 0 then begin
            Debt := abs(Cash);
            Cash := 0;
            if Debt > (Assets.TotalVal - (2 * Assets.Rente.SoldVal)) then begin
                Bankrupt(Assets, Date, true, true);
            end else begin
                STOP := false;
                NumSpecs(Debt, LEN, AMT);
                ParamText(AMT, '', '', '');
                LEN := StopAlert(270, nil);
                SellYRente(Assets, Date);
            end; (* If *)
        end else begin
            Debt := 0;
            STOP := true;
        end; (* If *)
    until STOP;
end; (* CheckDebt *)

(*****************************************************************************************)
procedure RaiseSalary(var OfficeList:OfficeHandle; var Salary:longint);

(* CALLED BY: NextOffice                                                                *)
```

```
(* CALLS TO: none                                                                    *)
(* GLOBALS: none                                                                     *)
(* ACTION: This procedure is called when the King decides to raise the player's salary.  *)
(* Each office's salary is raised the appropriate percentage, as is the total salary counter for all offices.  *)

var  MARKER : OfficeHandle;

begin (* RaiseSalary *)
    MARKER := OfficeList;
    while MARKER <> nil do begin
        Salary := Salary - MARKER^^.Salary;
        MARKER^^.Salary := MARKER^^.Salary + ((MARKER^^.Salary * RAISEPAY) div 100);
        Salary := Salary + MARKER^^.Salary;
        MARKER := MARKER^^.Next;
    end; (* While *)
end; (* RaiseSalary *)

(*****************************************************************************************)
procedure NextOffice(var Assets:AssetsType; Date:DateType);

(* CALLED BY: GoToNext                                                               *)
(* CALLS TO:RaiseSalary                                                              *)
(* GLOBALS: Assets, Date                                                             *)
(* ACTION: This procedure is used to advance the offices in time through to the next interval. During  *)
(* the Spring the salaries are added in. Then, if the percentages are right, the King may levy a fee  *)
(* on each office or may raise the salaries of the offices while charging a one-time fee.  *)

var  TEMP, LEN : integer;

begin (* NextOffice *)
    with Assets.Office do begin
        if not Date.Fall then begin
            Assets.Cash := Assets.Cash + Salary;
            TEMP := abs(Random) div RANDDIVIDER;
            if (TEMP <= LEVYPERCENT) and (Date.Year <> STARTYEAR) then begin
                Assets.Cash := Assets.Cash - ((TotPurchase * LEVYTAX) div 100);
                LEN := StopAlert(304, nil);
                Levied := true;
            end else begin
                if (TEMP <= RAISEPERCENT) and (Date.Year <> STARTYEAR) then begin
                    Assets.Cash := Assets.Cash - (((Salary * RAISEPAY) div 100) * RAISELEVY);
                    RaiseSalary(OfficeList, Salary);
                    Raise := true;
                    LEN := StopAlert(305, nil);
                end; (* If *)
            end; (* If *)
        end else begin
            Levied := false;
            Raise := false;
        end; (* If *)
    end; (* With *)
end; (* NextOffice *)
```

```
(*****************************************************************************)
procedure NextMarriage(var Assets:AssetsType; Date:DateType);

(* CALLED BY: SwitchGen, GoToNext                                            *)
(* CALLS TO: NumSpecs                                                        *)
(* GLOBALS: Assets, Date                                                     *)
(* ACTION: This procedure selects which brides will be available for the coming interval. If the person  *)
(* has just become eligible to marry again after having to wait because of a failed courtship, then the  *)
(* player is marked available again. It then reads in the information for the available brides and  *)
(* calculates their ages.                                                    *)

var  DUMMYMARR : BrideRec;
    CHOSENSET : set of 1..MAXBRIDES;
    FILENAME, GEN : Str255;
    NUMBER, REFNO, LEN, I, J : integer;
    TEMP, SIZE, STRSIZE, LONGSIZE, RECSIZE : longint;
    IO : OSErr;

begin (* NextMarriage *)
    with Assets.Marriage do begin
        if not Available.IsAvail then begin
            if Available.Year = Date.Year - WAITYEARS then begin
                Available.IsAvail := true;
            end; (* If *)
        end; (* If *)
        NumSpecs(Assets.Generation, LEN, GEN);
        FILENAME := Concat(MARRFILENAME, GEN);
        CHOSENSET := [];
        IO := FSOpen(FILENAME, 0, REFNO);
        SIZE := SizeOf(NUMBER);
        STRSIZE := SizeOf(Str255);
        LONGSIZE := SizeOf(longint);
        RECSIZE := SizeOf(BrideRec);
        IO := FSRead(REFNO, SIZE, @NUMBER);
        for I := 1 to NUMMARRPERYEAR do begin
            repeat
                TEMP := (abs(Random) div (32767 div NUMBER)) + 1;
            until (TEMP <= NUMBER) and (not (TEMP in CHOSENSET));
            IO := SetFPos(REFNO, 1, 0);
            IO := FSRead(REFNO, SIZE, @NUMBER);
            for J := 1 to TEMP - 1 do begin
                IO := FSRead(REFNO, RECSIZE, @ThisYear[I]);
            end; (* For *)
            CHOSENSET := CHOSENSET + [TEMP];
            IO := FSRead(REFNO, RECSIZE, @ThisYear[I]);
            if Assets.Generation = FIRSTGEN then begin
                ThisYear[I].Age := ThisYear[I].Age + (Date.Year - STARTYEAR);
            end else begin
                ThisYear[I].Age := ThisYear[I].Age + (Date.Year- DEATHYEAR);
            end; (* If *)
        end; (* For *)
```

```
        IO := FSClose(REFNO);
    end; (* With *)
end; (* NextMarriage *)

(*****************************************************************************************)
procedure NextLease(var Assets:AssetsType; Date:DateType; WatchHdl:CursHandle);

(* CALLED BY: GoToNext                                                                   *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Assets, Date, WatchHdl                                                       *)
(* ACTION: This procedure manages the annual leases. It calculates the return on a held lease,  *)
(* if it is due and one was owned, and adds this amount to the player's cash. Then, if it is the Spring,  *)
(* it determines which lease is to be made available for the coming year by going through the lease file  *)
(* and randomly picking one of the leases. It then determines what the face value is and how much the  *)
(* can get it for. It then marks this lease as not yet bought. NOTE: special year 1639, lease on fish toll.  *)

var  REFNO, IO, NUMBER, TEMP, I, RETURN : integer;
     SIZE : longint;
     LEASEDIALOG : DialogPtr;

begin (* NextLease *)
    with Assets.Land, Assets.Lease do begin
        if (not Date.Fall) and (Bought) then begin
            case Regional of
                Blight : RETURN := BRETURN;
                Poor : RETURN := PRETURN;
                Good : RETURN := GRETURN;
                Excellent : RETURN := ERETURN;
            end; (* Case *)
            GotThisYear := (FaceValue * RETURN) div 100;
            FaceValue := 0;
            Assets.Cash := Assets.Cash + GotThisYear;
        end; (* If *)
        if not Date.Fall then begin
            if Date.Year = 1639 then begin
                Title := 'Royal toll on herring and salmon in the Carenton district';
                FaceValue := 6000;
                OldOffer := Offer;
                Offer := 2499;
            end else begin
                Hlock(Pointer(WatchHdl));
                SetCursor(WatchHdl^^);
                Hunlock(Pointer(WatchHdl));
                IO := FSOpen(LEASEFILE, 0, REFNO);
                SIZE := SizeOf(NUMBER);
                IO := FSRead(REFNO, SIZE, @NUMBER);
                repeat
                    TEMP := (abs(Random) div (32767 div NUMBER)) + 1;
                until TEMP <= NUMBER;
                SIZE := SizeOf(Title);
                for I := 1 to TEMP do begin
                    IO := FSRead(REFNO, SIZE, @Title);
```

```
            end; (* For *)
            IO := FSClose(REFNO);
            SetCursor(Arrow);
           FaceValue := ord4(Random) + ord4(LEASEMIN) + ord4(32768);
            OldOffer := Offer;
            repeat
              Offer := abs(Random) div (32767 div 5);
            until Offer <= 4;
           Offer := (FaceValue * (((Offer + 3) *10)+8)) div 100;
           if (Assets.Protector.Name = Mazarin) or (Assets.Protector.Name = Fouquet) then begin
              Offer := (Offer * 3) div 4;
            end; (* If *)
         end; (* If *)
         Bought := false;
      end else begin
         GotThisYear := 0;
      end; (* If *)
   end; (* With *)
end; (* NextLease *)

(*****************************************************************************************)
procedure CalcExpenses(var Assets:AssetsType; Date:DateType);

(* CALLED BY: GoToNext                                                                   *)
(* CALLS TO: CheckDebt                                                                   *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure computes half-year expenses. During the Spring the taxes are computed. Then *)          *)
(* cost of living is computed, based on the player's prestige. The taxes and half the cost of living are *)
(* subtracted from the player's cash, along with the player's debt and half the amount owed from personal *)
(* Rentes.  Only these half-amounts are subtracted because this procedure is called twice a year. If the *)
(* player's cash is negative, then the cash is set to zero and the amount owed is assigned to the debt.  *)

begin (* CalcExpenses *)
   with Assets do begin
      Taxes := 0;
      if (not Date.Fall) and (not Assets.Noble) then begin
         Taxes := LANDTAX * (Land.Inherited + Land.Bought + Land.Seigneurie
                  + Land.Vicomte + Land.Marquisat);
      end; (* If *)
      Cash := Cash - Taxes - (CostOfLiving div 2) - (Rente.Owe div 2);
      CheckDebt(Cash, Debt, Assets, Date);
   end; (* With *)
end; (* CalcExpenses *)

(*****************************************************************************************)
procedure KillKid(var Marker:KidHandle; var Number, NumSex:longint);

(* CALLED BY: CheckSexDeath                                                              *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure is called when a kid has died. Marker^^.Next points to the deceased, and *)
(* the kid is killed by deletion of his/her KidHandle in the list of kids.               *)
```

```
var  DISPMARK : KidHandle;
     LEN : integer;

begin (* KillKid *)
   DISPMARK := Marker^^.Next;
   Marker^^.Next := Marker^^.Next^^.Next;
   DisposHandle(Pointer(DISPMARK));
   LEN := StopAlert(277, nil);
   Number := Number - 1;
   NumSex := NumSex - 1;
end; (* KillKid *)

(*****************************************************************************************)
procedure CheckSexDeath(var List:KidHandle; var TotNum, NumSex:longint; Str:Str255;
                                                          Date:DateType; Gen:longint);

(* CALLED BY: CheckDeaths                                                          *)
(* CALLS TO: KillKid                                                               *)
(* GLOBALS:  Date                                                                  *)
(* ACTION: This procedure checks to see if any of the kids pointed to by List have died.  *)
(* DEATHPERCENT of the kids die before age one, and OLDDEATHPERCENT die before age 20.  *)
(* The oldest son automatically survives.                                          *)

var  TEMP, LEN : integer;
     MARKER, DISPMARK : KidHandle;

begin (* CheckSexDeath *)
     if NumSex <> 0 then begin
          MARKER := List;
          while MARKER^^.Next <> nil do begin
            if (MARKER^^.Next^^.Birth.Year = Date.Year) or ((MARKER^^.Next^^.Birth.Year = Date.Year - 1)
                  and (MARKER^^.Next^^.Birth.Fall =  not Date.Fall) and (not Date.Fall)) then begin
               TEMP := abs(Random) div (32767 div 4);
               if TEMP = 0 then begin
                    ParamText(Str, '', '', '');
                    KillKid(MARKER, TotNum, NumSex);
               end else begin
                    MARKER := MARKER^^.Next;
               end; (* If *)
            end else begin
              if Date.Year - MARKER^^.Next^^.Birth.Year <= BEATDEATH then begin
                    TEMP := abs(Random) div (32767 div 10000);
                    if TEMP <= DEATHPERCENT then begin
                         ParamText(Str, '', '', '');
                         KillKid(MARKER, TotNum, NumSex);
                    end else begin
                         MARKER := MARKER^^.Next;
                    end; (* If *)
               end else begin
                    MARKER := MARKER^^.Next;
               end; (* If *)
```

```
            end; (* If *)
          end; (* While *)
          if (Str = 'daughters') or (Gen = SECONDGEN) then begin
            TEMP := abs(Random) div (32767 div 10000);
            if (TEMP <= OLDDEATHPERCENT) and (Date.Year - List^^.Birth.Year <= BEATDEATH) then begin
              ParamText(Str, '', '', '');
              DISPMARK := List;
              List := List^^.Next;
              DisposHandle(Pointer(DISPMARK));
              TotNum := TotNum - 1;
              NumSex := NumSex - 1;
              LEN := StopAlert(277, nil);
            end; (* If *)
          end; (* If *)
        end; (* If *)
    end; (* CheckSexDeath *)

(*****************************************************************************************)
procedure CheckDeaths(var Assets:AssetsType; Date:DateType);

(* CALLED BY: Demographics                                                              *)
(* CALLS TO: CheckSexDeath                                                              *)
(* GLOBALS: Assets, Date                                                                *)
(* ACTION: This procedure checks to see if any boys or girls have died, and, if so, kills them.  *)

begin (* CheckDeaths *)
    with Assets.Children do begin
        CheckSexDeath(Boys, Number, NumBoys, 'sons', Date, Assets.Generation);
        CheckSexDeath(Girls, Number, NumGirls, 'daughters', Date, Assets.Generation);
    end; (* With *)
end; (* CheckDeaths *)

(*****************************************************************************************)
procedure CheckWill(var Assets:AssetsType);

(* CALLED BY: AddChild, Demographics, MakeWill                                          *)
(* CALLS TO: none                                                                       *)
(* GLOBALS: Assets                                                                      *)
(* ACTION: This procedure determines whether or not the will is in accord with traditional distribution.  *)
(* The oldest son gets at least 20% plus (80 times (1/n))%, where 'n' is the number of children. The rest of  *)
(* the children must get at least 100/2n %. Nonkin, Otherkin, and charity must get at least 1%, and the  *)
(* Church must get at least 5%.                                                         *)

var  MINIMUM : WillArray;
    I : integer;

begin (* CheckWill *)
    with Assets.Will do begin
        if Assets.Children.Number <> NONE then begin
            MINIMUM[1] := 20 + (80 div Assets.Children.Number);
            MINIMUM[2] := (100 div (2 * Assets.Children.Number)) * (Assets.Children.NumBoys - 1);
            MINIMUM[3] := (100 div (2 * Assets.Children.Number)) * (Assets.Children.NumGirls);
```

```
        end else begin
            MINIMUM[1] := NONE;
            MINIMUM[2] := NONE;
            MINIMUM[3] := NONE;
        end; (* If *)
        if Assets.Children.Number = 1 then begin
            MINIMUM[1] := 90;
        end; (* If *)
        MINIMUM[4] := KINMIN;
        MINIMUM[5] := NONKINMIN;
        MINIMUM[6] := CHARMIN;
        MINIMUM[7] := CHURCHMIN;
        InAccord := true;
        for I := 1 to NUMWILLCATEGORIES do begin
            if Distribution[I] < MINIMUM[I] then begin
                InAccord := false;
            end; (* If *)
        end; (* For *)
    end; (* With *)
end; (* CheckWill *)

(*****************************************************************************************)
procedure AddChild(var TotNum, NumSex:longint; var List:KidHandle; Date:DateType; var Assets:AssetsType;
                PersMenu:MenuHandle);

(* CALLED BY: Demographics                                                              *)
(* CALLS TO: CheckWill                                                                  *)
(* GLOBALS: Assets, Date                                                                *)
(* ACTION: This procedure inserts a child into the list of children. It increments the total number of  *)
(* children and the number of children of that sex. It allocates a handle to that child and inserts that handle  *)
(* end of the list pointed to by LIST.                                                  *)

var  NEWKID, MARKER : Kidhandle;
    LEN : integer;

begin (* AddChild *)
    TotNum := TotNum + 1;
    NumSex := NumSex + 1;
   NEWKID := Pointer(Newhandle(SizeOf(KidRec)));
   NEWKID^^.Birth := Date;
   NEWKID^^.Next := nil;
    if List = nil then begin
        List := NEWKID;
   end else begin
        MARKER := List;
        while MARKER^^.Next <> nil do begin
            MARKER := MARKER^^.Next;
        end; (* While *)
       MARKER^^.Next := NEWKID;
   end; (* If *)
   if (Assets.Will.Made) and (Assets.Will.InAccord) then begin
        CheckWill(Assets);
```

```
        if not Assets.Will.InAccord then begin
            LEN := StopAlert(285, nil);
        end; (* If *)
    end; (* If *)
    if (Assets.Generation = SECONDGEN) then begin
        EnableItem(PersMenu, PLANITEM);
    end; (* If *)
end; (* AddChild *)

(*****************************************************************************)
procedure DemoGraphics(var Assets:AssetsType; Date:DateType; PersMenu:MenuHandle);

(* CALLED BY: GoToNext, DoMarriage                                           *)
(* CALLS TO: CheckDeaths, CheckWill, AddChild                                *)
(* GLOBALS: Assets, Date                                                     *)
(* ACTION: This procedure computes births and deaths. It increases the bride's age every year. If no child  *)
(* has been born, it sets the first birth for the next year. It then checks to see if any kids have         *)
(* died. If a baby is due and the mother is not too old, a baby is then born and, if the first generation,  *)
(* a new baby's birthdate is computed. In the second generation, FAMILY PLANNING is used to compute        *)
(* the birthdates.                                                           *)

var  TEMP, LEN : integer;
    NUM : longint;

begin (* DemoGraphics *)
    with Assets.Marriage do begin
        if (Married) and (not Date.Fall) then begin
            Bride.Age := Bride.Age + 1;
        end; (* If *)
    end; (* With *)
    with Assets.Children do begin
        if (NextBirth.Year = NONE) and (Assets.Generation = FIRSTGEN) then begin
            NextBirth.Year := Date.Year + 1;
            NextBirth.Fall := Date.Fall;
        end else begin
            NUM := Number;
            CheckDeaths(Assets, Date);
            if (Number < NUM) and (Assets.Will.Made) and (Assets.Will.InAccord) then begin
                CheckWill(Assets);
                if not Assets.Will.InAccord then begin
                    LEN := StopAlert(306, nil);
                end; (* If *)
            end; (* If *)
            if (NextBirth.Year = Date.Year) and (NextBirth.Fall = Date.Fall) and
                    (Assets.Marriage.Bride.Age <= TOOOLDFORKIDS) then begin
                TEMP := abs(Random) div (32767 div 2);
                if (TEMP = 0) or ((Number = 0) and (Assets.Generation =  FIRSTGEN)) then begin
                    ParamText('boy', '', '', '');
                    LEN := NoteAlert(276, nil);
                    AddChild(Number, NumBoys, Boys, Date, Assets, PersMenu);
                    if NumBoys = 1 then begin
                        DoPicture(FirstBorn);
```

```
                end; (* If *)
            end else begin
                ParamText('girl', '', '', '');
                LEN := NoteAlert(276, nil);
                AddChild(Number, NumGirls, Girls, Date, Assets, PersMenu);
            end; (* If *)
            if Assets.Generation = FIRSTGEN then begin
                TEMP := abs(Random) div (32767 div 2);
                NextBirth.Year := NextBirth.Year + TEMP + BIRTHOFFSET;
                TEMP := abs(Random) div (32767 div 2);
                case TEMP of
                    0 : NextBirth.Fall := true;
                    1 : NextBirth.Fall := false;
                end; (* Case *)
            end; (* If *)
            CheckWill(Assets);
        end; (* If *)
    end; (* If *)
  end; (* With *)
end; (* Demographics *)

{$S Seg13}

(*****************************************************************************************)
procedure PutWillItems(WillDialog:DialogPtr; ItemNums, Distribution:WillArray);

(* CALLED BY: MakeWill, Main                                                              *)
(* CALLS TO: NumSpecs                                                                     *)
(* GLOBALS: none                                                                          *)
(* ACTION: This procedure puts the current distribution of the player's will into the Dialog box.  *)

var  I, DUMMYTYPE, LEN : integer;
  ITEMHDL : Handle;
  DUMMYRECT : Rect;
  AMT : Str255;

begin (* PutWillItems *)
  for I := 1 to NUMWILLCATEGORIES do begin
    GetDItem(WillDialog, ItemNums[I], DUMMYTYPE, ITEMHDL, DUMMYRECT);
    NumSpecs(Distribution[I], LEN, AMT);
    SetIText(ITEMHDL, AMT);
  end; (* For *)
end; (* PutWillItems *)

(*****************************************************************************************)
procedure DisCancel (TheDialog:DialogPtr);

(* CALLED BY: MakeWill                                                                    *)
(* CALLS TO: none                                                                         *)
(* GLOBALS: none                                                                          *)
(* ACTION: This procedure cancels the CANCEL button in TheDialog.                         *)
```

```
var  DUMMYTYPE : integer;
    ITEMHDL : Handle;
    DUMMYRECT : Rect;

begin (* DisCancel *)
    GetDItem(TheDialog, Cancel, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    HiliteControl(Pointer(ITEMHDL), BTNINACTIVE);
end; (* DisCancel *)

(*****************************************************************************************)
procedure MakeWill(var Assets:AssetsType; InTestate:boolean);

(* CALLED BY: SwitchGen, DoCommand                                                       *)
(* CALLS TO: PutWillItems, DisCancel, SellText, ConvertNum, CheckWill                    *)
(* GLOBALS: Assets                                                                       *)
(* ACTION: This procedure is called when the player wants to make out a will. It displays the will and,  *)
(* if the player is about to die, disables the CANCEL button. It then reads in the new percentages. It   *)
(* determines whether or not the will is in accord with traditional practice and, if not, gives the player  *)
(* a chance to redo it. It then stores the new will.                                     *)

var  WILLDIALOG : DialogPtr;
    CONVOK, GOOD, OLDMADE : boolean;
    TOTAL, TEMP : longint;
    ITEM, LEN, I : integer;
    NEWAMT, STR : Str255;
    ITEMNUMS, OLDDIST : WillArray;

begin (* MakeWill *)
    ITEMNUMS[1] := OLDSONITEM;
    ITEMNUMS[2] := OTHERITEM;
    ITEMNUMS[3] := DAUGHTITEM;
    ITEMNUMS[4] := KINITEM;
    ITEMNUMS[5] := NONKINITEM;
    ITEMNUMS[6] := CHARITYITEM;
    ITEMNUMS[7] := CHURCHITEM;
    WILLDIALOG := GetNewDialog(272, nil, Pointer(-1));
    with Assets.Will do begin
        OLDDIST := Distribution;
        OLDMADE := Made;
        PutWillItems(WILLDIALOG, ITEMNUMS, Distribution);
        if InTestate then begin
            DisCancel(WILLDIALOG);
        end; (* If *)
        repeat
            CONVOK := true;
            SelIText(WILLDIALOG, ITEMNUMS[1], 0, 255);
            DlogManager(ITEM);
            if ITEM = OK then begin
                GetDText(WILLDIALOG, ITEMNUMS[1], NEWAMT);
                ConvertNum(NEWAMT, Distribution[1], CONVOK);
                TOTAL := Distribution[1];
                for I := 2 to NUMWILLCATEGORIES do begin
```

```
                GetDText(WILLDIALOG, ItemNums[I], NEWAMT);
                GOOD := true;
                ConvertNum(NEWAMT, Distribution[I], GOOD);
                if not GOOD then begin
                    CONVOK := false;
                end else begin
                    TOTAL := TOTAL + Distribution[I];
                end; (* If *)
            end; (* For *)
            if not CONVOK then begin
                LEN := StopAlert(256, nil);
            end else begin
                if TOTAL > 100 then begin
                    CONVOK := false;
                    LEN := StopAlert(281, nil);
                end else begin
                    CheckWill(Assets);
                    if not InAccord then begin
                        LEN := CautionAlert(283, nil);
                        if (LEN = Cancel) then begin
                            CONVOK := false;
                        end else begin
                            Made := true;
                        end; (* If *)
                    end else begin
                        LEN := StopAlert(319, nil);
                        Made := true;
                    end; (* If *)
                end; (* If *)
            end; (* If *)
        end else begin
            Distribution := OLDDIST;
            Made := OLDMADE;
            CheckWill(Assets);
        end; (* If *)
    until CONVOK;
    DisposDialog(WILLDIALOG);
  end; (* With *)
end; (* MakeWill *)

{$S Seg4}

(*****************************************************************************************)
procedure SwitchGen(var Assets:AssetsType; var Done:boolean; PersMenu:MenuHandle; var Date:DateType;
          Third:boolean; var Corrfile: Str255);

(* CALLED BY: EndSimulation, GoToNext, Main                                              *)
(* CALLS TO: MakeWill, NextMarriage                                                      *)
(* GLOBALS: CorrRefNum, Done, Assets, Date, Corrfile                                     *)
(* ACTION: This procedure is used to switch generations. If the player had no sons, then the *)
(* simulation ends. Otherwise, if no will has been made, it forces the player to make a will. It then *)
(* proceeds to distribute the player's assets to the  oldest son according to the latest *)
```

```
(* will. Also, all the player's personal aspects, such as marriage and kids, are reset for the son.          *)

var  LEN, I, NUM, ERR, DUMMYINT : integer;
     MARKER, DISPMARK : KidHandle;
     PERCENT, VAL, SIZE : longint;
     RMARKER : RenteHandle;
     OMARKER, ODISPMARK : OfficeHandle;
     STOP : boolean;

begin (* SwitchGen *)
     with Assets do begin
          if Children.NumBoys = NONE then begin
               LEN := StopAlert(279, nil);
               Done := true;
          end else begin
               if not Will.Made then begin
                    LEN := NoteAlert(282, nil);
                    MakeWill(Assets, true);
               end; (* If *)
               if not Third then begin
                    LEN := StopAlert(278, nil);
               (* end;   If *)
               Age := Date.Year - Children.Boys^^.Birth.Year;
               if (Age < STARTAGE) and (not Third) then begin
                    LEN := StopAlert(280, nil);
                    Date.Year := Date.Year + (STARTAGE - Age);
                    Age := STARTAGE;
               end; (* If *)
               Generation := SECONDGEN;
               Marriage.Married := false;
               Marriage.Available.IsAvail := true;
               Marriage.Failures := NONE;
               Marriage.MarrBelow := false;
               MARKER := Children.Boys;
               for I := 1 to Children.NumBoys do begin
                    DISPMARK := MARKER;
                    MARKER := MARKER^^.Next;
                    DisposHandle(Pointer(DISPMARK));
               end; (* For *)
               MARKER := Children.Girls;
               for I := 1 to Children.NumGirls do begin
                    DISPMARK := MARKER;
                    MARKER := MARKER^^.Next;
                    DisposHandle(Pointer(DISPMARK));
               end; (* For *)
               Children.Number := 0;
               Children.NumBoys := 0;
               Children.NumGirls := 0;
               Children.Boys := nil;
               Children.Girls := nil;
               Children.NextBirth.Year := 0;
               EnableItem(PersMenu, MARRITEM);
```

```
DisableItem(PersMenu, PLANITEM);
 PERCENT := Will.Distribution[1];
Will.Made := false;
for I := 1 to NUMWILLCATEGORIES do begin
    Will.Distribution[I] := NONE;
end; (* For *)
Will.WasInAccord := Will.InAccord;
Cash := Cash - Marriage.Bride.Dowry;
 if Cash < 0 then begin
    Cash := 0;
end; (* If *)
 Cash := ((Cash * PERCENT) div 100);
Land.Inherited := (((Land.Inherited + Land.Bought) * PERCENT) div 100);
 Land.Bought := NONE;
Land.Seigneurie := ((Land.Seigneurie * PERCENT) div 100);
Land.Vicomte := ((Land.Vicomte * PERCENT) div 100);
Land.Marquisat := ((Land.Marquisat * PERCENT) div 100);
 Grain := ((Grain * PERCENT) div 100);
 Rente.Owe := ((Rente.Owe * PERCENT) div 100);
 Rente.SoldVal := ((Rente.SoldVal * PERCENT) div 100);
 Rente.GotThisYear := NONE;
 RMARKER := Rente.IndivRentes;
 NUM := NONE;
 while RMARKER <> nil do begin
    NUM := NUM + 1;
   RMARKER := RMARKER^^.Next;
end; (* While *)
NUM := NUM - ((NUM * PERCENT) div 100);
for I := 1 to NUM do begin
    RMARKER := Rente.IndivRentes;
  Rente.IndivRentes := Rente.IndivRentes^^.Next;
    DisposHandle(Pointer(RMARKER));
end; (* For *)
VAL := ((Office.TotPurchase * PERCENT) div 100);
Office.TotPurchase := NONE;
Office.Number := NONE;
Office.Salary := NONE;
OMARKER := Office.OfficeList;
 STOP := false;
 if OMARKER <> nil then begin
    repeat
       OMARKER^^.Inherited := true;
       Office.TotPurchase := Office.TotPurchase + OMARKER^^.AmtPaid;
        Office.Number := Office.Number + 1;
       Office.Salary := Office.Salary + OMARKER^^.Salary;
        if Office.TotPurchse < VAL then begin
           OMARKER := OMARKER^^.Next;
        end else begin
            STOP := true;
        end; (* If *)
    until (OMARKER = nil) or (STOP);
    if OMARKER <> nil then begin
```

```
                while OMARKER^^.Next <> nil do begin
                    ODISPMARK := OMARKER^^.Next;
                   OMARKER^^.Next := OMARKER^^.Next^^.Next;
                    DisposHandle(Pointer(ODISPMARK));
                end; (* While *)
            end; (* If *)
        end; (* If *)
        ERR := FSClose (CorrRefNum);
        Corrfile := 'SecondMail.1.dat';
        ERR := FSOpen(CORRFILE, 0, CorrRefNum);
        SIZE := SizeOf(integer);
        ERR := FSRead(CorrRefNum, SIZE, @DUMMYINT);
        Assets.Mail.Year := NONE;
        SIZE := SizeOf(MailRec);
        with Assets do begin
          while (Mail.Year < Date.Year) or ((Mail.Year = Date.Year) and (Mail.Fall = Date.Fall)) or
              ((Date.Fall) and (Mail.Year = Date.Year)) do begin
              ERR := FSRead(CorrRefNum, SIZE, @Mail);
          end; (* While *)
        end; (* With *)
        TooAmbitious := NONE;
        PresFallen := false;
        Lease.NumBought := NONE;
        Marriage.Failures := NONE;
        Protector.YearFail := NONE;
        Protector.NumFailures := NONE;
        Protector.ThisProctFail := NONE;
        NextMarriage(Assets, Date);
        end;  (* If *)
    end; (* If *)
  end; (* With *)
end; (* SwitchGen *)

(*****************************************************************************************)
procedure DispLetter (Letter:Str255; CorrWindow:WindowPtr);

(* CALLED BY: NextCorr                                                                   *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure displays a window for correspondence.                          *)

var  ASTRING : StrArray;
    TEXTRECT : Rect;
    ITEM, I : integer;
    CONTDLOG : DialogPtr;

begin (* DispLetter *)
    SetPort(CorrWindow);
    PLSetWrPort(CorrWindow);
    SetRect(TEXTRECT, 10, 15, 370, 200);
    for I := 1 to Length(Letter) do begin
        ASTRING[I] := Letter[I];
```

```
    end; (* For *)
    TextBox(@ASTRING, Length(Letter), TEXTRECT, TEJustLeft);
{   STRHDL := NewString(Letter);
    TextBox(Pointer(STRHDL^), LEN, TEXTRECT, TEJustLeft);
    DisposHandle(Pointer(STRHDL));}
    MoveTo(0, 200);
    writeln('_____');
    writeln;
    writeln('This is the only time you will see this letter.');
    writeln('Are you ready to go on?');
    CONTDLOG := GetNewDialog(256, nil, Pointer(-1));
    DlogManager(ITEM);
    DisposDialog(CONTDLOG);
end; (* DispLetter *)

(*****************************************************************************************)
procedure NextCorr(var Assets:AssetsType; Date:DateType; CorrRefNum:integer);

(* CALLED BY: GoToNext                                                                   *)
(* CALLS TO: DisplayAssets, DispLetter                                                   *)
(* GLOBALS: CorrRefNum, Assets, Date, Corrfile                                           *)
(* ACTION: This procedure reads the correspondence for a given date and protector. Generic denotes mail  *)
(* for all. Rente value of 2 provides mail for followers of M. Colbert and the Duke of Burgundy.         *)

var  CORRWINDOW : WindowPtr;
    TITLE : Str255;
    WREC : WindowRecord;
    ERR, I, DUMMYINT : integer;
    SIZE : longint;
    ACTIVE : boolean;

begin (* NextCorr *)
    SIZE := SizeOf(MailRec);
    ACTIVE := false;

    with Assets, Assets.Mail do begin
        ERR := NONE;

        If Date.Year = 1656 then begin
        ERR := FSClose (CorrRefNum);
            Corrfile := 'FirstMail.2.dat';
            ERR := FSOpen(CORRFILE, 0, CorrRefNum);
            SIZE := SizeOf(integer);
            ERR := FSRead(CorrRefNum, SIZE, @DUMMYINT);
            Assets.Mail.Year := NONE;
            SIZE := SizeOf(MailRec);
             with Assets do begin
               while (Mail.Year < Date.Year) or ((Mail.Year = Date.Year) and (Mail.Fall = Date.Fall)) or
                    ((Date.Fall) and (Mail.Year = Date.Year)) do begin
                    ERR := FSRead(CorrRefNum, SIZE, @Mail);
                end; (* While *)
             end; (* With *)
```

```
    end;  (* If *)

    If Date.Year = 1695 then begin
     ERR := FSClose (CorrRefNum);
        Corrfile := 'SecondMail.2.dat';
         ERR := FSOpen(CORRFILE, 0, CorrRefNum);
        SIZE := SizeOf(integer);
        ERR := FSRead(CorrRefNum, SIZE, @DUMMYINT);
        Assets.Mail.Year := NONE;
        SIZE := SizeOf(MailRec);
         with Assets do begin
           while (Mail.Year < Date.Year) or ((Mail.Year = Date.Year) and (Mail.Fall = Date.Fall)) or
                ((Date.Fall) and (Mail.Year = Date.Year)) do begin
                ERR := FSRead(CorrRefNum, SIZE, @Mail);
            end; (* While *)
        end; (* With *)
    end;  (* If *)

    while (Date.Year = Year) and (Date.Fall = Fall) and (Contact <> Generic) and
        (Contact <> Protector.Name) and (ERR = NONE) do begin
            ERR := FSRead(CorrRefNum, SIZE, @Mail);
    end; (* While Mail Record not Current *)

    if ((Date.Year = Year) and (Date.Fall = Fall) and ((Contact = Generic) or (Contact = Protector.Name)))
        or ((Rente.CostDenier = 2) and ((Protector.Name = Colbert)
                or (Protector.Name = DukeOfBurgundy))) then begin
        CORRWINDOW := GetNewWindow(257, @WREC, Pointer(-1));
        TITLE := 'Correspondence';
        SetWTitle(CORRWINDOW, TITLE);
        ShowWindow(CORRWINDOW);
         ACTIVE := true;
        DisplayAssets(Assets, Date);
    end; (* If *)

    while (Date.Year = Year) and (Date.Fall = Fall) and (ERR = NONE) do begin
        if (Contact = Generic) or (Contact = Protector.Name) then begin
             Assets.Cash := Assets.Cash + Cash;
             if Cash <> NONE then begin
                 DisplayAssets(Assets, Date);
             end; (* If *)
             DispLetter(Content, CORRWINDOW);
        end; (* If *)
        ERR := FSRead(CorrRefNum, SIZE, @Mail);
    end; (* While *)

    if (Rente.CostDenier = 2)
                and ((Protector.Name = Colbert) or (Protector.Name = DukeOfBurgundy)) then begin
        TITLE := 'Your sources inform you that the King is selling Rentes at a discounted rate.';
        DispLetter(TITLE, CORRWINDOW);
    end; (* If *)
end; (* With *)
if ACTIVE then begin
```

```
        CloseWindow(CORRWINDOW);
    end; (* If *)
end; (* NextCorr *)

(*****************************************************************************************)
procedure NextProct (var Assets:AssetsType; Date:DateType);

(* CALLED BY: GoToNext                                                                   *)
(* CALLS TO: Bankrupt                                                                    *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure lists the dates of death or disfavor for each protector; it bankrupts followers of  *)
(* Fouquet and Particelli in their last year.                                            *)

var  LEN : integer;

begin (* NextProct *)
    with Date, Assets.Protector do begin
        Assets.ChoseProct := false;
       if  ((Year = 1640) and (Name = Cornuel)) or
           ((Year = 1648) and (Name = Particelli)) or
           ((Year = 1661) and (Name = Mazarin)) or
           ((Year = 1661) and (Name = Fouquet) and (Fall)) or
           ((Year = 1684) and (Name = Colbert)) or
           ((Year = 1686) and (Name = Conde)) or
           ((Year = 1711) and (Name = GrandDauphin)) or
           ((Year = 1712) and (Name = DukeOfBurgundy)) then begin

               if Name = Particelli then begin                {Bankruptcy routines}
                    LEN := StopAlert(315, nil);
                   Bankrupt(Assets, Date, false, true);
               end; (* If *)

               if Name = Fouquet then begin
                    LEN := StopAlert(313, nil);
                   Bankrupt(Assets, Date, false, true);
               end; (* If *)

              if (Name = Fouquet) or (Name = Particelli) or (Name = Cornuel) or
               (Name = Mazarin) or (Name = Maintenon) or (Name = GrandDauphin)
                                  or (Name = DukeofBurgundy) then begin
               LEN := StopAlert(314, nil);          {Assigns default of NoProtector}
               Name := NoProtector;
               ThisProctFail := NONE;
               end; (* If *)

               if Name = Conde then begin           {Assigns new protector for special coteries}
                   Name := GrandDauphin;
                    ThisProctFail := None;
                    Assets.ChoseProct := true;
                    Paramtext('Grand Dauphin',",",");
                    LEN := StopAlert(338, nil);
               end;  (* If *)
```

```
            if Name = Colbert then begin
               Name := DukeOfBurgundy;
                ThisProctFail := None;
                Assets.ChoseProct := true;
                Paramtext('Duke of Burgundy', ",",");
                LEN := StopAlert(338, nil);
            end;  (* If *)

        end; (* If *)
    end; (* With *)
end; (* NextProct *)


(*****************************************************************************************)
procedure EndSimulation(var Assets:AssetsType; var Done:Boolean; PersMenu:MenuHandle; var Date:DateType);

(* CALLED BY: GoToNext                                                                    *)
(* CALLS TO: SwitchGen, NumSpecs, DisplayAssets                                           *)
(* GLOBALS: Done, Assets, Date, Corrfile                                                  *)
(* ACTION: This procedure is called in 1715 when the simulation is about to end. It moves the player  *)
(* into the third generation and recounts the player's achievements in the simulation.    *)

var  LEN, count : integer;
     PRESSTR : Str255;

begin (* EndSimulation *)

    SwitchGen(Assets, Done, PersMenu, Date, true, Corrfile);

    DisplayAssets(Assets, Date);

    if not Done then begin
        NumSpecs(Assets.Prestige, LEN, PRESSTR);
        ParamText(PRESSTR, ", ", ");
        LEN := Alert(299, nil);
        Done := true;
        Assets.Won := true;
    end; (* If *)

end; (* EndSimulation *)

{$S Seg7}

(*****************************************************************************************)
procedure GoToNext(var Assets:AssetsType; var Date:DateType; var Icons:IconType;
            FinMenu, PersMenu:MenuHandle; var Done:boolean; WatchHdl:CursHandle; CorrRefNum:integer);

(* CALLED BY: DoCommand, Main                                                             *)
(* CALLS TO: CalcHarvest, NextLand, NextTextiles, NextRente, NextLease, NextOffice, NextMarriage,  *)
Demographics, En_Disable, NextCorr, NextProct, CalcExpenses, SwitchGen, EndSimulation     *)
(* GLOBALS: Icons, CorrRefNum, Done, Assets, Date, WatchHdl, Corrfile                     *)
```

```
(* ACTION: This procedure is called to advance the player to the next half-year period. It changes the     *)
(* season. If it is Fall, the local and regional harvests are computed. If it is Spring, the year is advanced.  *)
(* Then the statistics for all the various financial matters are updated and the appropriate menu items      *)
(* are enabled and disabled.                                                                                 *)

var  LEN : integer;
     PRESSTR : Str255;

begin (* GoToNext *)

    Date.Fall := not Date.Fall;
    if Date.Fall then begin
        CalcHarvest(Assets.Land.Local);
        CalcHarvest(Assets.Land.Regional);
    end else begin
        Date.Year := Date.Year + 1;
        Assets.Age := Assets.Age + 1;
    end; (* If *)
    if (Date.Year = 1641) and (Assets.Lease.Hanged) then begin
        Done := true;
        LEN := Alert(269, nil);
    end else begin
        NextLand(Assets, Date);
        NextTextiles(Assets, Date);
        NextRente(Assets, Date);
        NextLease(Assets, Date, WatchHdl);
        NextOffice(Assets, Date);
        if (not Assets.Marriage.Married) then begin
            Hlock(Pointer(WatchHdl));
            SetCursor(WatchHdl^^);
            Hunlock(Pointer(WatchHdl));
            NextMarriage(Assets, Date);
            SetCursor(Arrow);
        end else begin
            Demographics(Assets, Date, PersMenu);
        end; (* If *)
        if Icons.IconWasSelected then begin
            HiliteMenu(0);
            En_Disable(Date, FinMenu, Icons, Assets);
        end; (* If *)
        if (Date.Year = DEATHYEAR) and (not Date.Fall) then begin
            SwitchGen(Assets, Done, PersMenu, Date, false, Corrfile);
        end; (* If *)
        NextCorr(Assets, Date, CorrRefNum);
        NextProct(Assets, Date);
        CalcExpenses(Assets, Date);
        if (Date.Year = ENDYEAR) and (Date.Fall) then begin
            EndSimulation(Assets, Done, PersMenu, Date);
        end; (* If *)
    end; (* If *)
end; (* GoToNext *)
```

```
{$S Seg16}

(*****************************************************************************************)
procedure DispWealth (var Assets:AssetsType; InfoWindow:WindowPtr; Date:DateType);

(* CALLED BY: DisplayStatus, Main                                                        *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure displays a summary of the player's financial position. It states the player's  *)
(* name and  various financial attributes such as cash, grain, land value, salaries, cost of living, etc.  *)

var  TITLE : Str255;
    RENTEVAL : longint;

begin (* DispWealth *)
   case Assets.Generation of
      FIRSTGEN : TITLE := 'Denis Marin';
      SECONDGEN : TITLE := 'Jean-Francois Marin';
   end; (* Case *)
   TITLE := Concat('Financial Statement for ',TITLE);
   SetWTitle(InfoWindow, TITLE);
   ShowWindow(InfoWindow);
   with Assets do begin
      MoveTo(0,15);
      if (Cash > 0) or (Debt = 0) then begin
         writeln(' Cash:  £',Cash:0,'.');
      end else begin
         writeln(' Debt:  £',Debt:0,'.');
      end; (* If *)
      writeln('Land:  £',Land.Value:0,'.');
      writeln('Grain:  ',Grain:0,' quintels at £',Land.Price:0,' per quintel.');
      writeln('Purchase value of offices:  £',Office.TotPurchase:0,'.');
      writeln('Invested in textiles:  £',Textiles:0,'.');
      writeln('Total value of personal Rentes sold:    £',Rente.SoldVal:0,'.');
      RENTEVAL := CalcRenteVal(Rente);
      writeln('Total value of Rentes purchased from the King:  £', RENTEVAL:0,'.');
      writeln;
      writeln('Annual salaries from offices:  £',Office.Salary:0,'.');
      if not Date.Fall then begin
         writeln('Income from Leases:  £',Lease.GotThisYear:0,'.');
      end; (* If *)
      writeln('Income from Rentes:  £',Rente.GotThisYear div 2:0,'.');
      writeln;
      if (not Date.Fall) and (not Noble) then begin
         writeln('Paid this year in taxes:  £',Taxes:0,'.');
      end; (* If *)
      writeln('Annual Cost of living:  £',CostOfLiving:0,'.');
      writeln('Annual payment on personal Rentes:  £',Rente.Owe:0,'.');
      if Date.Fall then begin
         writeln;
      end; (* If *)
      writeln('At today"s prices, these assets give you a total value');
```

```
        write('of £',Assets.TotalVal:0,'.');
    end; (* With *)
end; (* DispWealth *)

(*****************************************************************************************)
procedure DispSex(Str:Str255; Number:longint; List:KidHandle; Date:DateType);

(* CALLED BY: DisplayPersonal                                                            *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Date                                                                         *)
(* ACTION: This procedure displays the number of children of a sex and their ages.       *)

var  I : integer;

begin (* DispSex *)
    if Number = 1 then begin
        if Date.Year <> List^^.Birth.Year then begin
            writeln('You have one ',Date.Year - List^^.Birth.Year:0,' year old ',Str,'.');
        end else begin
            writeln('You have one new-born ',Str,'.');
        end; (* If *)
    end else begin
        write('You have ',Number:0,' ',Str,'s, aged ');
        for I := 1 to Number do begin
            if I <> Number then begin
              write(Date.Year - List^^.Birth.Year:0);
               if Number <> 2 then begin
                    write(', ');
               end else begin
                    write(' ');
               end; (* If *)
            end else begin
              writeln('and ',Date.Year - List^^.Birth.Year:0,'.');
            end; (* If *)
            List := List^^.Next;
        end; (* For *)
    end; (* If *)
end; (* DispSex *)

(*****************************************************************************************)
procedure DispPersonal(var Assets:AssetsType; InfoWindow:WindowPtr; Date:DateType);

(* CALLED BY: DispStatus                                                                 *)
(* CALLS TO: DispSex                                                                     *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure summarizes the player's personal status. It lists the wife's name, her age, and  *)
(* her father's name. It tells how many children he has and their ages. It then displays the               *)
(* distribution of the player's will and whether or not it is in accord with tradition.   It tells         *)
(* the player status on nobility, and status on  the current protector.                   *)

var  TITLE : Str255;
```

```
begin (* DispPersonal *)
   case Assets.Generation of
      FIRSTGEN : TITLE := 'Denis Marin';
      SECONDGEN : TITLE := 'Jean-Francois Marin';
   end; (* Case *)
   TITLE := Concat('Personal information about ',TITLE);
   SetWTitle(InfoWindow, TITLE);
   ShowWindow(InfoWindow);
   MoveTo(0, 15);
   with Assets.Marriage do begin
      if Married then begin
         writeln(' You are married to ', Bride.Name, '.');
         writeln('She is ', Bride.Age:0, ' years old and had a dowry worth £',Bride.Dowry:0,'.');
         writeln('Her father is ',Bride.Father,'.');
         writeln;
      end else begin
         writeln(' You are not married.');
      end; (* If *)
   end; (* With *)
   with Assets.Children do begin
      if NumBoys > NONE then begin
         DispSex('son', NumBoys, Boys, Date);
      end; (* If *)
      if NumGirls > NONE then begin
         DispSex('daughter', NumGirls, Girls, Date);
      end; (* If *)
   end; (* With *)
   with Assets.Will do begin
      if Made then begin
         writeln;
         writeln('You have distributed your estate as follows:');
         writeln;
         writeln('Oldest son:  ',Distribution[1]:0,'%.');
         writeln('Other sons:  ',Distribution[2]:0,'%.          Non-kin:  ',Distribution[5]:0,'%.',
                 '          Daughters:  ',Distribution[3]:0,'%.');
         writeln('Charity:  ',Distribution[6]:0,'%.          Other kin:  ',Distribution[4]:0,'%.',
                 '          The Church:  ',Distribution[7]:0,'%.');
         writeln;
         write('This distribution is ');
         if not InAccord then begin
            write('not ');
         end; (* If *)
         writeln('in accord with regular practice.');
      end else begin
         writeln;
         writeln('You have not yet made up a will.');
      end; (* If *)
   end; (* With *)
   writeln;
   if Assets.Noble then begin
      write('You have achieved nobility');
   end else begin
```

```
var  VAL : Str255;
     AMOUNT : longint;

begin (* DispLand *)
  SetWTitle(InfoWindow, 'Land Status');
  ShowWindow(InfoWindow);
  with Assets.Land do begin
      AMOUNT := Bought + Inherited + Seigneurie + Vicomte + Marquisat;
      MoveTo(0,15);
      HarvValue(Local, VAL);
      writeln(' This fall the Local harvest was ',VAL,', and the Regional');
      HarvValue(Regional, VAL);
      writeln('harvest was ',VAL,'.');

      writeln('Grain is going for £',Price:0,' per quintel, and landowners receive:');
      writeln('    Renting in kind:     ',KINDRENT:0,' quintels per hectare, worth £', PRICE * KINDRENT:0,'.');
      writeln('    Renting for cash:  £',RENTVALUE:0,' per hectare.');
      writeln('    ShareCropping:  ',Yield:0,' quintels per hectare, worth £',  (PRICE * Yield):0, '.');

      writeln('You are now managing your ',AMOUNT:0,' hectares as follows:');
      writeln('    Rented in kind: ',Kind:0,'%.  Yield ', KINDRENT * ((Kind * AMOUNT) div 100):0,' quintels, worth £',
                PRICE * KINDRENT *  ((Kind * AMOUNT) div 100):0,'.');
      writeln('    Rented for cash: ',Rent:0,'%.  Income this year £', RENTVALUE * ((Rent * AMOUNT) div 100):0,'.');
      writeln('    Sharecropping: ',ShareCrop:0,'%.      Yield ',
                (Yield * ((Sharecrop * AMOUNT) div 100)):0,' quintels, worth £',
                  ((PRICE * Yield)) *  ((Sharecrop * AMOUNT) div 100):0,'.');

      if Assets.Grain > 400 then
        write('Damp storage ruined ', Lost:0,' quintels of grain, ')

        else if Assets.Grain > 200 then
              write('Fungus and rot ruined ', Lost:0,' quintels of grain, ')

        else write('Rats destroyed ',Lost:0,' quintels of grain, ');
      write('worth £', Lost * PRICE:0,'.');
      writeln;
      writeln('Your ',AMOUNT:0,' hectares are distributed as follows:');
      writeln('    Miscellaneous:  ',Bought + Inherited:0,' hectares.');
      writeln('    Seigneuries:  ',Seigneurie:0,' hectares.');
      writeln('    Vicomté:       ',Vicomte:0,' hectares.');
      write('    Marquisat:       ',Marquisat:0,' hectares.');
  end; (* With *)
end; (* DispLand *)

(*****************************************************************************************)
procedure DispText(var Assets:AssetsType; InfoWindow:WindowPtr);

(* CALLED BY: DisplayStatus                                                              *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure tells the player about the return on money invested in textiles. The return  *)
```

```
(* is based on the regional harvest, with better harvests yielding better returns.                *)

var  RETURN : integer;

begin (* DispText *)
   SetWTitle(InfoWindow, 'Textile Status');
   ShowWindow(InfoWindow);
   MoveTo(0,60);
   if Date.Fall then begin
      case Assets.Land.Regional of
         Blight : RETURN := BTEXT;
         Poor : RETURN := PTEXT;
         Good : RETURN := GTEXT;
         Excellent : RETURN := ETEXT;
      end; (* Case *)
      writeln('     You have £',Assets.Textiles:0,' invested in textiles');
         writeln('    for the coming year.');
         writeln;
      writeln('    Due to peasant demand, investors received their');
       write('    initial investment ');
      if Assets.Land.Regional = Blight then begin
         write('less ');
      end else begin
         write('plus ');
      end; (* If *)
      writeln(RETURN:0, '%');
      writeln;
      writeln('   {To change the amount you have invested in textiles, ');
      writeln('    choose BUY on the Investments menu and enter ');
      writeln('    a new sum.  The new amount will be invested,');
      writeln('    the old amount discarded.}')
   end else begin
      if Assets.Textiles <> NONE then begin
         writeln('    You have £',Assets.Textiles:0,' invested in textiles');
         writeln('    for this year.');
      end else begin
         writeln('   You did not invest in textiles during the Fall.');
      end; (* If *)
   end; (* If *)
end; (* DispText *)

(*****************************************************************************************)
procedure DispLease(var Assets:AssetsType; InfoWIndow:WindowPtr; Date:DateType);

(* CALLED BY: DisplayStatus                                                                *)
(* CALLS TO: none                                                                          *)
(* GLOBALS: Assets, Date                                                                   *)
(* ACTION: This procedure displays the status of the leases. It tells the player what lease is owned  *)
(* and how much was returned from an investment in the previous year's lease, if purchased.  *)

begin (* DispLease *)
   SetWTitle(InfoWindow, 'Lease Status');
```

```
    ShowWindow(InfoWindow);
    with Assets.Lease do begin
        MoveTo(0, 30);
        if not Bought then begin
            writeln(' You don''t own any leases.');
        end else begin
            write(' You own a one-year lease to collect the');
            if not Date.Fall then begin
              if Date.Year = 1639 then begin
               write (' new');
               end else begin
               end;  (* If *)
            end;  (* If *);
            writeln;
            writeln(Title,'.');
        end; (* If *)
        if (not Date.Fall) and (GotThisYear <> 0) then begin
            writeln;
            writeln('This past year you collected ',GotThisYear:0,' livres on a lease');
            writeln('which you bought for ',OldOffer:0,' livres.');
        end; (* If *)
    end; (* With *)
end; (* DispLease *)

(*****************************************************************************************)
procedure DispOffice (var Assets:AssetsType; InfoWindow:WindowPtr);

(* CALLED BY: DisplayStatus                                                              *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure displays the status of the offices. From 1642 to 1652 the market was glutted,  *)
(* so offices were discounted. Also, the King at various times charges a fee on the offices, or elects   *)
(* to raise the salaries while charging a fee at the same time. The procedure then tells how many offices *)
(* the player owns and the player's total salary.                                        *)

begin (* DispOffice *)
    SetWTitle(InfoWindow, 'Office Status');
    ShowWindow(InfoWindow);
    with Assets.Office do begin
        MoveTo(0,30);
        if (Date.Year >= 1642) and (Date.Year <= 1652) then begin
            writeln(' The market for offices has been glutted!!');
            writeln('As a result, ALL offices are worth 20% LESS than their');
            writeln('official value!!!!!');
            writeln;
        end; (* If *)
        if Levied then begin
            writeln(' The King has just realized that you purchased your offices');
            writeln('for LESS than their TRUE value!!!');
            writeln('To correct this oversight, you must pay 20% of the purchase');
            writeln('prices of your offices to the crown.');
            writeln;
```

```
        end; (* If *)
        if Raise then begin
            writeln(' The King has graciously raised all office salaries by ',RAISEPAY:0,'%.');
            writeln('In compensation for the raise, however, he has levied a');
            writeln('one-time payment of ',RAISELEVY:0,' times the amount of the raise.');
            writeln;
        end; (* If *)
        if Number = 0 then begin
            writeln(' You don"t own any offices.');
        end else begin
            if Number = 1 then begin
                writeln(' You own one office and are paid ',Salary:0,' livres for');
                writeln('it each Spring.');
            end else begin
                writeln(' You own ',Number:0,' offices and are paid ', Salary:0,' livres');
                writeln('them each Spring.');
            end; (* If *)
        end; (* If *)
        writeln;
        writeln;
        writeln('{To see which offices you own, select SELL from the');
        writeln(' INVESTMENTS menu.  This will give you a listing of');
        writeln(' your offices.  If you don"t wish to sell any');
        writeln(' offices, just click on the CANCEL button when you');
        writeln(' are done reviewing the list.}');
    end; (* With *)
end; (* DispOffice *)

(*****************************************************************************************)
procedure DispRente (var Assets:AssetsType; InfoWindow:WindowPtr; Date:DateType);

(* CALLED BY: DisplayStatus                                                              *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure tells the player about both the King's Rentes that are owned and the personal  *)
(* Rentes were sold to make money. In 1660 the King devalued all Rentes to denier 18 and in 1664         *)
(* made all Rentes purchased sonce 1656 void. The procedure then tells the player how many of the        *)
(* four quarterly payments the King actually made. It then tells the player how much was made in Rentes   *)
(* in the past interval andhow much is owed on his personal Rentes.                      *)


begin (* DispRente *)
    SetWTitle(InfoWIndow, 'Rente Status');
    SHowWIndow(InfoWindow);
    with Assets.Rente do begin
        MoveTo(0,30);
        if (Date.Year = 1660) and (not Date.Fall) then begin
            writeln('   All Rentes were reduced from denier 14 to denier 18 by');
            writeln('  order of the King!!!');
            writeln;
        end; (* If *)
        if (Date.Year = 1664) and (not Date.Fall) then begin
```

```
            writeln('  The King has decreed that all Rentes purchased since');
            writeln('  1656 are VOID!!!');
            writeln;
        end; (* If *)
        if Payment = 0 then begin
            writeln('  Our King, in his infinite wisdom, has chosen to');
            writeln('  withhold payments on Rentes this year.');
        end else begin
            if Payment = PARTPAY then begin
                writeln('  The King has paid you for your Rentes generously!');
                writeln('  (Although only for 2 1/2 quarters!)');
            end else begin
                writeln('  Our Gracious King has generously given payment on');
                writeln('  Rentes for ALL four quarters of the year!!');
            end; (* If *)
        end; (* If *)
        writeln;
        write('  The going price for Rentes is Denier ', CostDenier:0,'.');
        writeln;
        write('  You thus made ', GotThisYear div 2:0,' livres on Rentes since last ');
        if Date.Fall then begin
            writeln('Spring.');
        end else begin
            writeln('Fall.');
        end; (* If *)
        if Owe <> 0 then begin
            writeln;
            writeln('  You also have to pay ',Owe:0,' livres on your own Rentes');
            writeln('  every year.');
        end; (* If *)
    end; (* With *)
end; (* DispRente *)

(*****************************************************************************************)
procedure ContButton;

(* CALLED BY: DisplayStatus                                                              *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: myEvent                                                                      *)
(* ACTION: This procedure simply pauses until the button is pressed.  Same as procedure DebugDelay.    *)

var  TEMP : boolean;
    ANEVENT : EventRecord;

begin (* ContButton *)
    repeat
        SystemTask;
        TEMP := GetNextEvent(everyEvent, MYEVENT);
    until (Button);
end; (* ContButton *)

{$S}
```

```
(*****************************************************************************)
procedure DisplayStatus(var Assets:AssetsType; Choice:integer; Date:DateType);

(* CALLED BY: DoCommand                                                      *)
(* CALLS TO: DispLand, DispRente, DispOffice, DispLease, DispText, DispWealth, DispPersonal,  *)
(* ContButton                                                                *)
(* GLOBALS: Assets, Date                                                     *)
(* ACTION: This procedure is used to display the status of either an investment selection or one of the  *)
(* summary options. CHOICE determines what status is to be displayed, and the procedure calls the  *)
(*  appropriate procedure to display the desired status.                     *)

var INFOWINDOW : WindowPtr;
   WREC : WindowRecord;

begin (* DisplayStatus *)
   INFOWINDOW := GetNewWindow(257, @WREC, Pointer(-1));
   SetPort(INFOWINDOW);
   PLSetWrPort (INFOWINDOW);
   case Choice of
      LANDICON : DispLand(Assets, InfoWindow, Date);
      RENTEICON : DispRente(Assets, InfoWindow, Date);
      OFFICEICON : DispOffice(Assets, InfoWindow);
      LEASEICON : DispLease(Assets, InfoWindow, Date);
      TEXTILEICON : DispText(Assets, InfoWindow);
      WEALTHDISP : DispWealth(Assets, InfoWindow, Date);
      PERSDISP : DispPersonal(Assets, InfoWindow, Date);
   end; (* Case *)
    ContButton;
   CloseWindow (INFOWINDOW);
end; (* DisplayStatus *)

{$S Seg8}

(*****************************************************************************)
procedure DispOldManageValues (ManDialog:DialogPtr; Land:LandType);

(* CALLED BY: ManageMLand                                                    *)
(* CALLS TO: NumSpecs                                                        *)
(* GLOBALS: none                                                             *)
(* ACTION: This procedure displays the amount of land managed by Renting in Kind, Renting for Cash,  *)
(* and Sharecropping.                                                        *)

var ITEMHDL : Handle;
    DUMMYRECT : Rect;
    STR : Str255;
   LEN, DUMMYTYPE : integer;

begin (* DispOldManageValues *)
   with Land do begin
      NumSpecs(ShareCrop, LEN, STR);
      GetDItem(MANDIALOG, SHAREITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
```

```
        SetIText(ITEMHDL, STR);
        NumSpecs(Kind, LEN, STR);
        GetDItem(MANDIALOG, KINDITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        SetIText(ITEMHDL, STR);
        NumSpecs(Rent, LEN, STR);
        GetDItem(MANDIALOG, CASHITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        SetIText(ITEMHDL, STR);
    end; (* With *)
end; (* DispOldManageValues *)

(*****************************************************************************)
procedure ProcessManage(var Land:LandType; var ManageOK : boolean; MANDIALOG : DialogPtr);

(* CALLED BY: ManageMLand                                                    *)
(* CALLS TO: ConvertNum                                                      *)
(* GLOBALS: none                                                             *)
(* ACTION: This procedure reads the amounts from the manage land dialog and stores them, after *)
(* checking for errors.                                                      *)

var  STR : Str255;
    LEN : integer;
    CONVOK : boolean;

begin (* ProcessManage *)
    with Land do begin
        CONVOK := true;
        GetDText(MANDIALOG, SHAREITEM, STR);
        ConvertNum(STR, ShareCrop, CONVOK);
        GetDText(MANDIALOG, KINDITEM, STR);
        ConvertNum(STR, Kind, CONVOK);
        GetDText(MANDIALOG, CASHITEM, STR);
        ConvertNum(STR, Rent, CONVOK);
        if not CONVOK then begin
            MANAGEOK := false;
            LEN := StopAlert(256, nil);
        end else begin
            if ShareCrop + Kind + Rent > 100 then begin
                MANAGEOK := false;
                LEN := StopAlert(257, nil);
            end; (* If *)
        end; (* If *)
    end; (* With *)
end; (* ProcessManage *)

(*****************************************************************************)
procedure ManageMLand (var Land:LandType);

(* CALLED BY: ManageLand                                                     *)
(* CALLS TO: SelIText, DispOldManageValues, ProcessManage                    *)
(* GLOBALS: none                                                             *)
(* ACTION: This procedure displays a dialog window for managing miscellaneous land. *)
```

```
var  MANDIALOG : DialogPtr;
    LEN, ITEM, OLDSHARE, OLDKIND, OLDRENT : integer;
     STR : Str255;
    MANAGEOK : boolean;

begin (* ManageMLand *)
    MANDIALOG := GetNewDialog(257, nil, Pointer(-1));
    with Land do begin
        OLDSHARE := ShareCrop;
        OLDRENT := Rent;
        OLDKIND := Kind;
        repeat
            MANAGEOK := true;
           DispOldManageValues(MANDIALOG, Land);
            SelIText(MANDIALOG, KINDITEM, 0, 255);
            DlogManager(ITEM);
            if ITEM <> Cancel then begin
               ProcessManage(Land, MANAGEOK, MANDIALOG);
            end else begin
                ShareCrop := OLDSHARE;
                Kind := OLDKIND;
                Rent := OLDRENT;
            end; (* If *)
        until MANAGEOK;
    end; (* With *)
    DisposDialog(MANDIALOG);
end; (* ManageMLand *)

(*****************************************************************************************)
procedure DoConvCheck(ConvDialog:DialogPtr; Item:integer);

(* CALLED BY: InitConvert, ConvertTitles                                                 *)
(* CALLS TO: LightBtn                                                                    *)
(* GLOBALS: none                                                                         *)
(* ACTION: This dialog checks the dialog for converting lesser titled lands for the chosen conversion.  *)

var  DUMMYTYPE : integer;
    ITEMHDL : Handle;
     DUMMYRECT : Rect;

begin (* DoConvCheck *)
    if Item = SEIGRAD then begin
        GetDItem(ConvDialog, VIC1RAD, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        SetCtlValue(Pointer(ITEMHDL), NOTCHECKED);
        LightBtn(ConvDialog, VIC2RAD, BTNACTIVE);
    end else begin
        if Item = VIC1RAD then begin
            GetDItem(ConvDialog, SEIGRAD, DUMMYTYPE, ITEMHDL, DUMMYRECT);
            SetCtlValue(Pointer(ITEMHDL), NOTCHECKED);
            GetDItem(ConvDialog, VIC2RAD, DUMMYTYPE, ITEMHDL, DUMMYRECT);
            SetCtlValue(Pointer(ITEMHDL), NOTCHECKED);
            GetDItem(ConvDialog, MARQRAD, DUMMYTYPE, ITEMHDL, DUMMYRECT);
```

```
            SetCtlValue(Pointer(ITEMHDL), CHECKED);
            LightBtn(ConvDialog, VIC2RAD, BTNINACTIVE);
        end else begin
            GetDItem(ConvDialog, VIC2RAD, DUMMYTYPE, ITEMHDL, DUMMYRECT);
            SetCtlValue(Pointer(ITEMHDL), NOTCHECKED);
            GetDItem(ConvDialog, MARQRAD, DUMMYTYPE, ITEMHDL, DUMMYRECT);
            SetCtlValue(Pointer(ITEMHDL), NOTCHECKED);
        end; (* If *)
    end; (* If *)
    GetDItem(ConvDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    SetCtlValue(Pointer(ITEMHDL), CHECKED);
end; (* DoConvCheck *)

(*****************************************************************************************)
procedure InitConvert(var Assets:AssetsType; ConvDialog:DialogPtr);

(* CALLED BY: ConvertTitles                                                              *)
(* CALLS TO: LightBtn, DoConvCheck                                                       *)
(* GLOBALS: Assets                                                                       *)
(* ACTION: This procedure sets up the conversion dialog with initial values corresponding to lands owned. *)

begin (* InitConvert *)
    with Assets.Land do begin
        if (Seigneurie = NONE) and (Vicomte = NONE) then begin
            LightBtn(ConvDialog, SEIGRAD, BTNINACTIVE);
            LightBtn(ConvDialog, VIC1RAD, BTNINACTIVE);
            LightBtn(ConvDialog, VIC2RAD, BTNINACTIVE);
            LightBtn(ConvDialog, MARQRAD, BTNINACTIVE);
            LightBtn(ConvDialog, OK, BTNINACTIVE);
        end else begin
            if Seigneurie = NONE then begin
                LightBtn(ConvDialog, SEIGRAD, BTNINACTIVE);
                LightBtn(ConvDialog, VIC2RAD, BTNINACTIVE);
                DoConvCheck(ConvDialog, VIC1RAD);
                DoConvCheck(ConvDialog, MARQRAD);
            end else begin
                if Vicomte = NONE then begin
                    LightBtn(ConvDialog, VIC1RAD, BTNINACTIVE);
                    DoConvCheck(ConvDialog, SEIGRAD);
                    DoCOnvCheck(ConvDialog, VIC2RAD);
                end else begin
                    DoConvCheck(ConvDialog, SEIGRAD);
                    DoConvCheck(ConvDialog, VIC2RAD);
                end; (* If *)
            end; (* If *)
        end; (* If *)
    end; (* With *)
end; (* InitConvert *)

(*****************************************************************************************)
procedure DoConvert(var TitleNum, NewTitle, Cash:longint; Bound, Cost:longint; var ConvOK:boolean);
```

```
(* CALLED BY: ConvertTitles                                                    *)
(* CALLS TO: NumSpecs                                                         *)
(* GLOBALS: none                                                              *)
(* ACTION: This procedure checks for the proper number of acres and subtracts the cost of a   *)
(* conversion.                                                                *)

var  LEN : integer;
     BOUNDSTR, COSTSTR : Str255;

begin (* DoConvert *)
    if TitleNum < Bound then begin
        NumSpecs(Bound, LEN, BOUNDSTR);
        ParamText(BOUNDSTR, '', '', '');
        LEN := StopAlert(294, nil);
        ConvOK := false;
    end else begin
        NewTitle := NewTitle + TitleNum;
        TitleNum := NONE;
        Cash := Cash - Cost;
        NumSpecs(Cost, LEN, COSTSTR);
        ParamText(COSTSTR, '','','');
        LEN := NoteAlert(332, nil);
    end; (* If *)
end; (* DoConvert *)

(*****************************************************************************************)
function ItemChecked(TheDialog:DialogPtr; Item:integer) : boolean;

(* CALLED BY: ConvertTitles                                                    *)
(* CALLS TO: none                                                             *)
(* GLOBALS: none                                                              *)
(* ACTION: This function returns the value of an item in the dialog.          *)

var  DUMMYTYPE, VAL : integer;
    ITEMHDL : Handle;
     DUMMYRECT : Rect;

begin (* ItemChecked *)
    GetDItem(TheDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    VAL := GetCtlValue(Pointer(ITEMHDL));
    if VAL = CHECKED then begin
        ItemChecked := true;
    end else begin
        ItemChecked := false;
    end; (* If *)
end; (* ItemChecked *)

(*****************************************************************************************)
procedure ConvertTitles(var Assets:AssetsType);

(* CALLED BY: ManageLand                                                       *)
(* CALLS TO: InitConvert, DoConvCheck, NumSpecs                               *)
```

```
(* GLOBALS: Assets                                                    *)
(* ACTION: This procedure converts lesser titled lands into greater titled lands.  *)

var  CONVDIALOG : DialogPtr;
    ITEM, LEN : integer;
    CONVOK : boolean;
    DIFF, COST : longint;
    DIFFSTR, COSTSTR : Str255;
    CHANGED : ConvType;

begin (* ConvertTitles *)
    CONVDIALOG := GetNewDialog(276, nil, Pointer(-1));
    with Assets.Land do begin
        InitConvert(Assets, CONVDIALOG);
        repeat
            CONVOK := true;
            repeat
                SystemTask;
                ModalDialog(nil, ITEM);
                if ITEM in [SEIGRAD, VIC1RAD, VIC2RAD, MARQRAD] then begin
                    DoConvCheck(CONVDIALOG, ITEM);
                end; (* If *)
            until ITEM in [OK, Cancel];
            if ITEM = OK then begin
                if ItemChecked(CONVDIALOG, SEIGRAD) then begin
                    if ItemChecked(CONVDIALOG, VIC2RAD) then begin
                        DIFF := VICOMTECOST - SEIGCOST;
                        CHANGED := SeigToVic;
                    end else begin
                        DIFF := MARQCOST - SEIGCOST;
                        CHANGED := SeigToMarq;
                    end; (* If *)
                    COST := DIFF * Seigneurie;
                end else begin
                    DIFF := MARQCOST - VICOMTECOST;
                    COST := DIFF * Vicomte;
                    CHANGED := VicToMarq;
                end; (* If *)
                NumSpecs(DIFF, LEN, DIFFSTR);
                NumSpecs(COST, LEN, COSTSTR);
                ParamText(DIFFSTR, COSTSTR, '', '');
                if Assets.Cash < COST then begin
                    LEN := StopAlert(292, nil);
                    CONVOK := false;
                end else begin
                LEN := CautionAlert(293, nil);
                if LEN = OK then begin
                    case CHANGED of
                        SeigToVic : DoConvert(Seigneurie, Vicomte, Assets.Cash, VICOMTELOWER, COST, CONVOK);
                        SeigToMarq : DoConvert(Seigneurie, Marquisat, Assets.Cash, MARQLOWER, COST, CONVOK);
                        VicToMarq : DoConvert(Vicomte, Marquisat, Assets.Cash, MARQLOWER, COST, CONVOK);
                    end; (* Case *)
```

```
            end else begin
                CONVOK := false;
            end; (* If *)
             end; (* If *)
          end; (* If *)
       until CONVOK;
   end; (* With *)
   DIsposDialog(CONVDIALOG);
end; (* ConvertTitles *)

(*****************************************************************************************)
procedure ManageLand(var Assets:AssetsType; Date:DateType);

(* CALLED BY: DoCommmand, Main                                                          *)
(* CALLS TO: ConvertTitles, ManageMLand, DisplayAssets                                  *)
(* GLOBALS: Assets, Date                                                                *)
(* ACTION: This procedure calls ConvertTitles if the season is Fall, and gives a choice of Managing  *)
(* Miscellaneous land or converting titled lands if the season is Spring.               *)

var  LEN : integer;

begin (* ManageLand *)
   if Date.Fall then begin
      ConvertTitles(Assets);
   end else begin
      LEN := Alert(289, nil);
      if LEN = OK then begin
         ManageMLand(Assets.Land);
      end else begin
         ConvertTitles(Assets);
      end; (* If *)
   end; (* If *)
   DisplayAssets(Assets, Date);
end; (* ManageLand *)

{$S Seg9}

(*****************************************************************************************)
procedure BuyMiscLand(var Assets:AssetsType; FinMenu:MenuHandle; Date:DateType);

(* CALLED BY: BuyLand                                                                   *)
(* CALLS TO: NumSpecs, ConvertNum                                                       *)
(* GLOBALS: Assets, Date                                                                *)
(* ACTION: This procedure sets prices and quantities for the purchase of miscellaneous land.  *)

var  LANDDIALOG : DialogPtr;
   LEN, ITEM : integer;
   AMT, AVAILSTR, LANDCOST, BOUGHTSTR, COSTSTR, MAXPURSTR : Str255;
   CONVOK : boolean;
   TEMPLANDPUR, TEMP, COST, MISCLANDAVAIL : longint;

begin (* BuyMiscLand *)
```

```
COST := LANDVAL;
MISCLANDAVAIL := (MaxMiscLand - Assets.Land.Inherited - Assets.Land.Bought);
if (Assets.Protector.Name = Conde) or (Assets.Protector.Name = GrandDauphin) then
 begin
    if (Date.Year >= 1653) and (Assets.Protector.Name = Conde) then
        COST := (LANDVAL * 3) div 4
     else
        COST := LANDVAL div 2;
end; (* If *)
If (Assets.Cash div COST) <= MISCLANDAVAIL then
    TEMPLANDPUR := Assets.Cash div COST
    else
   TEMPLANDPUR := MISCLANDAVAIL;
 NumSpecs(Assets.Cash, LEN, AMT);
 NumSpecs(TEMPLANDPUR, LEN, MAXPURSTR);
 NumSpecs(COST, LEN, LANDCOST);
NumSpecs(MISCLANDAVAIL, LEN, AVAILSTR);
 ParamText(AMT, MAXPURSTR, LANDCOST, AVAILSTR);
LANDDIALOG := GetNewDialog(260, nil, Pointer(-1));

repeat
     CONVOK := true;
    SelIText (LANDDIALOG, LANDBUYITEM, 0, 200);
    DlogManager(ITEM);
    if ITEM = OK then begin
     GetDText(LANDDIALOG, LANDBUYITEM, AMT);
      ConvertNum(AMT, TEMP, CONVOK);
      if not CONVOK then LEN := StopAlert(256, nil)  (* Checks for incorrect typing *)
            else   (* Level one *)
                  if TEMP * COST > Assets.Cash then begin
                        LEN := StopAlert(258, nil);
                        CONVOK := false;
                        end (* If too little cash *)
            else  (* Level two *)
                  if TEMP > MISCLANDAVAIL then begin
                        LEN := StopAlert(323, nil);
                        CONVOK := false;
                       end; (* If too much land *)
    end; (* If Item OK*)

until CONVOK;
if (ITEM = OK) and (TEMP <> NONE) then begin

    Assets.Land.Bought := Assets.Land.Bought + TEMP;
     Assets.Cash := Assets.Cash - (TEMP * COST);
    NumSpecs(TEMP, LEN, BOUGHTSTR);
     NumSpecs(TEMP * COST, LEN, COSTSTR);
     ParamText(BOUGHTSTR, COSTSTR, '', '');
    LEN := Notealert(324, nil);
    EnableItem(FinMenu, SELLITEM);
end; (* If *)
```

```
    DisposDialog(LANDDIALOG);
end; (* BuyMiscLand *)

(*****************************************************************************************)
procedure BuyTitledLand(var Assets:AssetsType);

(* CALLED BY: BuyLand                                                                    *)
(* CALLS TO: NumSpecs, ConvertNum, SellText                                              *)
(* GLOBALS: Assets                                                                       *)
(* ACTION: This procedure displays a dialog for the purchase of titled lands.            *)

var  TITLEDIALOG : DialogPtr;
    ITEM, LEN : integer;
     SEIGSTR, VICOMTESTR, MARQSTR, CASHSTR, COSTSTR: Str255;
    SEIGNUM, VICOMTENUM, MARQNUM, VAL : longint;
    CONVOK : boolean;

begin (* BuyTitledLand *)
    NumSpecs(SEIGCOST, LEN, SEIGSTR);
    NumSpecs(VICOMTECOST, LEN, VICOMTESTR);
    NumSpecs(MARQCOST, LEN, MARQSTR);
    NumSpecs(Assets.Cash, LEN, CASHSTR);
    ParamText(SEIGSTR, VICOMTESTR, MARQSTR, CASHSTR);
   TITLEDIALOG := GetNewDialog(275, nil, Pointer(-1));
   repeat
       CONVOK := true;
       SelIText(TITLEDIALOG, SEIGITEM, 0, 255);
       DlogManager(ITEM);
       if ITEM = OK then begin
          GetDText(TITLEDIALOG, SEIGITEM, SEIGSTR);
          GetDText(TITLEDIALOG, VICOMTEITEM, VICOMTESTR);
          GetDText(TITLEDIALOG, MARQITEM, MARQSTR);
           ConvertNum(SEIGSTR, SEIGNUM, CONVOK);
           ConvertNum(VICOMTESTR, VICOMTENUM, CONVOK);
           ConvertNum(MARQSTR, MARQNUM, CONVOK);
        if not CONVOK then begin
            LEN := StopAlert(256, nil);
        end else begin
        if ((SEIGNUM < SEIGLOWER) and (SEIGNUM <> 0)) or (SEIGNUM > SEIGUPPER) or
          ((VICOMTENUM < VICOMTELOWER) and (VICOMTENUM <> 0)) or
                                              (VICOMTENUM > VICOMTEUPPER) or
          ((MARQNUM < MARQLOWER) and (MARQNUM <> 0)) or
                                        (MARQNUM > MARQUPPER) then begin
              CONVOK := false;
              LEN := StopAlert(290, nil);
        end else begin
                  VAL := (SEIGCOST * SEIGNUM) + (VICOMTECOST * VICOMTENUM) +
                                                              (MARQCOST * MARQNUM);
                  if Assets.Cash < VAL then begin
                     CONVOK := false;
                     LEN := StopAlert(291, nil);
                  end else begin
```

```
                    Assets.Cash := Assets.Cash - VAL;
                    Assets.Land.Seigneurie := Assets.Land.Seigneurie + SEIGNUM;
                    Assets.Land.Vicomte := Assets.Land.Vicomte + VICOMTENUM;
                    Assets.Land.Marquisat := Assets.Land.Marquisat + MARQNUM;
                    VAL := (SEIGCOST * SEIGNUM) + (VICOMTECOST * VICOMTENUM) +
                                                                (MARQCOST * MARQNUM);
                    NumSpecs(VAL, LEN, COSTSTR);
                    ParamText(COSTSTR, ",",");
                    if VAL <> 0 then begin
                       LEN := NoteAlert(332, nil);
                    end; (* If *)
                 end; (* If *)
              end; (* If *)
           end; (* If *)
        end; (* If *)
     until CONVOK;
     DisposDialog(TITLEDIALOG);
end; (* BuyTitledLand *)

(*****************************************************************************************)
procedure BuyLand(var Assets:AssetsType; FinMenu:MenuHandle; Date:DateType);

(* CALLED BY:                                                                            *)
(* CALLS TO: BuyMiscLand, BuyTitledLand                                                  *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure queries the user on land purchase:  Miscellaneous or Titled lands? and calls  *)
(* those proceduresin response to the dialog button chosen.                              *)

var  LANDDIALOG : DialogPtr;
     ITEM : integer;

begin (* BuyLand *)
     LANDDIALOG := GetNewDialog(270, nil, Pointer(-1));
     DlogManage(ITEM);
     DisposDialog(LANDDIALOG);
     if ITEM = OK then begin
        BuyMiscLand(Assets, FinMenu, Date);
     end else begin
        BuyTitledLand(Assets);
     end; (* If *)
end; (* BuyLand *)

(*****************************************************************************************)
procedure BuyTextiles(var Assets:AssetsType; Date:DateType);

(* CALLED BY: Purchase                                                                   *)
(* CALLS TO: NumSpecs, SellText, ConvertNum                                              *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure displays a dialog for investment in textiles.                  *)

var  TEXTDIALOG : DialogPtr;
     CONVOK : boolean;
```

```
    ITEM, DUMMYTYPE, LEN : integer;
    ITEMHDL : Handle;
    DUMMYRECT : Rect;
    AMT, AMTINV : Str255;
    OLD, TEMP : longint;

begin (* BuyTextiles *)
    OLD := Assets.Textiles;
    Assets.Cash := Assets.Cash + OLD;
    Assets.Textiles := 0;
    TEXTDIALOG := GetNewDialog(258, nil, Pointer(-1));
    NumSpecs(Assets.Cash, LEN, AMT);
    ParamText(AMT, '', '', '');
    GetDItem(TEXTDIALOG, TEXTBUYITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    NumSpecs(OLD, LEN, AMT);
    SetIText(ITEMHDL, AMT);
    SelIText(TEXTDIALOG, TEXTBUYITEM, 0, 255);
    repeat
        CONVOK := true;
        SelIText (TEXTDIALOG, TEXTBUYITEM, 0, 200);
        DlogManager(ITEM);
        if ITEM = OK then begin
            GetDText(TEXTDIALOG, TEXTBUYITEM, AMT);
            ConvertNum(AMT, TEMP, CONVOK);
            if not CONVOK then begin
                LEN := StopAlert(256, nil);
            end else begin
                if TEMP > Assets.Cash then begin
                    LEN := CautionAlert(258, nil);
                    CONVOK := false;
                end; (* If *)
            end; (* If *)
        end; (* If *)
    until CONVOK;
    if ITEM = OK then begin
        Numspecs(TEMP, LEN, AMTINV);
        Paramtext(AMTINV,'','');
        LEN := NoteAlert(328, nil);
        Assets.Textiles := TEMP;
        Assets.Cash := Assets.Cash - TEMP;
    end else begin
        Assets.Cash := Assets.Cash - OLD;
        Assets.Textiles := OLD;
    end; (* If *)

    DisposDialog(TEXTDIALOG);
end; (* BuyTextiles *)

(*****************************************************************************)
procedure AddRente(CostDenier:integer; Date:DateType; var IndivRentes:RenteHandle);

(* CALLED BY: BuyRente                                                       *)
```

```
(* CALLS TO: none                                                                    *)
(* GLOBALS: Date                                                                     *)
(* ACTION: This procedure adds a rente to the renteList with appropriate Rentetype information.  *)

var  REC : RenteType;
   MARKER : RenteHandle;

begin (* AddRente *)
   REC.Year := Date.Year;
    REC.Fall := Date.Fall;
    REC.CostDenier := CostDenier;
    REC.Next := nil;
    if IndivRentes = nil then begin
       IndivRentes := Pointer(ord(NewHandle(SizeOf(RenteType))));
       IndivRentes^^ := REC;
    end else begin
        MARKER := IndivRentes;
        while MARKER^^.Next <> nil do begin
            MARKER := MARKER^^.Next;
        end; (* While *)
       MARKER^^.Next := Pointer(ord(NewHandle(SizeOf(RenteType))));
       MARKER^^.Next^^ := REC;
   end; (* If *)
end; (* AddRente *)

(*****************************************************************************************)
function NumKRente (IndivRentes:RenteHandle) : longint;

(* CALLED BY: BuyRente, SellKRente                                                   *)
(* CALLS TO: none                                                                    *)
(* GLOBALS: none                                                                     *)
(* ACTION: This function retruns the number of King's Rentes held by the player.     *)

var  TEMP : longint;

begin (* NumKRente *)
    TEMP := 0;
    while IndivRentes <> nil do begin
        TEMP := TEMP + 1;
       IndivRentes := IndivRentes^^.Next;
    end; (* While *)
    NumKRente := TEMP;
end; (* NumKRente *)

(*****************************************************************************************)
procedure BuyRente(Date:DateType; var Assets:AssetsType);

(* CALLED BY: Purchase                                                               *)
(* CALLS TO: NumSpecs, SellText, ConvertNum, NumKRente, AddRente                     *)
(* GLOBALS: Assets, Date                                                             *)
(* ACTION: This procedure displays a dialog for the purchase of rentes, and calls AddRente to store them.  *)
```

```
var RENTEDIALOG : DialogPtr;
    ITEM, LEN, I : integer;
    FACESTR, COSTSTR, CASHSTR, RENTESTR, AMT, PRICESTR, NUMRENTESTR : Str255;
    PRICE, TEMP : longint;
    CONVOK : boolean;

begin (* BuyRente *)
   RENTEDIALOG := GetNewDialog(259, nil, Pointer(-1));
   with Assets.Rente do begin
      NumSpecs(FaceDenier, LEN, FACESTR);
      NumSpecs(CostDenier, LEN, COSTSTR);
      NumSpecs(Assets.Cash, LEN, CASHSTR);
      ParamText(FACESTR, COSTSTR, CASHSTR, '');
      repeat
         CONVOK := true;
         SelIText (RENTEDIALOG, RENTEBUYITEM, 0, 200);
         DlogManager(ITEM);
         if ITEM = OK then begin
            GetDText(RENTEDIALOG, RENTEBUYITEM, AMT);
            ConvertNum(AMT, TEMP, CONVOK);
            if not CONVOK then begin
               LEN := StopAlert(256, nil);
            end else begin
               if TEMP + NumKRente(IndivRentes) > MAXKRENTE then begin
                  Numspecs(MAXKRENTE ,LEN, RENTESTR);
                  Paramtext(RENTESTR,',',',');
                  LEN := StopAlert(308, nil);
                  CONVOK := false;
               end else begin
                  PRICE := ((Return * 1000) div (1000 div CostDenier)) * TEMP;
                  if PRICE > Assets.Cash then begin
                     LEN := StopAlert(259, nil);
                     CONVOK := false;
                  end; (* If *)
               end; (* If *)
            end; (* If *)
         end; (* If *)
      until CONVOK;
      if (ITEM = OK) and (TEMP > 0) then begin
         Assets.Cash := Assets.Cash - PRICE;
         for I := 1 to TEMP do begin
            AddRente(CostDenier, Date, IndivRentes);
         end; (* For *)
         NumSpecs(TEMP, LEN, NUMRENTESTR);
         NumSpecs(FaceDenier, LEN, FACESTR);
         NumSpecs(PRICE, LEN, PRICESTR);
         ParamText(NUMRENTESTR, FACESTR, PRICESTR, '');
         LEN := NoteAlert(333, nil);
      end; (* If *)
   end; (* With *)
   DisposDialog(RENTEDIALOG);
end; (* BuyRente *)
```

```
(*****************************************************************************)
procedure DoCheck(OfficeDialog:DialogPtr; Item:integer; NumOffices:longint);

(* CALLED BY: BuyOffice, SellOffice                                          *)
(* CALLS TO: LightBtn                                                        *)
(* GLOBALS: none                                                             *)
(* ACTION: This procedure checks the office dialog for checked items and returns a checked value in Item.  *)

var  DUMMYTYPE : integer;
    ITEMHDL, RADHDL : Handle;
    DUMMYRECT : Rect;
    VAL : integer;

begin (* DoCheck *)
   GetDItem(OfficeDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
   VAL := GetCtlValue(Pointer(ord(ITEMHDL)));
   if VAL = CHECKED then begin
       VAL := NOTCHECKED;
      LightBtn(OfficeDialog, BUYBTNITEM, BTNINACTIVE);
       if NumOffices > NUMOFFPERSCREEN then begin
          LightBtn(OfficeDialog, NEXTBTNITEM, BTNACTIVE);
          LightBtn(OfficeDialog, PREVBTNITEM, BTNACTIVE);
       end; (* If *)
   end else begin
       VAL := CHECKED;
      LightBtn(OfficeDialog, BUYBTNITEM, BTNACTIVE);
      LightBtn(OfficeDialog, NEXTBTNITEM, BTNINACTIVE);
      LightBtn(OfficeDialog, PREVBTNITEM, BTNINACTIVE);
       GetDItem(OfficeDialog, RAD1, DUMMYTYPE, RADHDL, DUMMYRECT);
       SetCtlValue(Pointer(RADHDL), NOTCHECKED);
       GetDItem(OfficeDialog, RAD2, DUMMYTYPE, RADHDL, DUMMYRECT);
       SetCtlValue(Pointer(RADHDL), NOTCHECKED);
       GetDItem(OfficeDialog, RAD3, DUMMYTYPE, RADHDL, DUMMYRECT);
       SetCtlValue(Pointer(RADHDL), NOTCHECKED);
       GetDItem(OfficeDialog, RAD4, DUMMYTYPE, RADHDL, DUMMYRECT);
       SetCtlValue(Pointer(RADHDL), NOTCHECKED);
   end; (* If *)
   SetCtlValue(Pointer(ITEMHDL), VAL);
end; (* DoCheck *)

(*****************************************************************************)
procedure AdvanceOffice(var Pos: integer; Direction, NumOffices, RefNum:integer; var ShownOffices:OffArray;
                  Date:DateType);

(* CALLED BY: BuyOffice                                                      *)
(* CALLS TO: NumSpecs                                                        *)
(* GLOBALS: Date                                                             *)
(* ACTION: This procedure displays a new set of offices in the buy office dialog.  *)

var  ERR : OSErr;
    STRLEN, VALLEN,BOOLLEN, OFFRECLEN, VALUE, ALTEREDVAL, PRESTIGE : longint;
```

```
    TITLE, VALSTR : Str255;
    I, LEN : integer;
    STR : array [1..NUMOFFPERSCREEN] of Str255;
    NOBILITY, TITANDNOB : boolean;

begin (* AdvanceOffice *)
    if Direction = PREVBTNITEM then begin
        Pos := Pos - (2 * NUMOFFPERSCREEN);
        if Pos < 0 then begin
            Pos := NumOffices + Pos;
        end; (* If *)
        if Pos < 0 then begin
            Pos := NumOffices + Pos;
        end; (* If *)
    end; (* If *)
    STRLEN := SizeOf(TITLE);
    VALLEN := SizeOf(VALUE);
    BOOLLEN := SizeOf(boolean);
    OFFRECLEN := STRLEN + (2 * VALLEN) + (2 * BOOLLEN);
    for I := 1 to NUMOFFPERSCREEN do begin
        ERR := SetFPos(RefNum, 1, Pos * OFFRECLEN + SizeOf(integer));
        Pos := Pos + 1;
        if Pos >= NumOffices then begin
            Pos := 0;
        end; (* If *)
        ERR := FSRead(RefNum, STRLEN, @TITLE);
        ERR := FSRead(RefNum, VALLEN, @VALUE);
        ERR := FSRead(RefNum, VALLEN, @PRESTIGE);
        ERR := FSRead(RefNum, BOOLLEN, @TITANDNOB);
        ERR := FSRead(RefNum, BOOLLEN, @NOBILITY);
        if (Date.Year >= 1642) and (Date.Year <= 1652) then begin
            ALTEREDVAL := (VALUE * GLUTPERCENT) div 100;
            NumSpecs(ALTEREDVAL, LEN, VALSTR);
        end else begin
            NumSpecs(VALUE, LEN, VALSTR);
        end; (* If *)
        STR[I] := Concat('£', VALSTR,' -- ',TITLE);
        ShownOffices[I].Title := TITLE;
        ShownOffices[I].Value := VALUE;
        ShownOffices[I].Prestige := PRESTIGE;
        ShownOffices[I].Nobility := NOBILITY;
        ShownOffices[I].TitAndNob := TITANDNOB;
    end; (* For *)
    ParamText(STR[1], STR[2], STR[3], STR[4]);
end; (* AdvanceOffice *)

{$S}

(*****************************************************************************************)
procedure ExaOffItem(OfficeDialog:DialogPtr; Item, ItemNum:integer; var NewOffice:DlogOffRec;
                     ShownOffices:OffArray; var Found:boolean);
```

```
(* CALLED BY: GetOfficeBought                                                    *)
(* CALLS TO: none                                                               *)
(* GLOBALS: none                                                                *)
(* ACTION: This procedure reads the value of a checked item in the office dialog and sets NewOffice to  *)
(* that value.                                                                  *)

var  DUMMYTYPE, VAL : integer;
    ITEMHDL : Handle;
    DUMMYRECT : Rect;

begin (* ExaOffItem *)
    GetDItem(OfficeDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    VAL := GetCtlValue(Pointer(ITEMHDL));
    if VAL = CHECKED then begin
        NewOffice := ShownOffices[ItemNum];
        Found := true;
    end; (* If *)
end; (* ExaOffItem *)

(*******************************************************************************)
procedure GetOffBought(OfficeDialog:DialogPtr; ShownOffices:OffArray; var NewOffice:DlogOffRec);

(* CALLED BY: BuyOffice, SellOffice                                             *)
(* CALLS TO: ExaOffItem                                                         *)
(* GLOBALS: none                                                                *)
(* ACTION: This procedure polls the buy office dialog for checked items and returns the choice made.  *)

var  FOUND : boolean;

begin (* GetOffBought *)
    FOUND := false;
    ExaOffItem(OfficeDialog, RAD1, 1, NewOffice, ShownOffices, FOUND);
    ExaOffItem(OfficeDialog, RAD2, 2, NewOffice, ShownOffices, FOUND);
    ExaOffItem(OfficeDialog, RAD3, 3, NewOffice, ShownOffices, FOUND);
    ExaOffItem(OfficeDialog, RAD4, 4, NewOffice, ShownOffices, FOUND);
    if not FOUND then begin
        NewOffice.Value := NONE;
    end; (* If *)
end; (* GetOffBought *)

{$S Seg9}

(*******************************************************************************)
procedure AddOffice(NewOffice:DlogOffRec; var Office:OfficeType; Date:DateType);

(* CALLED BY: BuyOffice                                                         *)
(* CALLS TO: none                                                               *)
(* GLOBALS: Date                                                                *)
(* ACTION: This procedure adds an office record to the list of offices held by the player.  *)

var  TEMP : longint;
    TEMPOFFICE : OfficeRec;
```

```
    MARKER : OfficeHandle;
    STOP : boolean;

begin (* AddOffice *)
    TEMP := abs(Random) div OFFDIVIDER + 10;
    with Office do begin
        TEMP := ((NewOffice.Value * TEMP) div 100);
        Salary := Salary + TEMP;
        TotPurchase := TotPurchase + NewOffice.Value;
        if (Date.Year >= 1642) and (Date.Year <=1652) then begin
            TotPurchase := TotPurchase - ((NewOffice.Value * LEVYTAX) div 100);
        end; (* If *)
        Number := Number + 1;
        TEMPOFFICE.Inherited := false;
        TEMPOFFICE.Title := NewOffice.Title;
        TEMPOFFICE.Value := NewOffice.Value;
        TEMPOFFICE.Salary := TEMP;
        TEMPOFFICE.AmtPaid := NewOffice.Value;
        if (Date.Year >= 1642) and (Date.Year <=1652) then begin
            TEMPOFFICE.AmtPaid := TEMPOFFICE.AmtPaid - ((NewOffice.Value * LEVYTAX) div 100);
        end; (* If *)
        TEMPOFFICE.Next := nil;
        TEMPOFFICE.Prestige := NewOffice.Prestige;
        if OfficeList = nil then begin
            OfficeList := Pointer(NewHandle(SizeOf(TEMPOFFICE)));
            OfficeList^^ := TEMPOFFICE;
        end else begin
            if TEMPOFFICE.Prestige >= OfficeList^^.Prestige then begin
                TEMPOFFICE.Next := OfficeList;
                OfficeList := Pointer(NewHandle(SizeOf(TEMPOFFICE)));
                OfficeList^^ := TEMPOFFICE;
            end else begin
                MARKER := OfficeList;
                STOP := false;
                while (MARKER^^.Next <> nil) and (not STOP) do begin
                    if TEMPOFFICE.Prestige >= MARKER^^.Next^^.Prestige then begin
                        STOP := true;
                    end else begin
                        MARKER := MARKER^^.Next;
                    end; (* If *)
                end; (* While *)
                TEMPOFFICE.Next := MARKER^^.Next;
                MARKER^^.Next := Pointer(NewHandle(SizeOf(TEMPOFFICE)));
                MARKER^^.Next^^ := TEMPOFFICE;
            end; (* If *)
        end; (* If *)
    end; (* With *)
end; (* AddOffice *)

(*****************************************************************************************)
function Credentials(Title, ReqTitle:Str255; OfficeList:OfficeHandle) : boolean;
```

```
(* CALLED BY: BuyOffice                                                              *)
(* CALLS TO: none                                                                    *)
(* GLOBALS: Assets                                                                   *)
(* ACTION: This function checks special offices for appropriate credentials of age, other offices held or  *)
(* inherited.  It returns a value of true if credentials are appropriate.           *)

var OLDENUF, HOLDOFFICE, INHERITEDOFF : boolean;

begin (* Credentials *)
 Credentials := true;
 If (Title = CHANCELLOR) or (Title = SECOFSTATE) then
  begin
    Credentials := false;
    HOLDOFFICE := false;
    OLDENUF := false;
    INHERITEDOFF := false;

    if OfficeList <> nil then begin
      while (OfficeList^^.Next <> nil) and (not HOLDOFFICE) do
        begin
          if OfficeList^^.Title = ReqTitle then
               HOLDOFFICE := true;
             if OfficeList^^.Inherited then INHERITEDOFF := true
           else
             OfficeList := OfficeList^^.Next;
        end; (* While *)
       if OfficeList^^.Title = ReqTitle then begin
            HOLDOFFICE := true;
          if OfficeList^^.Inherited then INHERITEDOFF := true
       end; (* If *)
    end; (* If *)

    if (Title = CHANCELLOR) and (Assets.Age >= 45) then OLDENUF := true;
    if (Title = SECOFSTATE) and (Assets.Age >= 35) then OLDENUF := true;

    if (OLDENUF and HOLDOFFICE and INHERITEDOFF) then CREDENTIALS := true;
 end; (* While *)
end; (* Credentials *)

(*****************************************************************************)
function OwnOffice(Title:Str255; OfficeList:OfficeHandle) : boolean;

(* CALLED BY: BuyOffice                                                              *)
(* CALLS TO: none                                                                    *)
(* GLOBALS: none                                                                     *)
(* ACTION: This function checks the player's office list to see if an office is already held by the player.  *)

begin (* OwnOffice *)
    TEMP := false;
    if OfficeList <> nil then begin
       while (OfficeList^^.Next <> nil) and (not TEMP) do begin
           if OfficeList^^.Title = Title then begin
```

```
            TEMP := true;
          end else begin
            OfficeList := OfficeList^^.Next;
          end; (* If *)
        end; (* While *)
        if OfficeList^^.Title = Title then begin
            TEMP := true;
        end; (* If *)
    end; (* If *)
    OwnOffice := TEMP;
end; (* OwnOffice *)

(*****************************************************************************************)
procedure BuyOffice (var Assets:AssetsType; Date:DateType; FinMenu, PersMenu:MenuHandle);

(* CALLED BY: Purchase                                                                   *)
(* CALLS TO: AdvanceOffice, LightBtn, DoCheck, GetOffBought, DisplayAssets, OwnOffice, Credentials*)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure displays a dialog for the purchase of offices and checks for cash, title,  *)
(* nobility, and credentials needed for purchase.                                        *)

var  OFFICEDIALOG : DialogPtr;
    ITEM, LEN, NUMOFFICES, REFNUM, POS: integer;
    RECLEN, COST, TITLAND: longint;
    ERR : OSErr;
    SHOWNOFFICES : OffArray;
    NEWOFFICE : DlogOffRec;
    BUYOK : boolean;
    COSTofLSTR, SALARYSTR, REQTITLE : Str255;

begin (* BuyOffice *)
    ERR := FSOpen(OFFICEFILE, 0, REFNUM);
    RECLEN := SizeOf(NUMOFFICES);
    ERR := FSRead(REFNUM, RECLEN, @NUMOFFICES);
    POS := 0;
    AdvanceOffice(POS, NEXTBTNITEM, NUMOFFICES, REFNUM, SHOWNOFFICES, Date);
    OFFICEDIALOG := GetNewDialog(267, nil, Pointer(-1));
   LightBtn(OfficeDialog, BUYBTNITEM, BTNINACTIVE);
    repeat
        BUYOK := true;
        repeat
            SystemTask;
            ModalDialog(nil, ITEM);
            if ITEM in [RAD1, RAD2, RAD3, RAD4] then begin
                DoCheck(OFFICEDIALOG, ITEM, NUMOFFICES);
            end; (* If *)
            if ITEM in [NEXTBTNITEM, PREVBTNITEM] then begin
                AdvanceOffice(POS, ITEM, NUMOFFICES, REFNUM, SHOWNOFFICES, Date);
                DrawDialog(OFFICEDIALOG);
            end; (* If *)
        until ITEM in [BUYBTNITEM, Cancel];
        if ITEM = BUYBTNITEM then
```

```
begin
   GetOffBought(OFFICEDIALOG, SHOWNOFFICES, NEWOFFICE);
    COST := NEWOFFICE.Value;
  TITLAND := Assets.Land.Seigneurie + Assets.Land.Vicomte + Assets.Land.Marquisat;
   REQTITLE := PRESIDENT;
 if (Date.Year >= 1642) and (Date.Year <= 1652) then
      COST := COST - ((NEWOFFICE.Value * LEVYTAX) div 100);

 if NEWOFFICE.Value = NONE then BUYOK := false
   else (* one *)
      if Assets.Cash < COST then begin
          LEN := CautionAlert(267, nil);
          BUYOK := false
          end
      else (* two *)
         if OwnOffice(NEWOFFICE.Title, Assets.Office.OfficeList) then begin
             LEN := CautionAlert(268, nil);
             BUYOK := false
             end
          else (* three *)
             if (NEWOFFICE.Nobility) and (not Assets.Noble) then begin
                 LEN := StopAlert(307, nil);
                 BUYOK := false
                 end
              else (* four *)
              if (NEWOFFICE.TitAndNob) and ((not Assets.Noble) or (TITLAND = 0)) then begin
                 LEN := StopAlert(309, nil);
                 BUYOK := false
                 end
              else (* five *)
             if not Credentials(NEWOFFICE.Title, REQTITLE, Assets.Office.OfficeList) then
                    begin
                    LEN := StopAlert(327, nil);
                    BUYOK := false
                    end
                else (* six *)
                  begin
                  AddOffice(NEWOFFICE, Assets.Office, Date);
                  if NEWOFFICE.Prestige = AMBITIOUS then
                         Assets.TooAmbitious := Assets.TooAmbitious + 1;
                  EnableItem(FinMenu, SELLITEM);
                  Assets.Cash := Assets.Cash - COST;

                  if NEWOFFICE.Title = SECYKING then
                      begin
                      LEN := NoteAlert(287, nil);
                      Assets.Noble := true;
                      DisableItem(PersMenu, NOBLEITEM);
                      end; (* If *)

                  ParamText(NEWOFFICE.Title, '', '', '');
                  LEN := Notealert(321, nil);
```

```
                        DisplayAssets(Assets, Date);
                        NumSpecs(Assets.Office.Salary, LEN, SALARYSTR);
                        NumSpecs(Assets.CostOfLiving, LEN, COSTofLSTR);
                        ParamText(SALARYSTR, COSTofLSTR, '', '');
                        LEN := Notealert(320, nil)
                end; (* else six and IF*)

        end; (* If *)
    until BUYOK;
    ERR := FSClose(REFNUM);
    DisposDialog(OFFICEDIALOG);
end; (* BuyOffice *)

{*****************************************************************************************}
procedure BuyLease(var Assets:AssetsType; FinMenu:MenuHandle; Date:DateType);

(* CALLED BY: Purchase                                                                    *)
(* CALLS TO: NumSpecs                                                                     *)
(* GLOBALS: Assets, Date                                                                  *)
(* ACTION: This procedure displays a dialog for the purchase of leases.                   *)

var  LEN, ITEM : integer;
    LEASEDIALOG : DialogPtr;
    FACESTR, OFFSTR, OFFERSTR : Str255;

begin (* BuyLease *)
    if Assets.Protector.Name = DukeOfBurgundy then begin
        LEN := StopAlert(316, nil);
    end else begin
        with Assets.Lease do begin
            NumSpecs(FaceValue, LEN, FACESTR);
            NumSpecs(Offer, LEN, OFFERSTR);
            NumSpecs((FaceValue * OFFICIAL) div 1000, LEN, OFFSTR);
            LEASEDIALOG := GetNewDialog(263, nil, Pointer(-1));
            ParamText(Title, FACESTR, OFFSTR, OFFERSTR);
            DlogManager(ITEM);
            if ITEM = OK then begin
                if Assets.Cash < Offer then begin
                    ITEM := CautionAlert(264, nil);
                end else begin
                    Assets.Cash := Assets.Cash - Offer;
                    Assets.Lease.Bought := true;
                    Assets.Lease.NumBought := Assets.Lease.NumBought + 1;
                    ParamText(Title, ',', ',');
                    LEN := NoteAlert(335, nil);
                    DisableItem(FinMenu, BUYITEM);
                    if Date.Year = 1639 then begin
                        Hanged := true;
                    end; (* If *)
                end; (* If *)
            end; (* If *)
```

```
            DisposDialog(LEASEDIALOG);
        end; (* With *)
    end; (* If *)
end; (* BuyLease *)

(*****************************************************************************************)
procedure Purchase(var Assets:AssetsType; Choice:integer; Date:DateType; FinMenu, PersMenu:MenuHandle);

(* CALLED BY: DoCommand, Main                                                            *)
(* CALLS TO: BuyLand, BuyRente, BuyOffice, BuyLease, BuyTextiles                         *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure checks the investments icons for the appropriate investment and calls *)
(* purchase procedures.                                                                  *)

begin (* Purchase *)
    case Choice of
        LANDICON : BuyLand(Assets, FinMenu, Date);
        RENTEICON : BuyRente(Date, Assets);
        OFFICEICON : BuyOffice(Assets, Date, FinMenu, PersMenu);
        LEASEICON : BuyLease(Assets, FinMenu, Date);
        TEXTILEICON : BuyTextiles(Assets, Date);
    end; (* Case *)
end; (* Purchase *)

{$S Seg10}

(*****************************************************************************************)
procedure LoseLast(var IndivRentes:RenteHandle);

(* CALLED BY: SellKRente                                                                 *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure disposes of the last handle in the rente list.                 *)

var  MARKER, DISPMARK : RenteHandle;

begin (* LoseLast *)
    if IndivRentes^^.Next = nil then begin
        DisposHandle(Pointer(ord(IndivRentes)));
        IndivRentes := nil;
    end else begin
        MARKER := IndivRentes;
        while MARKER^^.Next^^.Next <> nil do begin
            MARKER := MARKER^^.Next;
        end; (* While *)
        DISPMARK := MARKER^^.Next;
        MARKER^^.Next := nil;
        DisposHandle(Pointer(ord(DISPMARK)));
    end; (* If *)
end; (* LoseLast *)

(*****************************************************************************************)
```

```
procedure SellKRente(var Assets:AssetsType);

(* CALLED BY: SellRente                                                    *)
(* CALLS TO: NumSpecs, ConvertNum, LoseLast                                *)
(* GLOBALS: Assets                                                         *)
(* ACTION: This procedure displays a dialog for the sale of the King's rentes. *)

var  RENTEDIALOG : DialogPtr;
     NUM, TEMP, PRICE : longint;
     LEN, ITEM : integer;
     NUMSTR, COSTSTR, AMT, NUMSOLDSTR, CASHVALSTR : Str255;
     CONVOK : boolean;

begin (* SellKRente *)
    RENTEDIALOG := GetNewDialog(261, nil, Pointer(-1));
    with Assets.Rente do begin
        NUM := NumKRente(IndivRentes);
        NumSpecs(NUM, LEN, NUMSTR);
        NumSpecs(CostDenier, LEN, COSTSTR);
        ParamText(NUMSTR, COSTSTR, '', '');
        repeat
            CONVOK := true;
            SelIText (RENTEDIALOG, RENTESELLITEM, 0, 200);
            DlogManager(ITEM);
            if ITEM = OK then begin
                GetDText(RENTEDIALOG, RENTESELLITEM, AMT);
                ConvertNum(AMT, TEMP, CONVOK);
                if not CONVOK then begin
                    LEN := StopAlert(256, nil);
                end else begin
                    if TEMP > NUM then begin
                        LEN := CautionAlert(260, nil);
                        CONVOK := false;
                    end; (* If *)
                end; (* If *)
            end; (* If *)
        until CONVOK;
        if (ITEM = OK) and (TEMP > 0) then begin
            for LEN := 1 to TEMP do begin
                LoseLast(IndivRentes);
            end; (* For *)
            PRICE := ((Return * 1000) div (1000 div CostDenier)) * TEMP;
            Assets.Cash := Assets.Cash + PRICE;
            NumSpecs(TEMP, LEN, NUMSOLDSTR);
            NumSpecs(CostDenier, LEN, COSTSTR);
            NumSpecs(PRICE, LEN, CASHVALSTR);
            ParamText(NUMSOLDSTR, COSTSTR, CASHVALSTR, '');
            LEN := NoteAlert(334, nil);
        end; (* If *)
    end; (* With *)
    DisposDialog(RENTEDIALOG);
end; (* SellKRente *)
```

```
(*****************************************************************************************)
procedure SellRente(var Assets:AssetsType; Date:DateType);

(* CALLED BY: Sell                                                                       *)
(* CALLS TO: SellKRente, SellYRente                                                      *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure queries the player for the sale of rentes:  King's or Yours? and calls  *)
(*  corresponding  procedures.                                                           *)

var  RENTEDIALOG : DialogPtr;
     ITEM : integer;

begin (* SellRente *)
    RENTEDIALOG := GetNewDialog(264, nil, Pointer(-1));
    DlogManager(ITEM);
    DisposDialog(RENTEDIALOG);
    if ITEM = OK then begin
        if Assets.Rente.IndivRentes = nil then begin
            ITEM := CautionAlert(265, nil);
        end else begin
            SellKRente(Assets);
        end; (* If *)
    end else begin
        SellYRente(Assets, Date);
    end; (* If *)
end; (* SellRente *)

(*****************************************************************************************)
procedure SellAdvance(var Pos: integer; Direction, NumOffices:integer; var ShownOffices:OffArray;
                  Date:DateType; OfficeList:OfficeHandle);

(* CALLED BY: SellOffice                                                                 *)
(* CALLS TO: NumSpecs                                                                    *)
(* GLOBALS: Date                                                                         *)
(* ACTION: This procedure advances the list of offices owned by a player displayed in the sell office dialog.*)

var  VALUE, ALTEREDVAL, ThisSALARY : longint;
     TITLE, VALSTR, SALARYSTR : Str255;
     I, J, LEN : integer;
     STR : array [1..NUMOFFPERSCREEN] of Str255;
    MARKER : OfficeHandle;

begin (* SellAdvance *)
    if Direction = PREVBTNITEM then begin
        Pos := Pos - (2 * NUMOFFPERSCREEN);
        if Pos < 0 then begin
            Pos := NumOffices + Pos;
        end; (* If *)
        if Pos < 0 then begin
            Pos := NumOffices + Pos;
        end; (* If *)
```

```
    end; (* If *)
    for I := 1 to NUMOFFPERSCREEN do begin
        MARKER := OfficeList;
        for J := 1 to Pos do begin
            MARKER := MARKER^^.Next;
        end; (* For *)
        Pos := Pos + 1;
        if Pos >= NumOffices then begin
            Pos := 0;
        end; (* If *)
        TITLE := MARKER^^.Title;
        VALUE := MARKER^^.Value;
        ThisSALARY := MARKER^^.Salary;
        if (Date.Year >= 1642) and (Date.Year <= 1652) then begin
            ALTEREDVAL := (VALUE * GLUTPERCENT) div 100;
            NumSpecs(ALTEREDVAL, LEN, VALSTR);
        end else begin
            NumSpecs(VALUE, LEN, VALSTR);
        end; (* If *)
        NumSpecs(ThisSALARY, LEN, SALARYSTR);
        STR[I] := Concat('£', VALSTR,' -- ',TITLE, ' -- Salary £', SALARYSTR);
        ShownOffices[I].Title := TITLE;
        ShownOffices[I].Value := VALUE;
    end; (* For *)
    ParamText(STR[1], STR[2], STR[3], STR[4]);
end; (* SellAdvance *)

(*****************************************************************************************)
procedure HideCtl(OfficeDialog:DialogPtr; Item:integer);

(* CALLED BY: FewOffices                                                                 *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure hides radio buttons in the sell office dialog.                 *)

var  DUMMYTYPE : integer;
    ITEMHDL : Handle;
    DUMMYRECT : Rect;

begin (* HideCtl *)
    GetDItem(OfficeDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    HideControl(Pointer(ITEMHDL));
end; (* HideCtl *)

(*****************************************************************************************)
procedure FewOffices(OfficeDialog:DialogPtr; Number:longint; OfficeList:OfficeHandle;
                var ShownOffices:OffArray);

(* CALLED BY: SellOffice                                                                 *)
(* CALLS TO: HideCtl, NumSpecs, LightBtn                                                 *)
(* GLOBALS: Date                                                                         *)
(* ACTION: This procedure lists the appropriate number of radio buttons and offices in the sell office       *)
```

```
(* dialog.                                                                    *)

var  I, LEN : integer;
    MARKER : OfficeHandle;
    TITLE, VALSTR, SALARYSTR : Str255;
    ThisSALARY, VALUE, ALTEREDVAL : longint;
    STR : array[1..NUMOFFPERSCREEN] of Str255;

begin (* FewOffices *)
    for I := 1 to NUMOFFPERSCREEN do begin
        STR[I] := '';
    end; (* For *)
    case Number of
        1 : begin
                HideCtl(OfficeDialog, RAD2);
                HideCtl(OfficeDIalog, RAD3);
                HideCtl(OfficeDialog, RAD4);
            end; (* One *)

        2 : begin
                HideCtl(OfficeDialog, RAD3);
                HideCtl(OfficeDialog, RAD4);
            end; (* Two *)

        3 : HideCtl(OfficeDialog, RAD4);
    end; (* Case *)
    MARKER := OfficeList;
    for I := 1 to Number do begin
        TITLE := MARKER^^.Title;
        VALUE := MARKER^^.Value;
        ThisSALARY := MARKER^^.Salary;
        if (Date.Year >= 1642) and (Date.Year <= 1652) then begin
            ALTEREDVAL := (VALUE * GLUTPERCENT) div 100;
            NumSpecs(ALTEREDVAL, LEN, VALSTR);
        end else begin
            NumSpecs(VALUE, LEN, VALSTR);
        end; (* If *)
        NumSpecs(ThisSALARY, LEN, SALARYSTR);
        STR[I] := Concat('£', VALSTR,' -- ',TITLE, ' -- Salary £', SALARYSTR);
        ShownOffices[I].Title := TITLE;
        ShownOffices[I].Value := VALUE;
        MARKER := MARKER^^.Next;
    end; (* For *)
    ParamText(STR[1], STR[2], STR[3], STR[4]);
    LightBtn(OfficeDialog, NEXTBTNITEM, BTNINACTIVE);
    LightBtn(OfficeDialog, PREVBTNITEM, BTNINACTIVE);
end; (* FewOffices *)

(*****************************************************************************************)
procedure LoseOffice(var Offices:OfficeType; NewOffice:DlogOffRec; Date:DateType);

(* CALLED BY: SellOffice                                                      *)
```

```
(* CALLS TO: none                                                                *)
(* GLOBALS: Date                                                                 *)
(* ACTION: This procedure removes an office from the office list by disposing of its handle.  *)

var  FOUND : boolean;
   MARKER, DISPMARK : OfficeHandle;

begin (* LoseOffice *)
   with Offices do begin
      FOUND := false;
      MARKER := OfficeList;
      while not FOUND do begin
         if MARKER^^.Title = NewOffice.Title then begin
            FOUND := true;
          end else begin
            MARKER := MARKER^^.Next;
          end; (* If *)
       end; (* While *)
       Salary := Salary - MARKER^^.Salary;
       TotPurchase := TotPurchase - MARKER^^.AmtPaid;
       if OfficeList^^.Title = NewOffice.Title then begin
            DISPMARK := OfficeList;
          OfficeList := OfficeList^^.Next;
          DisposHandle(Pointer(DISPMARK));
       end else begin
          FOUND := false;
          MARKER := OfficeList;
          while not FOUND do begin
            if MARKER^^.Next^^.Title = NewOffice.Title then begin
                FOUND := true;
                DISPMARK := MARKER^^.Next;
              MARKER^^.Next := MARKER^^.Next^^.Next;
                DisposHandle(Pointer(DISPMARK));
            end else begin
                MARKER := MARKER^^.Next;
            end; (* If *)
          end; (* While *)
       end; (* If *)
   end; (* With *)
end; (* LoseOffice *)

(*****************************************************************************)
procedure SellOffice(var Assets:AssetsType; FinMenu:MenuHandle; Date:DateType);

(* CALLED BY: Sell                                                           *)
(* CALLS TO: FewOffices, SellAdvance, LightBtn, DoCheck, GetOffBought, NumSpecs,  *)
(* CalcCostOfLiving, LoseOffice                                              *)
(* GLOBALS: Assets, Date                                                     *)
(* ACTION: This procedure sets up a dialog for selling offices held by the player.  If the office selected is *)
(* Secretary of the King, the sale is not allowed.  FewOffices or SellAdvance are called, depending on  *)
(* whether the list of held offices is longer than one screenful.            *)
```

```
var  OFFICEDIALOG : DialogPtr;
     POS, ITEM, LEN : integer;
     SHOWNOFFICES : OffArray;
     SELLOK : boolean;
     NEWOFFICE : DlogOffRec;
     COST : longint;
     SALARYSTR, COSTofLSTR : STR255;

begin (* SellOffice *)
     OFFICEDIALOG := GetNewDialog(268, nil, Pointer(-1));
     with Assets.Office do begin
          if Number <= NUMOFFPERSCREEN then begin
               FewOffices(OFFICEDIALOG, Number, OfficeList, SHOWNOFFICES);
          end else begin
               POS := 0;
               SellAdvance(POS, NEXTBTNITEM, Number, SHOWNOFFICES, Date, OfficeList);
          end; (* If *)
          LightBtn(OFFICEDIALOG, SELLBTNITEM, BTNINACTIVE);
          repeat
               SELLOK := true;
               repeat
                    SystemTask;
                    ModalDialog(nil, ITEM);
                    if ITEM in [RAD1, RAD2, RAD3, RAD4] then begin
                         DoCheck(OFFICEDIALOG, ITEM, Number);
                    end; (* If *)
                    if ITEM in [NEXTBTNITEM, PREVBTNITEM] then begin
                         SellAdvance(POS, ITEM, Number, SHOWNOFFICES, Date, OfficeList);
                         DrawDialog(OFFICEDIALOG);
                    end; (* If *)
               until ITEM in [SELLBTNITEM, Cancel];
               if ITEM = SELLBTNITEM then begin
                    GetOffBought(OFFICEDIALOG, SHOWNOFFICES, NEWOFFICE);
                    if NEWOFFICE.Value = NONE then begin
                         SELLOK := false;
                    end else begin
                         if NEWOFFICE.Title = SECYKING then begin
                              LEN := StopAlert(301, nil);
                              SELLOK := false;
                         end else begin
                              COST := NEWOFFICE.Value;
                              if (Date.Year >= 1642) and (Date.Year <= 1652) then begin
                                   COST := COST - ((COST * LEVYTAX) div 100);
                              end; (* If *)
                              Assets.Cash := Assets.Cash + COST;
                              LoseOffice(Assets.Office, NEWOFFICE, Date);
                              ParamText(NEWOFFICE.Title, '', '', '');
                              LEN := Notealert(322, nil);
                              CalcCostofLiving (Assets);
                              NumSpecs(Assets.Office.Salary, LEN, SALARYSTR);
                              NumSpecs(Assets.CostOfLiving, LEN, COSTofLSTR);
                              ParamText(SALARYSTR, COSTofLSTR, '', '');
```

```
                    LEN := Notealert(320, nil);
                    if OfficeList = nil then begin
                        DisableItem(FinMenu, SELLITEM);
                    end; (* If *)

                    Number := Number - 1;
                end; (* If *)
            end; (* If *)
        end; (* If *)
    until SELLOK;
  end; (* With *)
  DisposDialog(OFFICEDIALOG);
end; (* SellOffice *)

(*****************************************************************************************)
procedure SellLand(var Assets:AssetsType; FinMenu:MenuHandle);

(* CALLED BY: Sell                                                                       *)
(* CALLS TO: NumSpecs, SellText, ConvertNum                                              *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure displays a dialog for the sale of land and grain held by the player.  Prices  *)
(* are posted and grain amounts are deducted, cash is increased for each sale. Incorrect values and inherited  *)
(* land (which can't be sold) are doublechecked.                                         *)

var  LANDDIALOG : DialogPtr;
    LEN, ITEM : integer;
    GRAINSTR, LANDSTR, AMT, PRICESTR,
    LANDPRICESTR, LANDSOLDSTR, LANDPROFITSTR,
    GRAINSOLDSTR, GRAINPROFITSTR, NEWAMTSTR : Str255;
    CONVOK : boolean;
    LANDTEMP, TEMP, VALTEMP : longint;

begin (* SellLand *)
   LANDDIALOG := GetNewDialog(262, nil, Pointer(-1));
    with Assets do begin
     VALTEMP := LANDVAL;
    if (Assets.Protector.Name = Conde) or (Assets.Protector.Name = GrandDauphin) then begin
       if (Date.Year >= 1653) and (Assets.Protector.Name = Conde) then begin
           VALTEMP := (LANDVAL * 3) div 4;
       end else begin
           VALTEMP := LANDVAL div 2;
       end; (* If *)
     end; (* If *)

       NumSpecs(Grain, LEN, GRAINSTR);
       NumSpecs(Land.Bought + Land.Inherited, LEN, LANDSTR);
       NumSpecs(Land.Price, LEN, PRICESTR);
       NumSpecs(VALTEMP, LEN, LANDPRICESTR);
       ParamText(LANDSTR, GRAINSTR, PRICESTR, LANDPRICESTR);
       repeat
           CONVOK := true;
           SellText (LANDDIALOG, LANDSELLITEM, 0, 200);
```

```
    DlogManager(ITEM);
    if ITEM = OK then begin
       GetDText(LANDDIALOG, LANDSELLITEM, AMT);
        ConvertNum(AMT, LANDTEMP, CONVOK);
        if not CONVOK then begin
            LEN := StopAlert(256, nil);
        end else begin
           if LANDTEMP > Land.Bought + Land.Inherited then begin
               LEN := CautionAlert(261, nil);
               CONVOK := false;
           end else begin
               if LANDTEMP > Land.Bought then begin
                   LEN := CautionAlert(262,nil);
                   CONVOK := false;
               end; (* If *)
           end; (* If *)
        end; (* If *)
        if CONVOK then begin
           GetDText(LANDDIALOG, GRAINSELLITEM, AMT);
            ConvertNum(AMT, TEMP, CONVOK);
            if not CONVOK then begin
                LEN := StopAlert(256, nil);
            end else begin
                if TEMP > Grain then begin
                    LEN := StopAlert(263, nil);
                    CONVOK := false;
                end; (* If *)
            end; (* If *)
        end; (* If *)
    end; (* If *)
 until CONVOK;
if (ITEM = OK) and (LANDTEMP > 0) then begin
    Land.Bought := Land.Bought - LANDTEMP;
    Assets.Cash := Assets.Cash + (LANDTEMP * VALTEMP);
    NumSpecs(Land.Bought, LEN, NEWAMTSTR);
    NumSpecs(LANDTEMP * VALTEMP, LEN, LANDPROFITSTR);
    NumSpecs(LANDTEMP, LEN, LANDSOLDSTR);
    ParamText(NEWAMTSTR, LANDPROFITSTR, LANDSOLDSTR,");
    LEN := NoteAlert(330, nil);
end; (* If *)
if (ITEM = OK) and (TEMP > 0) then begin
    Grain := Grain - TEMP;
    Cash := Cash + (TEMP * Land.Price);
    NumSpecs(Grain, LEN, NEWAMTSTR);
    NumSpecs(TEMP * Land.Price, LEN, GRAINPROFITSTR);
    NumSpecs(TEMP, LEN, GRAINSOLDSTR);
    ParamText(NEWAMTSTR, GRAINPROFITSTR, GRAINSOLDSTR,");
    LEN := NoteAlert(331, nil);
    if TEMP >= ((Grain + TEMP) div 10) then Assets.SoldGrain := true;
end; (* If *)
if (Land.Bought = 0) and (Grain = 0) then begin
    DisableItem(FinMenu, SELLITEM);
```

```
        end; (* If *)
    end; (* With *)
    DisposDialog(LANDDIALOG);
end; (* SellLand *)

(*****************************************************************************************)
procedure Sell(var Assets:AssetsType; Choice:integer; Date:DateType; FinMenu:MenuHandle);

(* CALLED BY: DoCommand, Main                                                            *)
(* CALLS TO: SellLand, SellRente, SellOffice                                             *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure checks the investment icons for the appropriate transaction and calls the *)
(*  procedure.                                                                           *)
begin (* Sell *)
    case Choice of
        LANDICON : SellLand(Assets, FinMenu);
        RENTEICON : SellRente(Assets, Date);
        OFFICEICON : SellOffice(Assets, FinMenu, Date);
        LEASEICON : ;
        TEXTILEICON : ;
    end; (* Case *)
end; (* Sell *)

{$S Seg2}

(*****************************************************************************************)
procedure SetUpWindow(var TextWindow:WindowPtr);

(* CALLED BY: SetUp                                                                      *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure creates a window for text display, with a grow icon.           *)

begin (* SetUpWindow *)
    TextWindow := GetNewWindow(259, nil, Pointer(-1));
    DrawGrowIcon(TextWindow);
    SetPort(TextWindow);
    PLSetWrPort(TextWindow);
end; (* SetUpWindow *)

(*****************************************************************************************)
procedure SetUpControls(var ScrlBar:ControlHandle; TextWindow:WindowPtr);

(* CALLED BY: SetUp                                                                      *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure sets up the scroll bar control on a textwindow.               *)

begin (* SetUpControls *)
    ScrlBar := GetNewControl(256, TextWindow);
    ShowControl(ScrlBar);
end; (* SetUpControls *)
```

```
(*****************************************************************************)
procedure SetUp(var TextWindow:WindowPtr; var ScrlBar:ControlHandle; var TopLine:integer; var hTE:TEHandle);

(* CALLED BY: ReadText                                                        *)
(* CALLS TO: SetUpWindow, SetUpControls, SetUpTextEdit                        *)
(* GLOBALS: TopLine, hTE                                                      *)
(* ACTION: This procedure initiates display of a text window with grow icon, scroll bars, and text display. *)

begin (* SetUp *)
    TopLine := 0;
   SetUpWindow(TextWindow);
   SetUpControls(ScrlBar, TextWindow);
   SetUpTextEdit(hTE, TextWindow);
end; (* SetUp *)

(*****************************************************************************)
procedure ScrollBits(ScrlBar:ControlHandle; var TopLine:integer; var hTE:TEHandle);

(* CALLED BY: Increase, Decrease, PageScroll, DoScroll                        *)
(* CALLS TO: none                                                             *)
(* GLOBALS: TopLine, hTE                                                      *)
(* ACTION: This procedure readjusts the top line of display in a window for scrolling. *)

var  OLDVERT,
   HEIGHT  : integer;

begin (* ScrollBits *)
   OLDVERT := TopLine;
   TopLine := GetCtlValue(ScrlBar);
   HEIGHT := hTE^^.LineHeight;
   TEScroll(0, (OLDVERT - TopLine) * HEIGHT, hTE);
end; (* ScrollBits *)

(*****************************************************************************)
procedure Increase(theControl:ControlHandle; partCode:integer);

(* CALLED BY: DoScroll                                                        *)
(* CALLS TO: ScrollBits                                                       *)
(* GLOBALS: TopLine, hTE                                                      *)
(* ACTION: This procedure reads the up arrow for amount scrolled, and moves the top line of display *)
(* to match.                                                                  *)

var  VAL : integer;

begin (* Increase *)
   if partCode = inUpButton then begin
      VAL := GetCtlValue(theControl);
      if VAL <> 0 then begin
         SetCtlValue(theControl, VAL - 1);
         ScrollBits(theControl, TopLine, hTE);
      end; (* If *)
```

```
    end; (* If *)
end; (* Increase *)

(*****************************************************************************************)
procedure Decrease(theControl:ControlHandle; partCode:integer);

(* CALLED BY: DoScroll                                                                   *)
(* CALLS TO: ScrollBits                                                                  *)
(* GLOBALS: TopLine, MaxScroll, hTE                                                      *)
(* ACTION: This procedure reads the down arrow for amount scrolled, and moves the top line of display  *)
(* to match.                                                                             *)

var  VAL : integer;

begin (* Decrease *)
    if partCode = inDownButton then begin
        VAL := GetCtlValue(theControl);
        if VAL < MaxScroll then begin
            SetCtlValue(theControl, VAL + 1);
            ScrollBits(theControl, TopLine, hTE);
        end; (* If *)
    end; (* If *)
end; (* Decrease *)

(*****************************************************************************************)
procedure PageScroll(Which:integer; var ScrlBar:ControlHandle; var hTE:TEHandle);

(* CALLED BY: DoScroll                                                                   *)
(* CALLS TO: ScrollBits                                                                  *)
(* GLOBALS: TopLine, hTE                                                                 *)
(* ACTION: This procedure scrolls text on a page using the values read in from the controls in the  *)
(* scroll bar.                                                                           *)

var  PT : Point;
    AMOUNT, HEIGHT : integer;
    TEMPRECT : Rect;

begin (* PageScroll *)
    if Which = inPageUp then begin
        AMOUNT := -1;
    end else begin
        AMOUNT := 1;
    end; (* If *)
    repeat
        GetMouse(PT);
        if TestControl(ScrlBar, PT) = Which then begin
            TEMPRECT := hTE^^.ViewRect;
            HEIGHT := hTE^^.LineHeight;
             with TEMPRECT do begin
                SetCtlValue(ScrlBar, GetCtlValue(ScrlBar) + AMOUNT * (Bottom - Top) div HEIGHT);
             end; (* With *)
             ScrollBits(ScrlBar, TopLine, hTE);
```

```
        end; (* If *)
    until not StillDown;
end; (* PageScroll *)

(*****************************************************************************************)
procedure DoScroll(var hTE:TEHandle;var ScrlBar:ControlHandle; AnEvent:EventRecord; TextWindow:WindowPtr);

(* CALLED BY: ReadText                                                                   *)
(* CALLS TO: ScrollBits                                                                  *)
(* GLOBALS: TopLine, hTE                                                                 *)
(* ACTION: This procedure scrolls text in a display window by reading the scroll control bar.  *)

var  MOUSELOC : Point;
    CTLPART : integer;
    OLDVALUE, VALUE : integer;
    THECONTROL : ControlHandle;

begin (* DoScroll *)
    MOUSELOC := AnEvent.where;
    GlobalToLocal(MOUSELOC);
    CTLPART := FindControl(MOUSELOC, TextWindow, THECONTROL);
    if THECONTROL = ScrlBar then begin
        OLDVALUE := GetCtlValue(ScrlBar);
        case CTLPART of
            inUpButton : VALUE := TrackControl(ScrlBar, MOUSELOC, @Increase);
            inDownButton : VALUE := TrackControl(ScrlBar, MOUSELOC, @Decrease);
            inPageUp : PageScroll(CTLPART, ScrlBar, hTE);
            inPageDown : PageScroll(CTLPART, ScrlBar, hTE);
            inThumb : begin
                        VALUE := TrackControl(ScrlBar, MOUSELOC, nil);
                        ScrollBits(ScrlBar, TopLine, hTE);
                    end; (* InThumb *)
        end; (* Case *)
        if VALUE = 0 then begin
            SetCtlValue(ScrlBar, OLDVALUE);
        end; (* If *)
    end; (* If *)
end; (* DoScroll *)

(*****************************************************************************************)
procedure SetScrollMax(var hTE:TEHandle; ScrlBar:ControlHandle; var TopLine, MaxScroll:integer);

(* CALLED BY: ReadText                                                                   *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: TopLine, MaxScroll, hTE                                                      *)
(* ACTION: This procedure sets initial and maximum values for scrolling in a display window.  *)

begin (* SetScrollMax *)
    MaxScroll := hTE^^.nLines - (hTE^^.viewRect.bottom - hTE^^.viewRect.top + 1) DIV (hTE^^.lineHeight);
    if MaxScroll < 0 then begin
        MaxScroll := 0;
    end; (* If *)
```

```
    SetCtlMax (ScrlBar, MaxScroll);
    TopLine := 0;
    SetCtlValue (ScrlBar, TopLine);
end; (* SetScrollMax *)

{$S}

(*******************************************************************************************)
procedure Treasury(var Assets:AssetsType; Date:DateType);

(* CALLED BY: Main                                                                         *)
(* CALLS TO: ConvertNum, DisplayAssets                                                     *)
(* GLOBALS: Assets, Date                                                                   *)
(* ACTION: This procedure is a debugging tool. By pressing option-command-shift-capslock-s the user    *)
(* gets a dialog in which to enter any amount of cash desired. This is transfered to the player's cash value.  *)

var  TreasuryDIALOG : DialogPtr;
    ITEM : integer;
    STR : Str255;
    NUM : longint;
    CONVOK : boolean;

begin (* Treasury *)
    TreasuryDIALOG := GetNewDialog(269, nil, Pointer(-1));
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
    until ITEM = OK;
    GetDText(TreasuryDIALOG, 2, STR);
    ConvertNum(STR, NUM, CONVOK);
    Assets.Cash := NUM;
    Assets.Debt := 0;
    DisposDialog(TreasuryDIALOG);
    DisplayAssets(Assets, Date);
end; (* Treasury *)

{$S Seg2}

(*******************************************************************************************)
procedure ReadText(var TopLine, MaxScroll:integer; var hTE:TEHandle; WatchHdl:CursHandle; VRefNum:integer);

(* CALLED BY: DoCommand, Main                                                              *)
(* CALLS TO: GetText, SetUp, SetScrollMax, DoScroll                                        *)
(* GLOBALS: Code, TopLine, MaxScroll, VRefNum, hTE, WatchHdl                               *)
(* ACTION: This procedure sets the cursor and event manager to wait for a mouse down event in a    *)
(* window before closing it. This allows the player to read a window and put it away when finished.  *)

var  STOP : boolean;
    TOP, TEMP : boolean;
    TEXTWINDOW, WHICHWINDOW : WindowPtr;
    ANEVENT : EventRecord;
    SCRLBAR : ControlHandle;
```

```
begin (* ReadText *)
    STOP := false;
    Hlock(Pointer(WatchHdl));
    SetCursor(WatchHdl^^);
    Hunlock(Pointer(WatchHdl));
    SetUp(TEXTWINDOW, SCRLBAR, Topline, hTE);
    GetText(INSTRFILE, hTE, VRefNum);
    SetScrollMax(hTE, SCRLBAR, TopLine, MaxScroll);
    TEUpdate(TextWindow^.portRect, hTE);
    SetCursor(Arrow);
    repeat
        SystemTask;
        TEMP := GetNextEvent(everyEvent, ANEVENT);
        case ANEVENT.what of
            MouseDown : begin
                CODE := FindWindow(ANEVENT.where, WHICHWINDOW);
                if WHICHWINDOW <> TEXTWINDOW then begin
                    SysBeep(BEEPDURATION);
                end else begin
                    case Code of
                        inContent : DoScroll(hTE, SCRLBAR, ANEVENT, TEXTWINDOW);
                        inGoAway : if TrackGoAway(WHICHWINDOW, ANEVENT.where) then begin
                                    STOP := true;
                                    TEDispose(hTE);
                                    hTE := nil;
                                    KillControls(TEXTWINDOW);
                                    DisposeWindow(TEXTWINDOW);
                                    SetPort(thePort);
                                    PLSetWrPort(thePort);
                                end; (* If *)
                        inSysWindow : SystemClick(ANEVENT, WHICHWINDOW);
                    end; (* Case *)
                end; (* If *)
            end; (* MouseDown *)
        end; (* Case *)
    until STOP;
end; (* ReadText *)

{$S Seg3}

(*****************************************************************************************)
procedure MarrCheck(MarrDialog:DialogPtr; Item:integer);

(* CALLED BY: DoMarriage                                                                 *)
(* CALLS TO: LightBtn,                                                                   *)
(* GLOBALS: none                                                                         *)
(* ACTION: This procedure examines the marriage dialog for mouse down events, that is, choices.  *)

var  DUMMYTYPE : integer;
    ITEMHDL, RADHDL : Handle;
    DUMMYRECT : Rect;
```

```
    VAL : integer;

begin (* MarrCheck *)
    GetDItem(MarrDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    VAL := GetCtlValue(Pointer(ord(ITEMHDL)));
    if VAL = CHECKED then begin
        VAL := NOTCHECKED;
       LightBtn(MarrDialog, COURTITEM, BTNINACTIVE);
       LightBtn(MarrDialog, INFOITEM, BTNINACTIVE);
    end else begin
        VAL := CHECKED;
       LightBtn(MarrDialog, COURTITEM, BTNACTIVE);
       LightBtn(MarrDIalog, INFOITEM, BTNACTIVE);
        GetDItem(MarrDialog, MARRAD1, DUMMYTYPE, RADHDL, DUMMYRECT);
        SetCtlValue(Pointer(RADHDL), NOTCHECKED);
        GetDItem(MarrDialog, MARRAD2, DUMMYTYPE, RADHDL, DUMMYRECT);
        SetCtlValue(Pointer(RADHDL), NOTCHECKED);
        GetDItem(MarrDialog, MARRAD3, DUMMYTYPE, RADHDL, DUMMYRECT);
        SetCtlValue(Pointer(RADHDL), NOTCHECKED);
        GetDItem(MarrDialog, MARRAD4, DUMMYTYPE, RADHDL, DUMMYRECT);
        SetCtlValue(Pointer(RADHDL), NOTCHECKED);
    end; (* If *)
    SetCtlValue(Pointer(ITEMHDL), VAL);
end; (* MarrCheck *)

(*****************************************************************************************)
procedure ExaMarrItem(MarrDialog:DialogPtr; Item, ItemNum:integer; var ThisBride:BrideRec;
              ThisYear:BrideArray; var Found:boolean);

(* CALLED BY: GetMarrChecked                                                              *)
(* CALLS TO: none                                                                         *)
(* GLOBALS: none                                                                          *)
(* ACTION: This procedure checks to see if an item is chosen. It returns the bride, year, and sets *)
(* choice found true.                                                                     *)

var  DUMMYTYPE, VAL : integer;
    ITEMHDL : Handle;
     DUMMYRECT : Rect;

begin (* ExaMarrItem *)
    GetDItem(MarrDialog, Item, DUMMYTYPE, ITEMHDL, DUMMYRECT);
    VAL := GetCtlValue(Pointer(ITEMHDL));
    if VAL = CHECKED then begin
       ThisBride := ThisYear[ItemNum];
        Found := true;
    end; (* If *)
end; (* ExaMarrItem *)

(*****************************************************************************************)
procedure GetMarrChecked(MarrDialog:DialogPtr; ThisYear:BrideArray; var ThisBride:BrideRec);

(* CALLED BY: DoMarriage                                                                  *)
```

```
(* CALLS TO: ExaMarrItem                                                              *)
(* GLOBALS: none                                                                      *)
(* ACTION: This procedure polls the radio buttons in the marriage dialog for choices, and returns a  *)
(* bride if found.                                                                    *)

var  FOUND : boolean;

begin (* GetMarrChecked *)
   FOUND := false;
   ExaMarrItem(MarrDialog, MARRAD1, 1, ThisBride, ThisYear, FOUND);
   ExaMarrItem(MarrDialog, MARRAD2, 2, ThisBride, ThisYear, FOUND);
   ExaMarrItem(MarrDialog, MARRAD3, 3, ThisBride, ThisYear, FOUND);
   ExaMarrItem(MarrDialog, MARRAD4, 4, ThisBride, ThisYear, FOUND);
   if not FOUND then begin
      ThisBride.Age := NONE;
   end; (* If *)
end; (* GetMarrChecked *)

(*****************************************************************************************)
procedure DoMarriage(var Assets:AssetsType; Date:DateType; PersMenu:MenuHandle);

(* CALLED BY: DoCommand, Main                                                         *)
(* CALLS TO: LightBtn, MarrCheck, NumSpecs, GetMarrChecked, Demographics              *)
(* GLOBALS: Assets, Date                                                              *)
(* ACTION: This dialog displays a marriage dialog and responds to choices made.       *)

var  MARRDIALOG : DialogPtr;
   ITEM, LEN : integer;
   THISBRIDE : BrideRec;
   AMT, AGESTR : Str255;
   MARROK : boolean;

begin (* DoMarriage *)
   with Assets.Marriage do begin
      if Available.IsAvail then begin
         ParamText(ThisYear[1].Name, ThisYear[2].Name, ThisYear[3].Name, ThisYear[4].Name);
         MARRDIALOG := GetNewDialog(271, nil, Pointer(-1));
         LightBtn(MARRDIALOG, COURTITEM, BTNINACTIVE);
         LightBtn(MARRDIALOG, INFOITEM, BTNINACTIVE);
         repeat
            MARROK := true;
            repeat
               SystemTask;
               ModalDialog(nil, ITEM);
               if ITEM in [MARRAD1, MARRAD2, MARRAD3, MARRAD4] then begin
                  MarrCheck(MARRDIALOG, ITEM);
               end; (* If *)
               if ITEM = INFOITEM then begin
                  GetMarrChecked(MARRDIALOG, ThisYear, THISBRIDE);
                  NumSpecs(THISBRIDE.Dowry, LEN, AMT);
                  NumSpecs(THISBRIDE.Age, LEN, AGESTR);
                  ParamText(THISBRIDE.Name, AMT, THISBRIDE.Father, AGESTR);
```

```
                LEN := Alert(272, nil);
                ParamText(ThisYear[1].Name, ThisYear[2].Name, ThisYear[3].Name, ThisYear[4].Name);
              end; (* If *)
            until ITEM in [COURTITEM, Cancel];
            if ITEM = COURTITEM then begin
              GetMarrChecked(MARRDIALOG, ThisYear, THISBRIDE);
              if THISBRIDE.Age <> NONE then begin
                if (THISBRIDE.Group div 10 <= Assets.Prestige div 10) and
                  (Assets.TotalVal >= 2 * THISBRIDE.Dowry) then begin
                    Bride := THISBRIDE;
                    Married := true;
                    Assets.Cash := Assets.Cash + Bride.Dowry;
                    DisableItem(PersMenu, MARRITEM);
                    if (Assets.Generation = SECONDGEN) then begin
                      EnableItem(PersMenu, PLANITEM);
                    end; (* If *)
                    Available.Year := Date.Year;
                    ParamText(Bride.Name, '', '', '');
                    LEN := Alert(273, nil);
                    if THISBRIDE.Group div 10 < Assets.Prestige div 10 then begin
                      MarrBelow := true;
                      LEN := StopAlert(310, nil);
                    end; (* If *)
                    DemoGraphics(Assets, Date, PersMenu);
                    if not Date.Fall then begin
                      Assets.Marriage.Bride.Age := Assets.Marriage.Bride.Age - 1;
                    end; (* If *)
                end else begin
                  Available.IsAvail := false;
                  Available.Year := Date.Year;
                  Failures := Failures + 1;
                  ParamText(THISBRIDE.Name, '', '', '');
                  LEN := StopAlert(275, nil);
                end; (* If *)
              end else begin
                MARROK := false;
              end; (* If *)
            end; (* If *)
          until MARROK;
          DisposDialog(MARRDIALOG);
        end else begin
          LEN := StopAlert(274, nil);
        end; (* If *)
      end; (* With *)
    end; (* DoMarriage *)

    {$S Seg14}

    (*****************************************************************************)
    procedure PlanFamily(var Assets:AssetsType; Date:DateType; PersMenu:MenuHandle);

    (* CALLED BY: DoCommand, Main                                                *)
```

```
(* CALLS TO: none                                                                    *)
(* GLOBALS: Assets, Date                                                             *)
(* ACTION: This procedure determines if a wife is too old for children, and, if not, records the  *)
(* wish to have a child.                                                             *)

var  PLANDIALOG : DialogPtr;
     ITEM : integer;

begin (* PlanFamily *)
    if Assets.Marriage.Bride.Age > TOOOLDFORKIDS then begin
        ITEM := StopAlert(288, nil);
    end else begin
        PLANDIALOG := GetNewDialog(273, nil, Pointer(-1));
        DlogManager(ITEM);
        if ITEM = OK then begin
            Assets.Children.NextBirth.Year := Date.Year + 1;
            Assets.Children.NextBirth.Fall := Date.Fall;
            DisableItem(PersMenu, PLANITEM);
        end; (* If *)
        DisposDialog(PLANDIALOG);
    end; (* If *)
end; (* PlanFamily *)

(*****************************************************************************************)
procedure AboutProgram;

(* CALLED BY: DoCommand                                                              *)
(* CALLS TO: none                                                                    *)
(* GLOBALS: none                                                                     *)
(* ACTION: This procedure displays an alert describing the authors and designers of the program.  *)

var  LEN : integer;

begin (* AboutProgram *)
    LEN := Alert(286, nil);
end; (* AboutProgram *)

(*****************************************************************************************)
procedure BuyNobility(var Assets:AssetsType; PersMenu:MenuHandle);

(* CALLED BY: DoCommand                                                              *)
(* CALLS TO: NumSpecs, LightBtn,                                                     *)
(* GLOBALS: Assets                                                                   *)
(* ACTION: This procedure displays a dialog for the purchase of a letter of nobility. If purchased,  *)
(* it sets Noble to true.                                                            *)

var  NOBLEDIALOG : DialogPtr;
     LEN : integer;
     COSTSTR, CASHSTR : Str255;

begin (* BuyNobility *)
    with Assets do begin
```

```
        NumSpecs(NOBLECOST, LEN, COSTSTR);
        NumSpecs(Cash, LEN, CASHSTR);
        ParamText(COSTSTR, CASHSTR, '', '');
       NOBLEDIALOG := GetNewDialog(274, nil, Pointer(-1));
        if Cash < NOBLECOST then begin
           LightBtn(NOBLEDIALOG, OK, BTNINACTIVE);
        end; (* If *)
        DlogManager(LEN);
        if LEN = OK then begin
            Cash := Cash - NOBLECOST;
            Noble := true;
            BoughtLetter := true;
           DisableItem(PersMenu, NOBLEITEM);
        end; (* If *)
       DisposDialog(NOBLEDIALOG);
    end; (* With *)
end; (* BuyNobility *)

{$S Seg5}

(*****************************************************************************************)
procedure SaveRente(RMarker:RenteHandle; Size:longint; Refnum:integer);

(* CALLED BY: SaveSimulation                                                              *)
(* CALLS TO: none                                                                         *)
(* GLOBALS: none                                                                          *)
(* ACTION: This procedure writes the rente held by the player to disk for later recovery. *)

var  DONE : boolean;
     BSIZE : longint;
     ERR : OSErr;
     REC : RenteType;

begin (* SaveRente *)
    BSIZE := SizeOf(boolean);
    DONE := false;
    while RMarker <> nil do begin
        ERR := FSWrite(Refnum, BSIZE, @DONE);
       REC := RMarker^^;
        ERR := FSWrite(Refnum, Size, @REC);
       RMarker := RMarker^^.Next;
    end; (* While *)
    DONE := true;
    ERR := FSWrite(Refnum, BSIZE, @DONE);
end; (* SaveRente *)

(*****************************************************************************************)
procedure SaveOffice(OMarker:OfficeHandle; Size:longint; Refnum:integer);

(* CALLED BY: SaveSimulation                                                              *)
(* CALLS TO: none                                                                         *)
(* GLOBALS: none                                                                          *)
```

```
(* ACTION: This procedure writes the offices held by the player to disk for later recovery.      *)

var  DONE : boolean;
     BSIZE : longint;
     ERR : OSErr;
     REC : OfficeRec;

begin (* SaveOffice *)
    BSIZE := SizeOf(boolean);
    DONE := false;
    while OMarker <> nil do begin
        ERR := FSWrite(Refnum, BSIZE, @DONE);
       REC := OMarker^^;
        ERR := FSWrite(Refnum, Size, @REC);
       OMarker := OMarker^^.Next;
    end; (* While *)
    DONE := true;
    ERR := FSWrite(Refnum, BSIZE, @DONE);
end; (* SaveOffice *)

(*****************************************************************************************)
procedure SaveKid(KMarker:KidHandle; Size:longint; Refnum:integer);

(* CALLED BY: SaveSimulation                                                              *)
(* CALLS TO: none                                                                         *)
(* GLOBALS: none                                                                          *)
(* ACTION: This procedure saves information about children to disk for later recovery.    *)

var  DONE : boolean;
     BSIZE : longint;
     ERR : OSErr;
     REC : KidRec;

begin (* SaveKid *)
    BSIZE := SizeOf(boolean);
    DONE := false;
    while KMarker <> nil do begin
        ERR := FSWrite(Refnum, BSIZE, @DONE);
       REC := KMarker^^;
        ERR := FSWrite(Refnum, Size, @REC);
       KMarker := KMarker^^.Next;
    end; (* While *)
    DONE := true;
    ERR := FSWrite(Refnum, BSIZE, @DONE);
end; (* SaveKid *)

(*****************************************************************************************)
procedure SaveSimulation(var Assets:AssetsType; Date:DateType; WatchHdl:CursHandle);

(* CALLED BY: QuitHandler, DoCommand, Main                                                *)
(* CALLS TO: SaveRente, SaveOffice, SaveKid                                               *)
(* GLOBALS: Assets, Date, WatchHdl                                                        *)
```

```
(* ACTION: This procedure saves information about the player to disk for later recovery.                *)

var  LEN, REFNUM : integer;
     SIZE : longint;
     ERR : OSErr;

begin (* SaveSimulation *)
    LEN := CautionAlert(295, nil);
    if LEN = OK then begin
        Hlock(Pointer(WatchHdl));
       SetCursor(WatchHdl^^);
        Hunlock(Pointer(WatchHdl));
         ERR := FSDelete(SAVEFILE, 0);
         ERR := Create(SAVEFILE, 0, '????', 'SAVE');
         ERR := FSOpen(SAVEFILE, 0, REFNUM);
        SIZE := SizeOf(DateType);
         ERR := FSWrite(REFNUM, SIZE, @Date);
         SIZE := SizeOf(AssetsType);
         ERR := FSWrite(REFNUM, SIZE, @Assets);
        SaveRente(Assets.Rente.IndivRentes, SizeOf(RenteType), REFNUM);
       SaveOffice(Assets.Office.OfficeList, SizeOf(OfficeRec), REFNUM);
        SaveKid(Assets.Children.Boys, SizeOf(KidRec), REFNUM);
        SaveKid(Assets.Children.Girls, SizeOf(KidRec), REFNUM);
         ERR := FSClose(REFNUM);
    end; (* If *)
end; (* SaveSimulation *)

(*****************************************************************************************)
procedure QuitHandler(var Assets:AssetsType; Date:DateType; var Done:boolean; WatchHdl:Curshandle);

(* CALLED BY: DoCommand                                                                   *)
(* CALLS TO: SaveSimulation                                                               *)
(* GLOBALS: Done, Assets, Date, WatchHdl                                                  *)
(* ACTION: This procedure asks a player to either save their game or quit without saving. *)

var  LEN : integer;

begin (* QuitHandler *)
    LEN := CautionAlert(296, nil);
    Done := true;
    Assets.Quit := true;
    case LEN of
        OK : SaveSimulation(Assets, Date, WatchHdl);
        Cancel : begin
                    Done := false;
                    Assets.Quit := false;
                 end; (* Cancel *)
    end; (* Case *)
end; (* QuitHandler *)

(*****************************************************************************************)
procedure ReadRente(var RMarker:RenteHandle; Size:longint; Refnum:integer);
```

```
(* CALLED BY: LoadSimulation                                                    *)
(* CALLS TO: none                                                              *)
(* GLOBALS: none                                                               *)
(* ACTION: This procedure reads the rente information stored on the disk into a rente list.  *)

var  DONE : boolean;
     BSIZE : longint;
     ERR : OSErr;
     REC : RenteType;
     MARKER, TEMPHANDLE : RenteHandle;

begin (* ReadRente *)
    BSIZE := SizeOf(boolean);
    ERR := FSRead(Refnum, BSIZE, @DONE);
    if not DONE then begin
        ERR := FSRead(Refnum, Size, @REC);
        REC.Next := nil;
       RMarker := Pointer(NewHandle(Size));
       RMarker^^ := REC;
       ERR := FSRead(Refnum, BSIZE, @DONE);
       MARKER := RMarker;
       while not DONE do begin
            ERR := FSRead(Refnum, Size, @REC);
            REC.Next := nil;
           TEMPHANDLE := Pointer(NewHandle(Size));
           MARKER^^.Next := TEMPHANDLE;
           MARKER^^.Next^^ := REC;
            MARKER := MARKER^^.Next;
            ERR := FSRead(Refnum, BSIZE, @DONE);
       end; (* While *)
    end; (* If *)
end; (* ReadRente *)

(*****************************************************************************)
procedure ReadOffice(var OMarker:OfficeHandle; Size:longint; Refnum:integer);

(* CALLED BY: LoadSimulation                                                    *)
(* CALLS TO: none                                                              *)
(* GLOBALS: none                                                               *)
(* ACTION: This procedure reads the office information stored on the disk into a list.  *)

var  DONE : boolean;
     BSIZE : longint;
     ERR : OSErr;
     REC : OfficeRec;
     MARKER, TEMPHANDLE : OfficeHandle;

begin (* ReadOffice *)
    BSIZE := SizeOf(boolean);
    ERR := FSRead(Refnum, BSIZE, @DONE);
    if not DONE then begin
```

```
        ERR := FSRead(Refnum, Size, @REC);
        REC.Next := nil;
       OMarker := Pointer(NewHandle(Size));
       OMarker^^ := REC;
       ERR := FSRead(Refnum, BSIZE, @DONE);
       MARKER := OMarker;
       while not DONE do begin
           ERR := FSRead(Refnum, Size, @REC);
           REC.Next := nil;
          TEMPHANDLE := Pointer(NewHandle(Size));
          MARKER^^.Next := TEMPHANDLE;
          MARKER^^.Next^^ := REC;
           MARKER := MARKER^^.Next;
           ERR := FSRead(Refnum, BSIZE, @DONE);
       end; (* While *)
   end; (* If *)
end; (* ReadOffice *)

(*****************************************************************************************)
procedure ReadKid(var KMarker:KidHandle; Size:longint; Refnum:integer);

(* CALLED BY: LoadSimulation                                                              *)
(* CALLS TO: none                                                                         *)
(* GLOBALS: none                                                                          *)
(* ACTION: This procedure reads the kids information stored on the disk into a list.      *)

var  DONE : boolean;
    BSIZE : longint;
    ERR : OSErr;
    REC : KidRec;
    MARKER, TEMPHANDLE : KidHandle;

begin (* ReadKid *)
   BSIZE := SizeOf(boolean);
   ERR := FSRead(Refnum, BSIZE, @DONE);
   if not DONE then begin
       ERR := FSRead(Refnum, Size, @REC);
       REC.Next := nil;
      KMarker := Pointer(NewHandle(Size));
      KMarker^^ := REC;
       ERR := FSRead(Refnum, BSIZE, @DONE);
       MARKER := KMarker;
       while not DONE do begin
           ERR := FSRead(Refnum, Size, @REC);
           REC.Next := nil;
          TEMPHANDLE := Pointer(NewHandle(Size));
          MARKER^^.Next := TEMPHANDLE;
          MARKER^^.Next^^ := REC;
           MARKER := MARKER^^.Next;
           ERR := FSRead(Refnum, BSIZE, @DONE);
       end; (* While *)
   end; (* If *)
```

```
end; (* ReadKid *)

(*****************************************************************************)
procedure LoseKids(var KMarker:KidHandle);

(* CALLED BY: LoadSimulation                                                 *)
(* CALLS TO: none                                                            *)
(* GLOBALS: none                                                             *)
(* ACTION: This procedure disposes of a list of kids by disposing of their handles.  *)

var  KDISPMARK : KidHandle;

begin (* LoseKids *)
    while KMarker <> nil do begin
        KDISPMARK := KMarker;
       KMarker := KMarker^^.Next;
        DisposHandle(Pointer(KDISPMARK));
    end; (* While *)
end; (* LoseKids *)

(*****************************************************************************)
procedure LoadSimulation(var Assets:AssetsType; var Date:DateType; WatchHdl:CursHandle; FinMenu,
      PersMenu:MenuHandle; var Icons:IconType; FinWindow:WindowPtr; var CorrRefNum:integer);

(* CALLED BY: DoCommand                                                      *)
(* CALLS TO: ReadRente, LoseKids, Bankrupt, ReadOffice, ReadKid, DrawIcons   *)
(* GLOBALS: FinWindow, Icons, CorrRefNum, Assets, Date, WatchHdl, Corrfile   *)
(* ACTION: This procedure restores a previously saved game from the disk to the Macintosh.  *)

var  LEN, REFNUM, DUMMYINT : integer;
    ERR : OSErr;
    SIZE : longint;
    KMARKER, KDISPMARK : KidHandle;
    FINRECT : Rect;

begin (* LoadSimulation *)
    LEN := CautionAlert(297, nil);
    if LEN = OK then begin
        ERR := FSOpen(SAVEFILE, 0, REFNUM);
        if ERR <> NONE then begin
            LEN := StopAlert(298, nil);
        end else begin
            Hlock(Pointer(WatchHdl));
            SetCursor(WatchHdl^^);
            Hunlock(Pointer(WatchHdl));
            Bankrupt(Assets, Date, false, false);
            LoseKids(Assets.Children.Boys);
            LoseKids(Assets.Children.Girls);
            SIZE := SizeOf(DateType);
            ERR := FSRead(REFNUM, SIZE, @Date);
            SIZE := SizeOf(AssetsType);
            ERR := FSRead(REFNUM, SIZE, @Assets);
```

```
    Assets.Quit := false;
   ReadRente(Assets.Rente.IndivRentes, SizeOf(RenteType), REFNUM);
  ReadOffice(Assets.Office.OfficeList, SizeOf(OfficeRec), REFNUM);
   ReadKid(Assets.Children.Boys, SizeOf(KidRec), REFNUM);
   ReadKid(Assets.Children.Girls, SizeOf(KidRec), REFNUM);
    Icons.Selected := NONE;
    Icons.MenuDisabled := true;
   Icons.IconWasSelected := false;
    SetPort(FinWindow);
    PLSetWrPort(FinWindow);
     SetRect(FINRECT, 0, 0, 114, 183);
    EraseRect(FINRECT);
   DrawIcons(Icons, FinWindow);
    DisableItem(FinMenu, 0);
   if Assets.Marriage.Married then begin
       DisableItem(PersMenu, MARRITEM);
    end else begin
       EnableItem(PersMenu, MARRITEM);
    end; (* If *)
    if Assets.Noble then begin
       DisableItem(PersMenu, NOBLEITEM);
    end else begin
       EnableItem(PersMenu, NOBLEITEM);
    end; (* If *)
  if (Assets.Generation = SECONDGEN) and (Assets.Marriage.Married) and
      (Assets.Children.NextBirth.Year <= Date.Year) and
      ((Assets.Children.NextBirth.Year <> Date.Year) or (Assets.Children.NextBirth.Fall <> true)
       or (Date.Fall <> false)) then begin
           EnableItem(PersMenu, PLANITEM);
    end else begin
       DisableItem(PersMenu, PLANITEM);
    end; (* If *)

    HiliteMenu(0);
   DrawMenuBar;
end; (* If *)
If (Assets.Generation = SECONDGEN) and (Date.Year < 1695) then Corrfile := 'SecondMail.1.dat'
 else if (Assets.Generation = SECONDGEN) and (Date.Year >=1695) then Corrfile := 'SecondMail.2.dat'
    else if Date.Year < 1656 then Corrfile := 'FirstMail.1.dat'
            else Corrfile := 'FirstMail.2.dat';
 ERR := FSClose(REFNUM);
 ERR := FSClose (CorrRefNum);
 ERR := FSOpen(CORRFILE, 0, CorrRefNum);
SIZE := SizeOf(integer);
ERR := FSRead(CorrRefNum, SIZE, @DUMMYINT);
Assets.Mail.Year := NONE;
SIZE := SizeOf(MailRec);
with Assets do begin
   while (Mail.Year < Date.Year) or ((Mail.Year = Date.Year) and (Mail.Fall = Date.Fall)) or
       ((Date.Fall) and (Mail.Year = Date.Year)) do begin
       ERR := FSRead(CorrRefNum, SIZE, @Mail);
    end; (* While *)
```

```
        end; (* With *)
    end; (* If *)
end; (* LoadSimulation *)

{$S}

(*****************************************************************************************)
procedure DoCommand (mResult:longInt; var Icons:IconType; var Done:boolean; var Assets:AssetsType;
                myMenus:MenuArray; var Date:DateType; var hTE:TEHandle; var TopLine, MaxScroll:integer;
                WatchHdl:CursHandle; FinWindow:WindowPtr; var CorrRefNum, VRefNum:integer);

(* CALLED BY: Main                                                                       *)
(* CALLS TO: AboutProgram, ReadText, GoToNext, QuitHandler, SaveSimulation, LoadSimulation,  *)
(*         DisplayStatus, Purchase, Sell, ManageLand, DisplayStatus, DoMarriage, PlanFamily,  *)
(*         MakeWill, BuyNobility, ChooseProtector,  DisplayAssets                        *)
(* GLOBALS: FinWindow, Icons, TopLine, MaxScroll, CorrRefNum, VRefNum, Done, myMenus,    *)
(*           Assets, Date, hTE, WatchHdl                                                 *)
(* ACTION: This procedure reads the menuitem and calls the appropriate procedure in response to the  *)
(*           command given.                                                              *)

var  theItem,
    theMenu : integer;

begin (* DoCommand *)
    Done := false;
    randSeed := TickCount;
    theMenu := HiWord(mResult);
    theItem := LoWord(mResult);
    case theMenu of

    HELPMENU :
        case theItem of
            ABOUTITEM : AboutProgram;
            INSTRITEM : ReadText(TopLine, MaxScroll, hTE, WatchHdl, VRefNum);
        end; (* Case *)

    PROGRESSMENU :
        case theItem of
          NEXTITEM : GoToNext(Assets, Date, Icons, myMenus[FINMENU], myMenus[DECMENU], Done,
                        WatchHdl, CorrRefNum);
          SAVEITEM : SaveSimulation(Assets, Date, WatchHdl);
          LOADITEM : LoadSimulation(Assets, Date, WatchHdl, myMenus[FINMENU], myMenus[DECMENU],
                        Icons, FinWindow, CorrRefNum);
          QUITITEM : QuitHandler(Assets, Date, Done, WatchHdl);
    end; (* Case *)

    FINMENU :
        case theItem of
          STATUSITEM : DisplayStatus(Assets, Icons.Selected, Date);
          BUYITEM : Purchase(Assets, Icons.Selected, Date, myMenus[FINMENU], myMenus[DECMENU]);
          SELLITEM : Sell(Assets, Icons.Selected, Date, myMenus[FINMENU]);
          MANAGEITEM : ManageLand(Assets, Date);
```

```
    end; (* Case *)

    VIEWMENU :
        case theItem of
          WEALTHITEM : DisplayStatus(Assets, WEALTHDISP, Date);
          PERSITEM : DisplayStatus(Assets, PERSDISP, Date);
        end; (* Case *)

    DECMENU :
        case theItem of
          MARRITEM : DoMarriage(Assets, Date, myMenus[DECMENU]);
          PLANITEM : PlanFamily(Assets, Date, myMenus[DECMENU]);
          WILLITEM : MakeWill(Assets, false);
          NOBLEITEM : BuyNobility(Assets, myMenus[DECMENU]);
          PROCITEM : ChooseProtector(Assets, Date);
        end; (* Case *)

end; (* Case *)
    if theItem <> NONE then begin
        DisplayAssets(Assets, Date);
    end; (* If *)
    SetCursor(Arrow);
    if not Done then begin
        HiliteMenu(0);
    end; (* If *)
end; (* DoCommand *)

{$S Seg6}

(*****************************************************************************************)
procedure FinalStats(var Assets:AssetsType; Date:DateType);

(* CALLED BY: Main                                                                       *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: Assets, Date                                                                 *)
(* ACTION: This procedure stores final stats for later review using Examine.             *)

var  REFNUM : integer;
    ERR : OSErr;
    SIZE, WHEN : longint;

begin (* FinalStats *)
    ERR := FSOpen(ENDFILE, 0, REFNUM);
    if ERR <> NONE then begin
        ERR := FSClose(REFNUM);
        ERR := Create(ENDFILE, 0, '????', 'ENDS');
        ERR := FSOpen(ENDFILE, 0, REFNUM);
    end else begin
        ERR := SetFPos(REFNUM, 2, 0);
    end; (* If *)
    ERR := ReadDateTime(WHEN);
    SIZE := SizeOf(Assets.Prestige);
```

```
    ERR := FSWrite(REFNUM, SIZE, @WHEN);
    ERR := FSWrite(REFNUM, SIZE, @Assets.Prestige);
    ERR := FSWrite(REFNUM, SIZE, @Assets.TotalVal);
    SIZE := SizeOf(boolean);
    ERR := FSWrite(REFNUM, SIZE, @Assets.Won);
    SIZE := SizeOf(DateType);
    ERR := FSWrite(REFNUM, SIZE, @Date);
    ERR := FSClose(REFNUM);
end; (* FinalStats *)

(*****************************************************************************************)
procedure StopSimulation(CorrRefNum:integer);

(* CALLED BY: Main                                                                       *)
(* CALLS TO: none                                                                        *)
(* GLOBALS: CorrRefNum                                                                   *)
(* ACTION: This procedure stops the simulation.      It closes the mail file, calls up the *)
(* how-to-start-again window,and ejects the disk from the Macintosh.                     *)

var  ERR : OSErr;
    VOLNAME : Str255;
    VREFNUM : integer;
    FREEBYTES : longint;
    ENDDIALOG : DialogPtr;

begin (* StopSimulation *)
    ERR := FSClose(CorrRefNum);
    ENDDIALOG := GetNewDialog(278, nil, Pointer(-1));
    DrawDialog(ENDDIALOG);
    ERR := GetVInfo(0, @VOLNAME, VREFNUM, FREEBYTES);
    ERR := Eject(@VOLNAME, VREFNUM);
end; (* StopSimulation *)

{$S}
(* CALLED BY: This is the Main Program Loop.                                             *)
(* CALLS TO: Initialize, CalcHarvest, GoToNext, DisplayAssets, DoCommand, SelectIcon, Treasury,  *)
(*     FinalStats, SEGMENTS                                                              *)
(* GLOBALS: FinWindow, AssetWindow, WhichWindow, Icons, myEvent, Code, TopLine, MaxScroll,  *)
(*         CorrRefNum, VRefNum, Done, Temp, myMenus, Letter, Assets, Date, hTE, WatchHdl, Corrfile *)
(* ACTION: Here begins the Main program.  Variables are initialized, the first season's investments are  *)
(* calculated, and the simulation awaits a response from the player.  Responses to the player begin here.  *)

begin (* Main Program*)
    Initialize(FinWindow, AssetWindow, Icons, myMenus, Assets, Date, WatchHdl, hTE,
                                                    Corrfile, CorrRefNum, VRefNum);
    CalcHarvest(Assets.Land.Local);
    CalcHarvest(Assets.Land.Regional);
    GoToNext(Assets, Date, Icons, myMenus[FINMENU], myMenus[DECMENU], Done,
                                                    WatchHdl, CorrRefNum);
    DisplayAssets(Assets, Date);
    Done := false;
    repeat
```

```
    SystemTask;
    Temp := GetNextEvent(everyEvent, myEvent);

    case myEvent.what of
        mousedown : begin
            Code := FindWindow(myEvent.where, WhichWindow);
            if WhichWindow <> nil then begin
                SetPort(WhichWindow);
                PLSetWrPort(WhichWindow);
            end; (* If *)
            case Code of
              inMenuBar : DoCommand(MenuSelect(myEvent.Where), Icons, Done, Assets, myMenus, Date,
                                    hTE, TopLine, MaxScroll, WatchHdl, FinWindow, CorrRefNum, VRefNum);
              inSysWindow : SystemClick(myEvent, WhichWindow);
              inContent : if WhichWindow = FinWindow then begin
                            SelectIcon(Icons, myEvent, myMenus, Date, Assets);
                          end; (* If *)
            end; (* Case *)
        end; (* MouseDown *)

        KeyDown : begin
            Letter := chr(myEvent.message mod 256);
            if BitAnd(myEvent.modifiers, 256) <> 0 then begin
                DoCommand(MenuKey(Letter), Icons, Done, Assets, myMenus, Date, hTE, TopLine,
                          MaxScroll, WatchHdl, FinWindow, CorrRefNum, VRefNum);
            end; (* If *)
            if (myEvent.modifiers = 3968) and (myEvent.message = 490) then begin
                Treasury(Assets, Date);
            end; (* If *)
        end; (* KeyDown *)

    end; (* Case *)
    UnloadSeg(@SwitchGen);
    UnloadSeg(@SaveSimulation);
    UnloadSeg(@ReadText);
    UnloadSeg(@Initialize);
    UnloadSeg(@DoMarriage);
    UnloadSeg(@FinalStats);
    UnloadSeg(@GoToNext);
    UnloadSeg(@ManageLand);
    UnloadSeg(@Purchase);
    UnloadSeg(@Sell);
    UnloadSeg(@SetUpTextEdit);
    UnloadSeg(@HarvValue);
    UnloadSeg(@PutWillItems);
    UnloadSeg(@PlanFamily);
    UnloadSeg(@HidePCtl);
    UnloadSeg(@DispWealth);
    UnloadSeg(@DoPicture);
until Done;
if (not Assets.Quit) then begin
    FinalStats(Assets, Date);
```

```
    end; (* If *)
    StopSimulation(CorrRefNum);
end.  (* Main *)
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Source Code:SUNKING/4.1/FINANCER.TEXT

Date: Monday, September 22, 1986

Time: 9:38:20 PM

Printer: LaserWriter Plus

*{The Would-Be Gentleman, Faculty Author Development Program at Stanford University. }
*{Version 4.1, Steve Fisher (version 1.0) 12/20/84 and Tom Maliska (versions to 4.1) 3/12/86, 9/15/86. }
*{Faculty Author Development Team: Carolyn Lougee, Michael Carter, Ed McGuigan, Steve Fisher, }
*{and Tom Maliska. }

SunKing/4.1/Finance.RSRC

Type SIMU = STR
,0
History Simulation (version 4.1) by Steve Fisher (12/20/84) and Tom Maliska (3/13/86)

Type FREF
,128 (32)
APPL 0

Type BNDL
,128
SIMU 0
2
ICN# 1
0 128
FREF 1
1 128

Type ICN#
,128 (32)
2
00000000
00000000
00000000
00020000
00020000
01020400
00820800
00421000
003FE000
00401000
20800820
1D0005C0
02000200
02525200
02525200
02225200
7E5253F8
02522200
02522200
02000200
02000200
1D0005C0
20800820
00401000
003FE000

```
00421000
00820800
01020400
00020000
00020000
00000000
00000000
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF
FFFFFFFF

Type DITL

 ,256 (4)
 1

    BtnItem Enabled
     0 0 20 60
Ready

 ,257 (32)
 10
   BtnItem Enabled
```

240 46 260 116
OK

BtnItem Enabled
240 260 260 330
Cancel

StatText Disabled
20 137 40 400
ALLOCATE LAND

StatText Disabled
50 5 85 365
Please enter the percentage of your land you would like to be allocated in the following ways:

StatText Disabled
105 4 125 245
Percentage of land to rent in kind

StatText Disabled
145 4 165 245
Percentage of land to rent for cash

StatText Disabled
185 4 205 245
Percentage of land to sharecrop

EditText Enabled
105 260 120 320

EditText Enabled
145 260 160 320

EditText Enabled
185 260 200 320

,258 (32)
2

BtnItem Enabled
60 13 80 83
OK

StatText Disabled
8 60 50 330
Only whole numbers may be entered. Please check your answers.

,259 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You have tried to allocate more than 100% of your land. Please try again.

,260 (32)
6
BtnItem Enabled
150 46 170 116
Invest

BtnItem Enabled
150 260 170 330
Cancel

StatText Disabled
20 125 40 400
PURCHASE TEXTILES

StatText Disabled
50 4 70 400
You have ^0 livres of cash to invest.

StatText Disabled
120 4 135 245
Amount of cash to invest in textiles

EditText Enabled
120 255 135 310

,261 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't have that much cash. Please try again.

,262 (32)
6

BtnItem Enabled
150 46 170 116
Buy

BtnItem Enabled
150 260 170 330
Cancel

StatText Disabled
10 130 25 400
PURCHASE RENTES

StatText Disabled
35 10 100 350
The King is offering Rentes of 1000 livres on the city of Paris. Their face value is denier ^0; you can get them at denier ^1. You have ^2 livres of cash to spend.

StatText Disabled
115 4 130 170
Number you wish to buy

EditText Enabled
115 180 130 260

,263 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't have enough cash to purchase that many Rentes.

,264 (32)
6
BtnItem Enabled
150 46 170 116
Buy

BtnItem Enabled
150 260 170 330
Cancel

StatText Disabled
20 136 40 400
PURCHASE LAND

StatText Disabled
50 20 95 350
You have ^0 livres of cash to purchase from ^3 available hectare(s), and land costs ^2 livres per hectare.

StatText Disabled
120 4 135 260

Hectares to buy (^1 max)

EditText Enabled
120 265 135 355

,265 (32)
6
BtnItem Enabled
150 46 170 116
Sell

BtnItem Enabled
150 260 170 330
Cancel

StatText Disabled
20 136 40 400
SELL RENTES

StatText Disabled
50 20 95 350
You own ^0 Rente(s), and Rente is currently going for denier ^1.

StatText Disabled
120 20 135 220
Number of Rentes to sell

EditText Enabled
120 230 135 310

,266 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't own that many Rentes. Please try again.

,267 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't own that much miscellaneous land. Please try again.

,268 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You cannot sell land that was gained through inheritance. Please try again.

,269 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't own that much grain. Please try again.

,270 (32)
9
BtnItem Enabled
240 46 260 116
Sell

BtnItem Enabled
240 260 260 330
Cancel

StatText Disabled
30 146 50 400
SELL LAND

StatText Disabled
70 20 100 350
You own ^1 quintel(s) of grain, with grain selling for ^2 livres per quintel.

StatText Disabled
115 4 130 210
Number of quintels to sell

StatText Disabled
145 20 175 350
You own ^0 hectare(s) of miscellaneous land, with land going for ^3 livres per hectare.

StatText Disabled
190 4 205 210
Number of hectares to sell

EditText Enabled
115 220 130 270

EditText Enabled
190 220 205 270

,271 (32)
4

BtnItem Enabled
180 46 200 116
Buy

BtnItem Enabled
180 260 200 330
Cancel

StatText Disabled
20 130 35 400
PURCHASE LEASES

StatText Disabled
50 10 200 350
The King is offering a one-year lease to collect the ^0. Its face value is ^1 livres, making the official price ^2 livres. You, however, can get it for ^3 livres. Do you wish to buy it?

,272 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't have enough cash to purchase that lease.

,273 (32)
3

BtnItem Enabled
90 46 110 116
King's

BtnItem Enabled
90 260 110 330
Yours

StatText Disabled
10 10 80 360
You may either sell Rentes you have purchased from the King, or else Rentes of your own.

Which would you like to sell?

,274 (32)
6

BtnItem Enabled
145 46 165 116
Sell

BtnItem Enabled
145 260 165 330
Cancel

StatText Disabled
10 120 25 360
SELL PERSONAL RENTE

StatText Disabled
30 15 75 360
The King is currently selling Rentes for denier ^0 and you have already sold £^1 worth of personal Rentes.

StatText Disbaled
85 15 100 250
Sell Rente for how many livres?

EditText Enabled
85 260 100 350

,275 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't own any of the King's Rente, so you can't sell any.

,276 (32)
3

BtnItem Enabled
80 46 100 116
OK

BtnItem Enabled
80 260 100 330
Cancel

StatText Disabled

20 10 70 360
The best deal you can get for a Rente of that value is denier ^0.

,277 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't have enough wealth for collateral to back a Rente this large.

,278 (32)
14

BtnItem Enabled
230 30 250 100
Buy

BtnItem Enabled
230 270 250 340
Cancel

BtnItem Enabled
260 30 280 100
Next

BtnItem Enabled
260 270 280 340
Previous

StatText Disabled
5 125 20 300
PURCHASE OFFICE

StatText Disabled
27 5 57 365
Please check the office you wish to buy or choose NEXT or PREVIOUS to scan through the offices.

RadioItem Enabled
85 10 100 30

RadioItem Enabled
120 10 135 30

RadioItem Enabled
155 10 170 30

RadioItem Enabled
190 10 205 30

StatText Disabled
85 35 115 360
^0

StatText Disabled
120 35 150 360
^1

StatText Disabled
155 35 185 360
^2

StatText Disabled
190 35 220 360
^3

,279 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't have enough cash to purchase that office.

,280 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You already own that office.

,281 (32)
14

BtnItem Enabled
230 30 250 100
Sell

BtnItem Enabled
230 270 250 340

Cancel

BtnItem Enabled
260 30 280 100
Next

BtnItem Enabled
260 270 280 340
Previous

StatText Disabled
5 140 20 300
SELL OFFICE

StatText Disabled
27 5 57 365
Please check the office you wish to sell or choose NEXT or PREVIOUS to scan through the offices.

RadioItem Enabled
70 10 85 30

RadioItem Enabled
110 10 125 30

RadioItem Enabled
150 10 165 30

RadioItem Enabled
190 10 205 30

StatText Disabled
70 35 105 360
^0

StatText Disabled
110 35 145 360
^1

StatText Disabled
150 35 185 360
^2

StatText Disabled
190 35 225 360
^3

,282 (32)
3

BtnItem Enabled
90 13 110 83
OK

StatText Disabled
5 60 75 300
Because you purchased the lease to collect the Royal Toll on herring and salmon in the Carenton district, you were just hanged!!!

IconItem Disabled
10 20 42 52
261

,283 (32)
2

StatText DIsabled
10 75 25 300
Welcome to the King's Treasury!

EditText Enabled
40 60 55 300

,284 (32)
2

BtnItem Enabled
90 13 110 83
OK

StatText Disabled
5 60 75 300
You do not have enough cash to pay off your debt of £^0. You must sell a personal Rente in order to get the cash.

,285 (32)
3

BtnItem Enabled
90 31 110 131
Miscellaneous

BtnItem Enabled
90 245 110 345
Titled

StatText Disabled
10 10 80 360
You may purchase either miscellaneous properties or special titled properties. Which would you like to buy?

,286 (32)
3

BtnItem Enabled
170 13 190 83
OK

StatText Disabled
5 60 55 300
You do not have enough collateral wealth to use personal Rentes to pay off your debts.

StatText Disabled
60 60 150 300
You are therefore forced to declare bankruptcy and to liquidate all holdings except for the miscellaneous land you inherited from your father.

,287 (32)
13

BtnItem Enabled
260 30 280 100
Court

BtnItem Enabled
260 270 280 340
Cancel

BtnItem Enabled
260 145 280 215
Info

StatText Disabled
5 140 20 300
MARRIAGE

StatText Disabled
27 5 62 365
Please select a prospective bride. Choose INFO for information about her, or COURT to try to marry her.

RadioItem Enabled
85 10 100 30

RadioItem Enabled
120 10 135 30

RadioItem Enabled
155 10 170 30

RadioItem Enabled

190 10 205 30

StatText Disabled
85 35 115 360
^0

StatText Disabled
120 35 150 360
^1

StatText Disabled
155 35 185 360
^2

StatText Disabled
190 35 220 360
^3

,288 (32)
2

BtnItem Enabled
85 13 105 83
OK

StatText Disabled
10 10 75 300
^0 is ^3 years old, has a dowry worth £^1, and her Father is ^2.

,289 (32)
3

BtnItem Enabled
90 13 110 83
OK

StatText Disabled
5 60 75 300
After a hectic courtship, ^0's father has agreed to the marriage. Congratulations!!

IconItem Disabled
10 20 42 52
262

,290 (32)
2

BtnItem Enabled
90 13 110 83
OK

StatText Disabled
5 60 75 300
Since you were refused marriage for attempting a foolish courtship, no prospective bride will pay attention to you at this time.

,291 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
^0 has refused your proposal of marriage.

,292 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
Congratulations!! You have just had a bouncing baby ^0!

,293 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
It was with great sadness that you learned of the death of one of your ^0.

,294 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
You have just died! Your life now continues through the person of your son, Jean-Francois Marin!

,295 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 75 300
You have just died! Since you never had a son, your family name dies with you and the simulation ends in failure!

,296 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
Since Jean-Francois is under-age, the years pass uneventfully until he reaches his majority.

,297 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You have exceeded 100%. Please try again.

,298 (32)
18

BtnItem Enabled
260 30 280 100
Submit

BtnItem Enabled
260 270 280 340
Cancel

StatText Disabled
10 135 25 300
Make a will

StatText Disabled
37 10 105 350
In order to distribute your estate you must divide your holdings among the following categories.
Please enter the percentage of your estate which you would like to go to each category.

StatText Disabled
125 5 140 100
Oldest Son

EditText Enabled
125 105 140 170

StatText Disabled
155 5 170 100
Other Sons

EditText Enabled
155 105 170 170

StatText Disabled
185 5 200 100
Daughters

EditText Enabled
185 105 200 170

StatText Disabled
215 5 230 100
Other Kin

EditText Enabled
215 105 230 170

StatText Disabled
140 180 155 275
Non-Kin

EditText Enabled
140 280 155 345

StatText Disabled
170 180 185 275
Charity

EditText Enabled
170 280 185 345

StatText Disabled
200 180 215 275
The Church

EditText Enabled
200 280 215 345

,299 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
Age and wise counsel persuade you to consider your son's fortunes. You decide to make out a will.

,300 (32)
3

BtnItem Enabled
70 13 90 83
Keep

BtnItem Enabled
75 223 95 303
Redo

StatText Disabled
8 60 60 300
Your will is not in accord with tradition. Do you wish to redo it?

,302 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
Because of your new-born baby your will is no longer in accord with what is traditionally acceptable.

,303 (32)
4

BtnItem Enabled
95 46 115 116
Yes

BtnItem Enabled
95 260 115 330
No

StatText Disabled
15 130 30 400
FAMILY PLANNING

StatText Disabled
40 10 75 350
Do you wish to have a baby during the coming year?

,304 (32)
2

BtnItem Enabled
175 13 195 103
Continue

StatText Disabled
8 10 160 300
The Would-Be Gentleman 4.1. Faculty Author Development team: Carolyn Lougee, Steve Fisher, Michael Carter, Ed McGuigan and Tom Maliska. Copyright 1985 Carolyn Lougee & the Board of Trustees of the Leland Stanford Junior University.

,305 (32)
3

BtnItem Enabled
95 41 115 111
Yes

BtnItem Enabled
95 260 115 330
No

StatText Disabled
10 10 75 350
A letter of nobility costs £^0, and you have £^1 in cash to spend. Do you wish to buy a Letter of Nobility?

,306 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
That office automatically bestows nobility upon you!

,307 (32)
2

BtnItem Enabled
75 13 95 103

Continue

StatText Disabled
8 60 60 300
Your wife is too old to bear children.

,308 (32)
11
BtnItem Enabled
250 41 270 111
Buy

BtnItem Enabled
250 265 270 335
Cancel

StatText Disabled
5 126 20 400
BUY TITLED LAND

StatText Disabled
30 10 65 359
There are three types of titled land, differing in size and price per hectare.

StatText Disabled
70 10 105 350
Please enter the number of hectares of each you wish to buy. You have £^3 of cash to spend.

StatText Disabled
125 4 140 290
Seigneuries--75-150 hectares, at £^0.

StatText Disabled
160 4 175 290
Vicomté--300-450 hectares, at £^1.

StatText Disabled
195 4 210 290
Marquisat--600-900 hectares, at £^2.

EditText Enabled
125 295 140 345

EditText Enabled
160 295 175 345

EditText Enabled
195 295 210 345

,309 (32)
10

BtnItem Enabled
260 41 280 111
Convert

BtnItem Enabled
265 265 285 335
Cancel

StatText Disabled
10 110 25 300
CONVERT TITLED LAND

StatText Disabled
35 10 85 350
The cost of conversion is the difference between the cost of the greater titled land and the cost of the lesser titled land.

StatText Disabled
100 20 115 350
Which title would you like to convert from?

RadioItem Enabled
125 115 140 300
Seigneury

RadioItem Enabled
145 115 160 300
Vicomté

StatText Disabled
175 20 190 350
Which title would you like to convert to?

RadioItem Enabled
200 115 215 300
Vicomté

RadioItem Enabled
220 115 235 300
Marquisat

,310 (32)
3

BtnItem Enabled
90 36 110 126
Allocate

BtnItem Enabled

90 250 110 340
Convert

StatText Disabled
10 10 80 360
You may either allocate your land for the harvest or else convert lesser titled lands into greater titled lands.

,311 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You can only buy within the specified ranges. Please try again.

,312 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You don't have enough cash to purchase those lands.

,313 (32)
2

BtnItem Enabled
105 13 125 83
OK

StatText Disabled
8 60 90 300
Since the difference in cost between those titles is £^0 per hectare, that conversion would cost £^1.
You don't have that much cash.

,314 (32)
3

BtnItem Enabled
90 13 110 83
Yes

BtnItem Enabled
90 223 110 303
No

StatText Disabled
8 60 75 300
Since the difference in cost between those titles is £^0 per hectare, that conversion will cost £^1.
Will you convert?

,315 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You must have at least ^0 hectares of that title to do that conversion.

,316 (32)
3

BtnItem Enabled
75 13 95 83
Save

BtnItem Enabled
75 223 95 303
Cancel

StatText Disabled
8 60 60 300
Are you sure you want to save the game?

,317 (32)
4

BtnItem Enabled
65 13 85 83
Yes

BtnItem Enabled
100 223 120 303
Cancel

BtnItem Enabled
100 13 120 83
No

StatText Disabled
8 60 40 300
Do you wish to save the game before you quit?

,318 (32)
3

BtnItem Enabled
75 13 95 83
Restore

BtnItem Enabled
75 223 95 303
Cancel

StatText Disabled
8 60 60 300
Are you sure you want to restore a previously saved game?

,319 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
There was no previously saved game.

,320 (32)
5

BtnItem Enabled
215 13 235 83
Done

StatText Disabled
10 10 45 350
With the deaths of both Jean-Francois Marin and King Louis XIV, this simulation comes to an end.

StatText Disabled
50 10 85 350
Jean-Francois' son, Hyacinthe-Florent Marin de Montville will carry on the family name.

StatText Disabled
90 10 140 350
The simulation began under Denis Marin in 1638 with a prestige rating of 40 and, after 77 years, the third generation inherits a rating of ^0.

StatText Disabled
145 10 195 350
Thanks for using our simulation. We hope it has been both an enjoyable and an educational experience.

,322 (32)
2

BtnItem Enabled

75 13 95 83
OK

StatText Disabled
8 60 60 300
Since that office confers nobility, it cannot be sold.

,323 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
The King has decreed that all Rentes purchased since 1656 are void.

,324 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
All Rentes were reduced from denier 14 to denier 18 by order of the King.

,325 (32)
3

BtnItem Enabled
125 13 145 83
OK

StatText Disabled
8 60 60 300
The King has just realized that you purchased your offices for LESS than their TRUE value!!!

StatText Disabled
65 60 112 300
To correct this oversight, you must pay 20% of the purchase prices of your offices to the crown.

,326 (32)
3

BtnItem Enabled
130 13 150 83
OK

StatText Disabled

8 60 40 300
The King has graciously raised all office salaries by 25%.

StatText Disabled
45 60 110 300
In compensation for the raise, however, he has levied a one-time payment of 10 times the amount of the raise.

,327 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
Because of the death of your child, your will is no longer in accord with traditional practice.

,328 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
That office requires that the owner is noble.

,329 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
You cannot own more than ^0 of the King's Rente.

,330 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
That office requires that the owner is noble and has a title.

,331 (32)

2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
You have lost face among your peers for having married beneath you!

,332 (32)
14

BtnItem Enabled
260 30 280 100
Choose

BtnItem Enabled
260 270 280 340
Cancel

StatText Disabled
5 100 20 300
CHOOSE PROTECTOR

StatText Disabled
27 5 62 365
Please select your choice for Protector. Click the CHOOSE button to act on this choice.

RadioItem Enabled
85 10 100 30

RadioItem Enabled
120 10 135 30

RadioItem Enabled
155 10 170 30

RadioItem Enabled
190 10 205 30

RadioItem Enabled
225 10 240 30

StatText Disabled
85 35 115 360
^0

StatText Disabled

120 35 150 360
^1

StatText Disabled
155 35 185 360
^2

StatText Disabled
190 35 220 360
^3

StatText Disabled
225 35 255 360
None

,333 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
That Protector refuses to count you among his clientele.

,334 (32)
2

BtnItem Enabled
90 13 110 83
OK

StatText Disabled
8 60 75 300
Because you seem to be unable to choose a proper Protector, nobody will pay attention to you at this time.

,335 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
Fouquet has been arrested and all his followers have been thrown into bankruptcy!!

,336 (32)
2

BtnItem Enabled

75 13 95 83
OK

StatText Disabled
8 60 60 300
Your Protector is no longer able to assist you. You must seek a new one.

,337 (32)
3

StatText Disabled
20 175 36 400
To restart the game:

StatText Disabled
140 130 157 480
and double-click the 'Louis XIV' Icon.

StatText Disabled
180 20 300 480
If you are done, you do not need to insert the disk. Simply remove the disk and turn the Macintosh off.

,338 (32)
2

BtnItem Enabled
90 13 110 83
OK

StatText Disabled
8 60 75 300
Particelli, your Protector, has been ruined. All his followers have been thrown into bankruptcy!

,339 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
The Duke Of Burgundy will not allow you to buy leases!

,340 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled

8 60 60 300
Congratulations on your acceptance. May your family prosper!

,341 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
You have no protector at this time. Beware the vagaries of fortune!

,342 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
Your will is in accord with traditional practice.

,343 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
This decision makes your annual salaried income £^0. Your cost of living is now £^1.

,344 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
Congratulations on your appointment as ^0. May you serve well!

,345 (32)
2

BtnItem Enabled
95 13 115 83

OK

StatText Disabled
8 60 80 300
You have relinquished your post as ^0.

,346 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
There are ^3 hectare(s) of miscellaneous cultivable land available in your local area.

,347 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
You have added ^0 hectare(s) to your holdings, at a cost of £^1.

,348 (32)
2

BtnItem Enabled
100 13 120 83
OK

StatText Disabled
8 60 90 300
The peasants have requisitioned your store of grain and burned your barns because you sold little or no grain last fall. The grain has been distributed to the local population.

,349 (32)
2

BtnItem Enabled
100 13 120 83
OK

StatText Disabled
8 60 90 300
M. Condé apologizes for his inability to act in time to quell the peasant revolt with troops. He begs you accept a stipend of ^0 as a token of his good faith.

,350 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
Regrettably, you do not have the credentials to bid for this office.

,351 (32)
2
BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
Your total investment in textiles for the coming year is £^0.

,352 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
It is unwise and potentially disgraceful to change protectors so quickly. Bide your time, M. Marin!

,353 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
You have sold ^2 hectare(s) of miscellaneous land for £^1. Your current saleable holdings are now ^0 hectare(s).

,354 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300

You have sold ^2 quintel(s) of grain for £^1. Your granaries now hold ^0 quintel(s) of grain.

,355 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
At a cost of £^0, you have added titled lands to your family's holdings. May you continue to prosper!

,356 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
You have purchased ^0 Rente(s) with face value of denier ^1 at a cost of £^2. May the tides of fortune favor you!

,357 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
You have sold ^0 Rente(s) at denier ^1. Your return in cash was £^2.

,358 (32)
2

BtnItem Enabled
95 13 115 83
OK

StatText Disabled
8 60 80 300
You have been appointed to collect the ^0. May this venture profit you well!

,359 (32)
2

BtnItem Enabled
115 13 135 83
OK

StatText Disabled
8 60 80 300
You have sold a personal Rente at denier ^0. Your return in cash was £^1.

,360 (32)
2

BtnItem Enabled
70 13 90 83
OK

StatText Disabled
8 60 60 300
This is break point ^0.

,361 (32)
2

BtnItem Enabled
75 13 95 83
OK

StatText Disabled
8 60 60 300
Upon his deathbed, your Protector has provided for you. You are welcomed by the ^0.

Type ALRT
,256 (32)
150 130 250 500
258
4444

,257 (32)
150 130 250 500
259
4444

,258 (32)
150 130 250 500
261
4444

,259 (32)
150 130 250 500
263
4444

,260 (32)
150 130 250 500
266
4444

,261 (32)
150 130 250 500
267
4444

,262 (32)
150 130 250 500
268
4444

,263 (32)
150 130 250 500
269
4444

,264 (32)
150 130 250 500
272
4444

,265 (32)
150 130 250 500
275
4444

,266 (32)
150 130 250 500
277
4444

,267 (32)
150 130 250 500
279
4444

,268 (32)
150 130 250 500
280
4444

,269 (32)
60 155 195 471
282
4444

,270 (32)
60 155 195 471
284
4444

,271 (32)

60 155 265 471
286
4444

,272 (32)
60 155 185 471
288
4444

,273 (32)
60 155 185 471
289
4444

,274 (32)
60 155 195 471
290
4444

,275 (32)
60 155 180 471
291
4444

,276 (32)
60 155 180 471
292
4444

,277 (32)
60 155 180 471
293
4444

,278 (32)
60 155 180 471
294
4444

,279 (32)
60 155 195 471
295
4444

,280 (32)
60 155 180 471
296
4444

,281 (32)
150 130 250 500
297

4444

,282 (32)
60 155 180 471
299
4444

,283 (32)
150 130 250 500
300
4444

,285 (32)
60 155 180 471
302
4444

,286 (32)
60 155 265 471
304
4444

,287 (32)
60 155 180 471
306
4444

,288 (32)
60 155 180 471
307
4444

,289 (32)
34 130 154 500
310
4444

,290 (32)
150 130 250 500
311
4444

,291 (32)
150 130 250 500
312
4444

,292 (32)
60 155 210 471
313
4444

,293 (32)
60 155 195 471
314
4444

,294 (32)
150 130 250 500
315
4444

,295 (32)
60 155 180 471
316
4444

,296 (32)
60 155 195 471
317
4444

,297 (32)
60 155 180 471
318
4444

,298 (32)
60 155 180 471
319
4444

,299 (32)
34 130 289 500
320
4444

,301 (32)
60 155 180 471
322
4444

,302 (32)
150 130 250 500
323
4444

,303 (32)
150 130 250 500
324
4444

,304 (32)
60 155 225 471

325
4444

,305 (32)
60 155 230 471
326
4444

,306 (32)
60 155 180 471
327
4444

,307 (32)
60 155 180 471
328
4444

,308 (32)
150 130 250 500
329
4444

,309 (32)
60 155 180 471
330
4444

,310 (32)
60 155 180 471
331
4444

,311 (32)
60 155 180 471
333
4444

,312 (32)
60 155 195 471
334
4444

,313 (32)
60 155 180 471
335
4444

,314 (32)
60 155 180 471
336
4444

,315 (32)
60 155 195 471
338
4444

,316 (32)
60 155 180 471
339
4444

,317 (32)
60 155 180 471
340
4444

,318 (32)
60 155 180 471
341
4444

,319 (32)
60 155 180 471
342
4444

,320 (32)
60 155 200 471
343
4444

,321 (32)
60 155 200 471
344
4444

,322 (32)
60 155 200 471
345
4444

,323 (32)
150 130 250 500
346
4444

,324 (32)
60 155 200 490
347
4444

,325

60 155 220 471
348
4444

,326
60 155 220 471
349
4444

,327 (32)
60 155 200 471
350
4444

,328 (32)
150 130 250 500
351
4444

,329 (32)
60 155 200 471
352
4444

,330 (32)
60 155 190 471
353
4444

,331 (32)
60 155 190 471
354
4444

,332 (32)
60 155 190 471
355
4444

,333 (32)
60 155 190 471
356
4444

,334 (32)
60 155 210 471
357
4444

,335 (32)
50 155 190 471
358

4444

,336 (32)
34 130 225 500
359
4444

,337 (32)
60 155 180 471
360
4444

,338 (32)
60 155 180 471
361
4444

Type DLOG

,256 (4)
290 400 310 460
Visible 1 NoGoAway 0
256

,257 (32)
34 130 325 500
Visible 1 NoGoAway 0
257

,258 (32)
34 130 225 500
Visible 1 NoGoAway 0
260

,259 (32)
34 130 225 500
Visible 1 NoGoAway 0
262

,260 (32)
34 130 225 500
Visible 1 NoGoAway 0
264

,261 (32)
34 130 225 500
Visible 1 NoGoAway 0
265

,262 (32)
34 130 325 500
Visible 1 NoGoAway 0

270

,263 (32)
34 130 245 500
Visible 1 NoGoAway 0
271

,264 (32)
34 130 154 500
Visible 1 NoGoAway 0
273

,265 (32)
34 130 225 500
Visible 1 NoGoAway 0
274

,266 (32)
34 130 144 500
Visible 1 NoGoAway 0
276

,267 (32)
34 130 325 500
Visible 1 NoGoAway 0
278

,268 (32)
34 130 325 500
Visible 1 NoGoAway 0
281

,269 (32)
34 130 114 500
Visible 1 NoGoAway 0
283

,270 (32)
34 130 154 500
Visible 1 NoGoAway 0
285

,271 (32)
34 130 325 500
Visible 1 NoGoAway 0
287

,272 (32)
34 130 325 500
Visible 1 NoGoAway 0
298

,273 (32)
34 130 159 500
Visible 1 NoGoAway 0
303

,274 (32)
34 130 159 500
Visible 1 NoGoAway 0
305

,275 (32)
34 130 325 500
Visible 1 NoGoAway 0
308

,276 (32)
30 130 330 500
Visible 1 NoGoAway 0
309

,277 (32)
34 130 325 500
Visible 1 NoGoAway 0
332

,278 (32)
30 12 330 500
Visible 1 NoGoAway 0
337

Type WIND
,256 (4)
Finance Selector
17 1 200 115
Visible NoGoAway
2
0

,257 (4)
.
39 120 330 500
InVisible NoGoAway
0
0

,258 (4)
Assets
215 1 335 115
Visible NoGoAway
2
0

,259 (32)
Instructions
39 120 330 500
Visible GoAway
0
0

,260 (32)
Beginning
30 12 330 500
Visible NoGoAway
1
0

,261 (32)
Art
30 135 330 500
Visible NoGoAway
1
0

Type CNTL
,256 (4)
Vertical Scroll Bar
0 365 276 380
Visible
16
0
0 0 100

Type MENU
,1 (4)
Information
The Authors
Instructions

,2 (4)
Progression
Next Interval/N
(-
Save Game
Restore Game
(-
Quit

,3 (4)
Investments
Status
Buy
Sell
Manage

,4 (4)
Personal Decisions
Marriage
(Family Planning
Make a will
Buy Letter of Nobility
Choose Protector

,5 (4)
View
Wealth
Personal Info

Type ICON
* Office Icon
,258 (4)
001B C000
002D A000
00F6 F000
00AA D000
00FD B000
01A6 7800
0160 2800
01E0 3400
01A0 3600
02E0 2E00
07C0 3A00
0480 1600
0740 3400
0540 1E00
03C0 0E00
0080 0800
0000 0080
0000 0100
00FF FF00
0112 0280
0222 0640
0442 0620
087E 0010
1FFF FFF8
2918 1894
2573 CEA4
2400 0024
18FF FF18
1B00 00D8
1800 0018
3800 001C
3000 000C

* Rente Icon
,257 (4)
0000 0000

0000 0000
0000 0000
0000 0000
0000 0000
0007 F800
000F FC00
001C 0600
001C 0200
001C 0000
001C 0000
001C 0000
001C 0000
001C 0000
00FF 8000
00FF 8000
001C 0000
001C 0000
001C 0000
001C 0000
001C 0000
001C 0000
001C 0000
001C 0100
001C 0300
00FF FE00
00FF FC00
0000 0000
0000 0000
0000 0000
0000 0000
0000 0000

* Land Icon
,256 (4)
0000 0000
0288 0000
0174 0000
008B 0000
012A 8000
0000 4000
0000 E000
0000 2000
0000 503F
0000 78E4
5FFF C992
08A2 4E5E
2400 4B60
027F F892
28BF E40B
111F F249
0208 0A49
4AC9 2A2D

22C9 2A25
12C8 0A25
03FF FF00
3C00 01E0
4030 283E
0C83 0303
0000 6040
0330 0A18
6003 0100
1060 0060
0C0C 6008
6100 0600
1881 80C0
0820 3000

* Lease Icon
,259 (4)
0000 0000
0000 0000
0000 0000
7FFF FFCE
C000 005A
AC02 016E
A9B7 6B94
9DB2 4D24
9000 0648
F39E 0888
1001 1110
11E7 2230
1000 4450
13C4 8890
1001 9110
1B75 2258
0803 4C08
099E 91C8
0402 6004
059B 8774
0403 0004
06DB 1DB6
0202 0002
015F 38EA
010E 0002
1F4D 1CD2
313C 8006
233C 9DC4
3638 800C
0FFF FFF8
003F 8000
0000 0000

* Textile Icon
,260 (4)

0000 0000
0000 0000
0000 0000
0000 0000
001F F800
0068 1600
00A4 2500
0112 4880
0209 9040
0426 6420
0442 4220
0440 0220
0441 8220
0440 0220
0440 0220
0221 8440
013C 3C80
0144 2280
0089 9100
0090 0900
00A0 0500
00C1 8300
0020 0400
0040 0200
0061 8600
0038 1C00
000F F000
0000 0000
0000 0000
0000 0000
0000 0000
0000 0000

* Hang Icon
,261 (32)
FFFF FFFF
8000 2001
8000 2001
8000 2001
8000 2001
8000 2001
8000 4001
8000 4001
8000 C001
8000 8001
8000 8001
800F 8001
8010 8001
8029 4001
8020 4001
8016 8001
8010 8001

800F 0001
800F 0001
8030 C001
8060 6001
8040 2001
8040 2001
8050 A001
8050 A001
8070 E001
8030 C001
8010 8001
81F0 F801
8210 8401
8410 8201
FFFF FFFF

* Heart Icon
,262 (32)
00000000
00000000
00000000
00000000
00000040
00000040
00000170
00000100
000001C0
00F1E200
010A1400
02040800
04001400
0831A200
084A4200
0884E200
08812200
08802200
04404400
02208800
01111000
008A2000
00444000
00A08000
05110000
060A0000
07040000
00000000
00000000
00000000
00000000
00000000

Type CODE

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/EXEC.TEXT

Date: Thursday, September 18, 1986

Time: 11:51:41 PM

Printer: LaserWriter Plus

```
$EXEC  {3.0only/example/exec.text}
${3.0 version, assumes automatic generator invocation                    }
$
${ This exec file compiles the programs Sunking/4.0/= and Sunking/4.1/=.}
${ It is invoked with the Lisa Workshop run command and the syntax }
${ <exec(filename), e.g. <exec(Sunking/4.1/finance) compiles the main program. }
$
${ This single exec file can generate a Macintosh resource file from most of the   }
${ example source files.  The source can be Pascal, or assembly, or both.  The      }
${ naming convention is that assembly files have 'ASM' appended to the file name,   }
${ and resource files have an added 'R', although this can easily be changed below. }
$
${ The exec file is run by typing 'R' from the command line, then typing a line of  }
${ of the form:                                                                     }
$
${    <Example/Exec[([pascal_source], [assembler_source], [resource_file],          }
${              [source_volume], [library_volume])]                                 }
$
${ Each of the elements in the square brackets are optional.  The default values are}
$
${ Defaults:  pascal_source   =  'example/file'                                     }
${          assembler_source =  pascal_source'ASM'                                  }
${          resource_file    =  pascal_source'R'                                    }
${          source_volume     =  prefix volume                                      }
${          library_volume    =  prefix volume                                      }
$
${ This will work for Samp, File, Grow, Scroll, ShowPaint,SoundLab, PicScrap,       }
${ Modal,and most other applications, including Gentleman support programs.         }
${ However it does not set the creator or bundle bit in MacCom to add an icon       }
${ (e.g.: for File set creator = CARY and set bundle bit Yes)                       }
${ To build applications which use Graf3D (e.g. Boxes and SineGrid) change }
${ this exec to link with two additional files (see below)                          }
${ It will not work for desk accessories (e.g. ADeskAcc)                            }
$
$DEFAULT %0 TO 'SunKing/4.0/finance'
{Sets  file name defaults for the Would-Be Gentleman.}
$DEFAULT %1 TO CONCAT(%0, 'Asm')
$DEFAULT %2 TO CONCAT(%0, 'R')
$
$IF %3 <> '' THEN                {If a source volume is specified, }
   $SET %8 TO CONCAT('-', %3, '-') {set '%8' to the name of the source volume}
$ELSE
   $SET %8 TO ''                {otherwise use the prefix volume }
$ENDIF
$IF %4 <> '' THEN                {If a library volume is specified, }
   $SET %7 TO CONCAT('-', %4, '-') {set '%7' to the name of the library volume}
$ELSE
   $SET %7 TO ''
$ENDIF
$
$
$SET %9 TO 'F'                   {Start out assuming there is no file to assemble}
```

```
      $IF EXISTS(CONCAT(%8, %0, 'L.OBJ')) THEN
        $IF NEWER(CONCAT(%8, %1, '.TEXT'), CONCAT(%8, %0, 'L.OBJ')) THEN
          $SET %9 TO 'T'
        $ENDIF
      $ELSE
        $SET %9 TO 'T'
      $ENDIF
    $ENDIF
$ENDIF
$
$
$IF %9 = 'T' THEN
L{ink}%6
?
+X

%7obj/QuickDraw
%7obj/ToolTraps
%7obj/OSTraps
%7obj/PrLink
%7obj/PackTraps
%7obj/SaneLibAsm
{To use "The Old World" of SANE replace this with %7obj/Sane, %7obj/SaneAsm,}
        {%7obj/Elems, and %7obj/ElemsAsm}
%7obj/PasLib   {Mac PasLib is composed of the next four files}
%7obj/PasLibAsm
%7obj/PasInit
%7obj/RTLib
    $IF EXISTS(CONCAT(%8, %1, '.TEXT')) THEN
      $IF CONCAT(%8, %1, '.OBJ') <> %6 THEN
%8%1.obj
      $ENDIF
    $ENDIF


%8%0L.OBJ
$ENDIF
$
$
R{un}%7RMaker
%8%2
$
$
R{un}%7MacCom
R{emove example/}Y
FYL%0.RSRC
%0
APPL{type APPL}
SIMU{creator ????}
Y{o bundle bit --- change if you want it set}E{ject}Q{uit}F{iler}D{elete}%0.errors.text
Y{es}Q{uit}
$ENDEXEC
```

```
$IF EXISTS(CONCAT(%8, %1, '.TEXT')) THEN        {If a text ASM file exists, }
  $IF NOT(EXISTS(CONCAT(%8, %1, '.OBJ'))) THEN   {and if no code file exists,}
    $SET %9 TO 'T'                               {then assemble it        }
  $ELSEIF NEWER(CONCAT(%8, %1, '.TEXT'), CONCAT(%8, %1, '.OBJ')) THEN
    $SET %9 TO 'T'      {Otherwise assemble if the text is newer than the code  }
  $ENDIF
$ENDIF
$
$IF %9 = 'T' THEN                {Assemble if the assembly file is true}
  $WRITELN CONCAT('Assemble: ', %8, %1, '.TEXT')  {a debugging statement}
A{ssemble}%8%1                   {&8 is the volume prefix, and &1 is the file name}
                        {this blank line is for the listing file}
                        {this blank line is for the default output file}
$ENDIF
$
$
$SET %9 TO 'F'                   {Assume there is no Pascal program}
$IF EXISTS(CONCAT(%8, %0, '.TEXT')) THEN
  $IF NOT(EXISTS(CONCAT(%8, %0, '.OBJ'))) THEN
    $SET %9 TO 'T'
  $ELSEIF NEWER(CONCAT(%8, %0, '.TEXT'), CONCAT(%8, %0, '.OBJ')) THEN
    $SET %9 TO 'T'
  $ENDIF
$ENDIF
$
$IF %9 = 'T' THEN
$WRITELN CONCAT('Compile: ', %8, %0, '.TEXT')
P{ascal}$M+
%8%0

%8%0
$ENDIF
$
$
$SET %6 TO ''
$SET %9 TO 'F'
$IF EXISTS(CONCAT(%8, %0, '.TEXT')) THEN
  $SET %6 TO CONCAT(%8, %0, '.OBJ')
  $IF EXISTS(CONCAT(%8, %0, 'L.OBJ')) THEN
    $IF NEWER(CONCAT(%8, %0, '.TEXT'), CONCAT(%8, %0, 'L.OBJ')) THEN
      $SET %9 TO 'T'
    $ENDIF
  $ELSE
    $SET %9 TO 'T'
  $ENDIF
$ENDIF
$
$IF %9 = 'F' THEN
  $IF EXISTS(CONCAT(%8, %1, '.TEXT')) THEN
    $IF %6 = '' THEN
      $SET %6 TO CONCAT(%8, %1, '.OBJ')
    $ENDIF
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/EXAMINE.TEXT

Date: Thursday, September 18, 1986

Time: 11:10:15 PM

Printer: LaserWriter Plus

```
{$M+}     {mac code}
{$X-}     {no automatic stack expansion}
{$R-}     {no range checking, paslib is buggy}

{The Would-Be Gentleman, Faculty Author Development Program at Stanford University. }
{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86.                        }
{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan,      }
{Steve Fisher, and Tom Maliska.                                                      }
{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford      }
{Junior University.                                                                  }

program Examine;
{ACTION: This support program opens the file FINAL STATS recorded during the         }
{simulation. This aids the instructor's evaluation of student progress.              }


uses  {$U-}
    {$U obj/MemTypes}          MemTypes,
    {$U obj/QuickDraw}     QuickDraw,
    {$U obj/OSIntf}            OSIntf,
    {$U obj/ToolIntf}          ToolIntf,
    {$U Obj/PackIntf}          PackIntf,
    {$U Obj/MacPrint}          MacPrint,
    {$U Obj/PasLibIntf}    PasLibIntf;

const       ENDFILE = 'Final Stats';

type DateType = record
        Year : longint;
        Fall : boolean;
    end; (* DateType *)

var  myWindow : WindowPtr;

procedure Initialize(var myWindow:WindowPtr);

begin (* Initialize *)
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyevent, 0);
    InitWindows;
    TEInit;
    InitDialogs(nil);
    InitMenus;
    InitCursor;
    myWindow := GetNewWindow(256, nil, Pointer(-1));
    SetPort(myWindow);
    PLSetWrPort(myWindow);
    MoveTo(0,30);
    TextSize(9);
    TextFont(Monaco);
end; (* Initialize *)
```

```
procedure DispInfo;

var  ERR : OSErr;
     REFNUM : integer;
     SIZE, PRESTIGE, WHEN, WEALTH : longint;
     WON : boolean;
     DATE : DateType;
     DATEREC : DateTimeRec;

begin (* DispInfo *)
     ERR := FSOpen(ENDFILE, 0, REFNUM);
     if ERR <> 0 then begin
          writeln(' This player has never finished any games.');
     end else begin
          writeln(' Date Played    Prestige   Game Status      Final Game Date   Wealth');
          writeln;
          while ERR = 0 do begin
               SIZE := SizeOf(longint);
               ERR := FSRead(REFNUM, SIZE, @WHEN);
               if ERR = 0 then begin
                    ERR := FSRead(REFNUM, SIZE, @PRESTIGE);
                    ERR := FSRead(REFNUM, SIZE, @WEALTH);
                    SIZE := SizeOf(boolean);
                    ERR := FSRead(REFNUM, SIZE, @WON);
                    SIZE := SizeOf(DateType);
                    ERR := FSRead(REFNUM, SIZE, @DATE);
                    Secs2Date(WHEN, DATEREC);
                      write(' ',DATEREC.Month:2,'/',DATEREC.Day:2,'/',DATEREC.Year:4,' ');
                    write(PRESTIGE);
                    if WON then begin
                         write('      complete        ');
                    end else begin
                         write('     not complete');
                    end; (* If *)
                    if DATE.Fall then begin
                         write('       Fall, ');
                    end else begin
                         write('     Spring, ');
                    end; (* If *)
                    write(DATE.Year:0);
                     writeln('       £',WEALTH:0);
               end; (* If *)
          end; (* While *)
     end; (* If *)
     ERR := FSClose(REFNUM);
end; (* DispInfo *)

procedure DoneButton;

var  DONEDIALOG : DIalogPtr;
     ITEM : integer;
```

```
begin (* DoneButton *)
   DONEDIALOG := GetNewDialog(256, nil, Pointer(-1));
   repeat
      SystemTask;
      ModalDialog(nil, ITEM);
   until ITEM = OK;
   DisposDialog(DONEDIALOG);
end; (* DoneButton *)

begin (* Main *)
   Initialize(myWindow);
   DispInfo;
   DoneButton;
end. (* Main *)
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/EXAMINER.TEXT

Date: Thursday, September 18, 1986

Time: 11:30:41 PM

Printer: LaserWriter Plus

```
*{The Would-Be Gentleman, Faculty Author Development Program at Stanford University.}
*{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86.                        }
*{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan,     }
*{Steve Fisher, and Tom Maliska.                                                      }
*{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford     }
*{Junior University.                                                                 }

SunKing/4.0/Examine.RSRC

Type DITL
 ,256 (4)
 1

    BtnItem Enabled
    0 0 20 60
Quit

Type DLOG
 ,256 (4)
 290 400 310 460
 Visible 1 NoGoAway 0
 256

Type WIND

 ,256 (4)
 Finance Selector
 0 0 342 512
 Visible NoGoAway
 2
 0


Type CODE
 SunKing/4.0/ExamineL,0
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/LEASE.TEXT

Date: Thursday, September 18, 1986

Time: 11:37:26 PM

Printer: LaserWriter Plus

```
{$M+}    {mac code}
{$X-}    {no automatic stack expansion}
{$R-}    {no range checking, paslib is buggy}

{The Would-Be Gentleman, Faculty Author Development Program at Stanford University. }
{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86.                        }
{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan,      }
{Steve Fisher, and Tom Maliska.                                                      }
{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford      }
{Junior University.                                                                  }

program Lease;
{ACTION: This program creates the Lease data file required by the simulation.        }

uses  {$U-}
    {$U obj/MemTypes}        MemTypes,
    {$U obj/QuickDraw}    QuickDraw,
    {$U obj/OSIntf}          OSIntf,
    {$U obj/ToolIntf}        ToolIntf,
    {$U Obj/PackIntf}        PackIntf,
    {$U Obj/MacPrint}        MacPrint,
    {$U Obj/PasLibIntf}   PasLibIntf;

const       AddItem = 1;
       DeleteItem = 2;
       SaveItem = 3;
       QuitItem = 4;
       DelTextItem = 3;
       AddTextItem = 3;
       DriveNum = 0;
       FileName = 'Lease.Dat';
       NumLeases = 100;

type LeaseRec = record
       Title : Str255;
       Deleted : boolean;
   end; (* LeaseRec *)

   LeaseArray = array [1..NumLeases] of LeaseRec;

   LeaseType = record
      List : LeaseArray;
      Number,              (* Deleted plus undeleted *)
      RealNum : integer;       (* Only undeleted *)
   end; (* LeaseType *)

var  WhichWindow, TextWindow : WindowPtr;
   Done,
   Temp : boolean;
   myEvent : EventRecord;
   Code : integer;
   Letter : char;
```

```
    myMenu : MenuHandle;
    Leases : LeaseType;

procedure DebugDelay;

begin (* DebugDelay *)
    repeat
        SystemTask;
    until Button;
end; (* DebugDelay *)

procedure SetUpMenus(var myMenu:MenuHandle);

begin (* SetUpMenus *)
    InitMenus;
    myMenu := GetMenu(256);
    InsertMenu(myMenu, 0);
    DrawMenuBar;
end; (* SetUpMenus *)

procedure SetUpWindow(var TextWindow:WindowPtr);

begin (* SetUpWindow *)
    TextWindow := GetNewWindow(256, nil, Pointer(-1));
    SetPort(TextWindow);
    PLSetWrPort (TextWindow);
end; (* SetUpWindow *)

procedure Initialize(var myMenu:MenuHandle; var TextWindow:WindowPtr; var Leases:LeaseType);

begin (* Initialize *)
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyEvent, 0);
    InitWindows;
    SetUpMenus(myMenu);
    SetUpWindow(TextWindow);
    TEInit;
    InitDialogs(nil);
    InitCursor;
    Leases.Number := 0;
    Leases.RealNum := 0;
end; (* Initialize *)

procedure GetLeases (var Leases:LeaseType);

var  ERR : OSErr;
    REFNUM, I, NUM : integer;
    RECLEN : longint;
    TITLE : Str255;

begin (* GetLeases *)
```

```
    ERR := FSOpen(FileName, 0, REFNUM);
    if ERR = 0 then begin
        RECLEN := SizeOf(Leases.RealNum);
        ERR := FSRead(REFNUM, RECLEN, @NUM);
        Leases.RealNum := NUM;
        Leases.Number := Leases.RealNum;
        RECLEN := SizeOf(Str255);
        for I := 1 to Leases.Number do begin
            ERR := FSRead(REFNUM, RECLEN, @TITLE);
            Leases.List[I].Title := TITLE;
            Leases.List[I].Deleted := false;
        end; (* For *)
        ERR := FSClose(REFNUM);
    end; (* If *)
end; (* GetLeases *)

procedure DisplayLeases (Leases:LeaseType; TextWindow:WindowPtr);

var  I : integer;
    DISPRECT : Rect;

begin (* DisplayLeases *)
    SetPort(TextWindow);
    PLSetWrPort (TextWindow);
    SetRect(DISPRECT, 0, 0, 512, 342);
    EraseRect(DISPRECT);
    MoveTo(0,30);
    for I := 1 to Leases.Number do begin
        if not Leases.List[I].Deleted then begin
            writeln(I:0,'      ',Leases.List[I].Title);
        end; (* If *)
    end; (* For *)
end; (* DisplayLeases *)

procedure AddLeases (var Leases:LeaseType);

var  LEASEDIALOG : DialogPtr;
    ITEM, DUMMYTYPE : integer;
    DUMMYRECT : Rect;
    ITEMHDL : Handle;

begin (* AddLeases *)
    LEASEDIALOG := GetNewDialog (257, nil, Pointer(-1));
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
    until ITEM in [OK, Cancel];
    if ITEM = OK then begin
        Leases.Number := Leases.Number + 1;
        Leases.RealNum := Leases.RealNum + 1;
        GetDItem(LEASEDIALOG, AddTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, Leases.List[Leases.Number].Title);
```

```
        Leases.List[Leases.Number].Deleted := false;
    end; (* If *)
    DisposDialog(LEASEDIALOG);
end; (* AddLeases *)

function PowerOfTen(Num : integer) : integer;

var  I, TEMP : integer;

begin (* PowerOfTen *)
    TEMP := 1;
    for I := 1 to Num do begin
        TEMP := TEMP * 10;
    end; (* For *)
    PowerOfTen := TEMP;
end; (* PowerOfTen *)

function ConvertNum(StrNum : Str255) : integer;

var  I, TEMP : integer;

begin (* ConvertNum *)
    TEMP := 0;
    if Length(StrNum) <> 0 then begin
        for I := 1 to Length(StrNum) do begin
            TEMP := TEMP + (ord(StrNum[I]) - ord('0')) * PowerOfTen(Length(StrNum) - I);
        end (* For *)
    end; (* If *)
    ConvertNum := TEMP;
end; (* ConvertNum *)

procedure DelLeases (var Leases:LeaseType);

var  LEASEDIALOG : DialogPtr;
    ITEM, DUMMYTYPE, NUM : integer;
    DUMMYRECT : Rect;
    ITEMHDL : Handle;
    NUMSTR : Str255;

begin (* DelLeases *)
    LEASEDIALOG := GetNewDialog (256, nil, Pointer(-1));
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
    until ITEM in [OK, Cancel];
    if (ITEM = OK) then begin
        Leases.RealNum := Leases.RealNum - 1;
        GetDItem(LEASEDIALOG, AddTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, NUMSTR);
        NUM := ConvertNum(NUMSTR);
        if NUM <= NumLeases then begin
            Leases.List[NUM].Deleted := true;
```

```
        end; (* If *)
    end; (* If *)
    DisposDialog(LEASEDIALOG);
end; (* DelLeases *)

procedure SaveLeases (Leases:LeaseType);

var  RECLEN : longint;
    REFNUM, I, NUM : integer;
    ERR : OSErr;
    TITLE : Str255;

begin (* SaveLeases *)
    RECLEN := SizeOf(Leases.RealNum);
    ERR := FSDelete(FileName, 0);
    ERR := Create(FileName, 0, '????', 'LEAS');
    ERR := FSOpen(FileName, 0, REFNUM);
    NUM := Leases.RealNum;
    ERR := FSWrite(REFNUM, RECLEN, @NUM);
    RECLEN := SizeOf(TITLE);
    for I := 1 to Leases.Number do begin
        if not Leases.List[I]. Deleted then begin
            TITLE := Leases.List[I].Title;
            ERR := FSWrite(REFNUM, RECLEN, @TITLE);
        end; (* If *)
    end; (* For *)
    ERR := FSClose(REFNUM);
end; (* SaveLeases *)

procedure DoCommand(mResult:longint; var Done:boolean; var Leases:LeaseType);

var  theItem : integer;

begin (* DoCommand *)
    Done := false;
    theItem := LoWord(mresult);
    case theItem of
        AddItem : AddLease(Leases);
        DeleteItem : DelLease(Leases);
        SaveItem : SaveLeases(Leases);
        QuitItem : Done := true;
    end; (* Case *)
    if not Done then begin
        HiliteMenu(0);
    end; (* If *)
end; (* DoCommand *)

begin (* Main *)
    Initialize(myMenu, TextWindow, Leases);
    GetLeases(Leases);
    Done := false;
    repeat
```

```
        SystemTask;
        Temp := GetNextEvent(everyEvent, myEvent);

        case myEvent.what of

            MouseDown : begin
                Code := FindWindow(myEvent.where, WhichWindow);
                case Code of
                    inMenuBar : DoCommand(MenuSelect(myEvent.where), Done, Leases);
                    inSysWindow : SystemClick(myEvent, WhichWindow);
                end; (* Case *)
            end; (* MouseDown *)

            KeyDown : begin
                Letter := chr(myEvent.message mod 256);
                if BitAnd(myEvent.modifiers, 256) <> 0 then begin
                    DoCommand(MenuKey(Letter), Done, Leases);
                end; (* If *)
            end; (* KeyDown *)

            UpdateEvt : begin
                        BeginUpdate(TextWindow);
                        EndUpdate(TextWindow);
                        DisplayLeases(Leases, TextWindow);
                    end; (* UpdateEvt *)

        end; (* Case *)
    until Done;
end. (* Main *)
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/LEASER.TEXT

Date: Thursday, September 18, 1986

Time: 11:40:06 PM

Printer: LaserWriter Plus

```
*{The Would-Be Gentleman, Faculty Author Development Program at Stanford University.}
*{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86.                        }
*{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan,     }
*{Steve Fisher, and Tom Maliska.                                                      }
*{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford     }
*{Junior University.                                                                 }

SunKing/4.0/Lease.RSRC

Type Menu
 ,256
 Main Menu
   Add Lease /A
   Delete Lease/D
   Save Data/S
   Quit/Q

Type WIND
 ,256 (32)
 Display
 0 0 342 512
 Visible NoGoAway
 1
 0

Type DITL
 ,256 (32)
 4

   BtnItem Enabled
   50 13 70 83
OK

   BtnItem Enabled
   50 300 70 370
Cancel

   EditText Enabled
   10 170 25 230


   StatText Disabled
   10 5 25 160
Number to delete

 ,257 (32)
 4

   BtnItem Enabled
   80 13 100 83
OK
```

BtnItem Enabled
80 300 100 370
Cancel

EditText Enabled
35 5 50 330

StatText Disabled
10 5 25 230
This is a lease to collect the...

Type DLOG
,256 (32)
50 40 140 472
Visible 1 NoGoAway 0
256

,257 (32)
50 40 160 472
Visible 1 NoGoAway 0
257

Type CODE
SunKing/4.0/LeaseL,0

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/MAIL.TEXT

Date: Thursday, September 18, 1986

Time: 11:41:17 PM

Printer: LaserWriter Plus

```
{$M+}    {mac code}
{$X-}    {no automatic stack expansion}
{$R-}    {no range checking, paslib is buggy}

{The Would-Be Gentleman, Faculty Author Development Program at Stanford University. }
{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86.                        }
{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan,      }
{Steve Fisher, and Tom Maliska.                                                      }
{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford      }
{Junior University.                                                                  }

program Mail;
{ACTION: This program creates a mail file "Mail.dat".  This file can be renamed to make  }
{the correspondence files FirstMail.1.dat, FirstMail.2.dat, SecondMail.1.dat, and       }
{SecondMail.2.dat used by the simulation.                                              }

uses {$U-}
    {$U obj/MemTypes}       MemTypes,
    {$U obj/QuickDraw}      QuickDraw,
    {$U obj/OSIntf}         OSIntf,
    {$U obj/ToolIntf}       ToolIntf,
    {$U Obj/PackIntf}       PackIntf,
    {$U Obj/MacPrint}       MacPrint,
    {$U Obj/PasLibIntf}     PasLibIntf;

const       AddItem = 1;
        DeleteItem = 2;
        EditItem = 3;
        SaveItem = 4;
        QuitItem = 5;
        ListItem = 6;
        DelTextItem = 3;
        EdTextItem = 3;
        ContTextItem = 3;
        CashTextItem = 20;
        YearTextItem = 16;
        GENITEM = 13;
        FALLITEM = 17;
        SPRINGITEM = 18;
        NUMITEM = 22;
        CONOFFSET = 3;
        NUMCONTACTS = 11;
        NUMDISP = 40;
        DriveNum = 0;
        FileName = 'Mail.Dat';
        MaxMail = 115;
        FontSize = 9;
        SCREENFUL = 20;

type ProctType = (Cornuel, Mazarin, Particelli, Conde, Fouquet, Colbert, Maintenon, GrandDauphin,
            DukeOfBurgundy, Generic, NoProtector);
```

```
    MailRec = record
        Content : Str255;
        Contact : ProctType;
        Year,
        Cash : longint;
        Fall : boolean;
    end; (* MailRec *)

    MailArray = array [1..MaxMail] of MailRec;

    MailType = record
        List : MailArray;
        Number : integer;
    end; (* MailType *)

var  WhichWindow, TextWindow : WindowPtr;
    Done,
    Temp : boolean;
    myEvent : EventRecord;
    Code, VRefNum : integer;
    Letter : char;
    myMenu : MenuHandle;
    Mails : MailType;

procedure DebugDelay;

begin (* DebugDelay *)
    repeat
        SystemTask;
        Temp := GetNextEvent(everyEvent, myEvent);
    until Button;
end; (* DebugDelay *)

procedure SetUpMenus(var myMenu:MenuHandle);

begin (* SetUpMenus *)
    InitMenus;
    myMenu := GetMenu(256);
    InsertMenu(myMenu, 0);
    DrawMenuBar;
end; (* SetUpMenus *)

procedure SetUpWindow(var TextWindow:WindowPtr);

begin (* SetUpWindow *)
    TextWindow := GetNewWindow(256, nil, Pointer(-1));
    SetPort(TextWindow);
    PLSetWrPort(TextWindow);
end; (* SetUpWindow *)

procedure Initialize(var myMenu:MenuHandle; var TextWindow:WindowPtr; var Mails:MailType);
```

```
begin (* Initialize *)
   InitGraf(@thePort);
   InitFonts;
   FlushEvents(everyEvent, 0);
   InitWindows;
   SetUpMenus(myMenu);
   SetUpWindow(TextWindow);
   TEInit;
   InitDialogs(nil);
   InitCursor;
   Mails.Number := 0;
end; (* Initialize *)

procedure GetMails (var Mails:MailType);

var  ERR : OSErr;
   REFNUM, I, NUM : integer;
   RECLEN : longint;
   MAIL : MailRec;

begin (* GetMails *)
   ERR := FSOpen(FileName, 0, REFNUM);
   if ERR = 0 then begin
      RECLEN := SizeOf(Mails.Number);
      ERR := FSRead(REFNUM, RECLEN, @NUM);
      Mails.Number := NUM;
      RECLEN := SizeOf(MailRec);
      for I := 1 to Mails.Number do begin
         ERR := FSRead(REFNUM, RECLEN, @MAIL);
         Mails.List[I] := MAIL;
      end; (* For *)
      ERR := FSClose(REFNUM);
   end; (* If *)
end; (* GetMails *)

procedure DisplayNote (TextWindow:WindowPtr);

var  I, J, K, LEN : integer;
   DISPRECT : Rect;
   SEASON, CONT : Str255;

begin (* DisplayNote *)
   SetPort(TextWindow);
   PLSetWrPort(TextWindow);
   SetRect(DISPRECT, 0, 0, 512, 342);
   EraseRect(DISPRECT);
   TextFont(Geneva);
   TextSize(FontSize);
   MoveTo(0,30);
   writeln(' Give a command from the menu or type Apple + L to list all mail items.  Maximum number = ',
      MaxMail:0);
   writeln;
```

```
    TextFont(0);
    TextSize(0);
end; (* DisplayNote *)

procedure DisplayMails (var Mails:MailType; TextWindow:WindowPtr);

var  I, J, K, LEN : integer;
    DISPRECT : Rect;
    SEASON, CONT : Str255;

begin (* DisplayMails *)
    SetPort(TextWindow);
    PLSetWrPort(TextWindow);
    SetRect(DISPRECT, 0, 0, 512, 342);
    EraseRect(DISPRECT);
    TextFont(Geneva);
    TextSize(FontSize);
    MoveTo(0,30);
    K := 0;
    for I := 1 to Mails.Number do begin
        with Mails.List[I] do begin
            write('       ',I:0,')  ');
            if Length(Content) < NUMDISP then begin
                LEN := Length(Content);
            end else begin
                LEN := NUMDISP;
            end; (* If *)
            if Fall then begin
                SEASON := 'Fall';
            end else begin
                SEASON := 'Spring';
            end; (* If *)
            case Contact of
                Cornuel : CONT := 'Cornuel';
                Mazarin : CONT := 'Mazarin';
                Particelli : CONT := 'Particelli';
                Conde : CONT := 'Conde';
                Fouquet : CONT := 'Fouquet';
                Colbert : CONT := 'Colbert';
                DukeOfBurgundy : CONT := 'Duke of Burgundy';
                Maintenon : CONT := 'Maintenon';
                GrandDauphin : CONT := 'Grand Dauphin';
                Generic : CONT := 'Generic';
                NoProtector : CONT := 'No Protector';
            end; (* Case *)
            for J := 1 to LEN do begin
                write(Content[J]);
            end; (* For *)
              writeln('   ',CONT,'    ',Year:0,'   ',SEASON,'      ',Cash:0);
            K := K + 1;
            if K >= SCREENFUL then
            begin
```

```
            writeln;
            writeln('      Press the Mouse Button to See More Mail.');
            K := 0;
            DebugDelay;
            end;  (* If *)
        end; (* With *)
    end; (* For *)
    TextFont(0);
    TextSize(0);
end; (* DisplayMails *)

function PowerOfTen(Num : integer) : longint;

var  I, TEMP : longint;

begin (* PowerOfTen *)
    TEMP := 1;
    for I := 1 to Num do begin
        TEMP := TEMP * 10;
    end; (* For *)
    PowerOfTen := TEMP;
end; (* PowerOfTen *)

function ConvertNum(StrNum : Str255) : longint;

var  I, TEMP : longint;

begin (* ConvertNum *)
    TEMP := 0;
    if Length(StrNum) <> 0 then begin
        for I := 1 to Length(StrNum) do begin
            TEMP := TEMP + (ord(StrNum[I]) - ord('0')) * PowerOfTen(Length(StrNum) - I);
        end (* For *)
    end; (* If *)
    ConvertNum := TEMP;
end; (* ConvertNum *)

procedure NumSpecs(Num:longint; var Len:integer; var STR:Str255);

(* This procedure returns the String representation of the number NUM in the variable STR. The number of *)
(* digits is returned through LEN. *)

var  NEWNUM : String[1];

begin (* NumSpecs *)
    LEN := 0;
     STR := '';
    if Num = 0 then begin
         STR := ' ';
         STR[1] := '0';
        LEN := 1;
    end; (* If *)
```

```
    while Num <> 0 do begin
      LEN := LEN + 1;
      NEWNUM := ' ';
      NEWNUM[1] := chr(Num mod 10 + ord('0'));
      STR := Concat(NEWNUM, STR);
      Num := Num div 10;
    end; (* While *)
end; (* NumSpecs *)

procedure GetOldVals(MailDialog:DialogPtr; var Mails:MailType; Number:longint);

var  DUMMYTYPE : integer;
    ITEMHDL : Handle;
    DUMMYRECT : Rect;
    LEN : integer;
    TEMP : Str255;

begin (* GetOldVals *)
    with Mails.List[Number] do begin
      GetDItem(MailDialog, ContTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      SetIText(ITEMHDL, Content);
      GetDItem(MailDialog, CONOFFSET + ord(Contact) + 1, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      SetCtlVal(Pointer(ITEMHDL), 1);
      NumSpecs(Year, LEN, TEMP);
      GetDItem(MailDialog, YearTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      SetIText(ITEMHDL, TEMP);
      if Cash < 0 then begin
        NumSpecs(0 - Cash, LEN, TEMP);
        TEMP := Concat ('-',TEMP);
      end else begin
        NumSpecs(Cash, LEN, TEMP);
      end; (* If *)
      GetDItem(MailDialog, CashTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      SetIText(ITEMHDL, TEMP);
      NumSpecs(Number, LEN, TEMP);
      GetDItem(MailDialog, NUMITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      SetIText(ITEMHDL, TEMP);
      if Fall then begin
        GetDItem(MailDialog, FALLITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      end else begin
        GetDItem(MailDialog, SPRINGITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      end; (* If *)
      SetCtlVal(Pointer(ITEMHDL), 1);
    end; (* With *)
end; (* GetOldVals *)

procedure AddMail (var Mails:MailType; Editing:boolean; Number:longint);

var  MAILDIALOG : DialogPtr;
    ITEM, DUMMYTYPE, VAL, I, NUM, J : integer;
    DUMMYRECT : Rect;
    ITEMHDL : Handle;
```

```
    AMT : Str255;

begin (* AddMail *)
    MAILDIALOG := GetNewDialog (257, nil, Pointer(-1));
    if not Editing then begin
        GetDItem(MAILDIALOG, GENITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        SetCtlValue(Pointer(ITEMHDL), 1);
        GetDItem(MAILDIALOG, FALLITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        SetCtlValue(Pointer(ITEMHDL), 1);
    end else begin
        GetOldVals(MAILDIALOG, Mails, Number);
    end; (* If *)
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
        if ITEM in [(CONOFFSET + 1)..(CONOFFSET + NUMCONTACTS)] then begin
            for I := 1 to NUMCONTACTS do begin
                GetDItem(MAILDIALOG, I + CONOFFSET, DUMMYTYPE, ITEMHDL, DUMMYRECT);
                SetCtlValue(Pointer(ITEMHDL), 0);
            end; (* For *)
            GetDItem(MAILDIALOG, ITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
            SetCtlVal(Pointer(ITEMHDL), 1);
        end; (* If *)
        if ITEM in [FALLITEM..SPRINGITEM] then begin
            GetDItem(MAILDIALOG, FALLITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
            SetCtlVal(Pointer(ITEMHDL), 0);
            GetDItem(MAILDIALOG, SPRINGITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
            SetCtlVal(Pointer(ITEMHDL), 0);
            GetDItem(MAILDIALOG, ITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
            SetCtlVal(Pointer(ITEMHDL), 1);
        end; (* If *)
    until ITEM in [OK, Cancel];
    if ITEM = OK then begin
        Mails.Number := Mails.Number + 1;
        GetDItem(MAILDIALOG, NUMITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        NUM := ConvertNum(AMT);
        if NUM = 0 then begin
            NUM := Mails.Number;
        end else begin
            for I := Mails.Number downto NUM + 1 do begin
                Mails.List[I] := Mails.List[I - 1];
            end; (* For *)
        end; (* If *)
        GetDItem(MAILDIALOG, FALLITEM, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        VAL := GetCtlVal(Pointer(ITEMHDL));
        case VAL of
            0 : Mails.List[NUM].Fall := false;
            1 : Mails.List[NUM].Fall := true;
        end; (* Case *)
        GetDItem(MAILDIALOG, ContTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, Mails.List[NUM].Content);
```

```
        Mails.List[NUM].Contact := Cornuel;
        for I := 1 to NUMCONTACTS do begin
           GetDItem(MAILDIALOG, I + CONOFFSET, DUMMYTYPE, ITEMHDL, DUMMYRECT);
           VAL := GetCtlVal(Pointer(ITEMHDL));
           if VAL = 1 then begin
              for J := 1 to I - 1 do begin
                 Mails.List[NUM].Contact := succ(Mails.List[NUM].Contact);
              end; (* For *)
           end; (* If *)
        end; (* For *)
        GetDItem(MAILDIALOG, CashTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        if AMT[1] = '-' then begin
           delete(AMT, 1, 1);
           Mails.List[NUM].Cash := 0 - ConvertNum(AMT);
        end else begin
           Mails.List[NUM].Cash := ConvertNum(AMT);
        end; (* If *)
        GetDItem(MAILDIALOG, YearTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        Mails.List[NUM].Year := ConvertNum(AMT);
     end; (* If *)
     DisposDialog(MAILDIALOG);
end; (* AddMail *)

procedure DelMail (var Mails:MailType);

var  MAILDIALOG : DialogPtr;
     ITEM, DUMMYTYPE, I : integer;
     NUM : longint;
     DUMMYRECT : Rect;
     ITEMHDL : Handle;
     NUMSTR : Str255;

begin (* DelMail *)
     MAILDIALOG := GetNewDialog (256, nil, Pointer(-1));
     repeat
        SystemTask;
        ModalDialog(nil, ITEM);
     until ITEM in [OK, Cancel];
     if (ITEM = OK) then begin
        GetDItem(MAILDIALOG, DelTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, NUMSTR);
        NUM := ConvertNum(NUMSTR);
        if NUM <= Mails.Number then begin
           for I := NUM + 1 to Mails.Number do begin
              Mails.List[I-1] := Mails.List[I];
           end; (* For *)
           Mails.Number := Mails.Number - 1;
        end; (* If *)
     end; (* If *)
     DisposDialog(MAILDIALOG);
```

```
end; (* DelMail *)

procedure EditMail (var Mails:MailType);

var  MAILDIALOG : DialogPtr;
     ITEM, DUMMYTYPE, I : integer;
     NUM : longint;
     DUMMYRECT : Rect;
     ITEMHDL : Handle;
     NUMSTR : Str255;

begin (* EditMail *)
     MAILDIALOG := GetNewDialog (258, nil, Pointer(-1));
     repeat
          SystemTask;
          ModalDialog(nil, ITEM);
     until ITEM in [OK, Cancel];
     if (ITEM = OK) then begin
          GetDItem(MAILDIALOG, EdTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
          GetIText(ITEMHDL, NUMSTR);
          NUM := ConvertNum(NUMSTR);
          AddMail(Mails, true, NUM);
     end; (* If *)
     DisposDialog(MAILDIALOG);
end; (* EditMail *)

procedure SaveMails (var Mails:MailType);

var  RECLEN : longint;
     REFNUM, I, NUM : integer;
     ERR : OSErr;
     TITLE : Str255;
     MAIL : MailRec;

begin (* SaveMails *)
     RECLEN := SizeOf(Mails.Number);
     ERR := FSDelete(FileName, 0);
     ERR := Create(FileName, 0, '????', 'MAIL');
     ERR := FSOpen(FileName, 0, REFNUM);
     NUM := Mails.Number;
     ERR := FSWrite(REFNUM, RECLEN, @NUM);
     RECLEN := SizeOf(MailRec);
     for I := 1 to Mails.Number do begin
          MAIL := Mails.List[I];
          ERR := FSWrite(REFNUM, RECLEN, @MAIL);
     end; (* For *)
     ERR := FSClose(REFNUM);
end; (* SaveMails *)

procedure DoCommand(mResult:longint; var Done:boolean; var Mails:MailType);

var  theItem : integer;
```

```
begin (* DoCommand *)
    Done := false;
    theItem := LoWord(mresult);
    case theItem of

        AddItem : AddMail(Mails, false, 0);
        DeleteItem : DelMail(Mails);
        EditItem : EditMail(Mails);
        SaveItem : SaveMails(Mails);
        QuitItem : Done := true;
        ListItem : DisplayMails(Mails, TextWindow);
    end; (* Case *)
    if not Done then begin
        HiliteMenu(0);
    end; (* If *)
end; (* DoCommand *)

begin (* Main *)
    Initialize(myMenu, TextWindow, Mails);
    GetMails(Mails);
    Done := false;
    repeat
        SystemTask;
        Temp := GetNextEvent(everyEvent, myEvent);

        case myEvent.what of

            MouseDown : begin
                Code := FindWindow(myEvent.where, WhichWindow);
                case Code of
                    inMenuBar : DoCommand(MenuSelect(myEvent.where), Done, Mails);
                    inSysWindow : SystemClick(myEvent, WhichWindow);
                end; (* Case *)
            end; (* MouseDown *)

            KeyDown : begin
                Letter := chr(myEvent.message mod 256);
                if BitAnd(myEvent.modifiers, 256) <> 0 then begin
                    DoCommand(MenuKey(Letter), Done, Mails);
                end; (* If *)
            end; (* KeyDown *)

            UpdateEvt : begin
                        BeginUpdate(TextWindow);
                        EndUpdate(TextWindow);
                        DisplayNote(TextWindow);
                    end; (* UpdateEvt *)

        end; (* Case *)
    until Done;
end. (* Main *)
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/MAILR.TEXT

Date: Thursday, September 18, 1986

Time: 11:42:29 PM

Printer: LaserWriter Plus

*{The Would-Be Gentleman, Faculty Author Development Program at Stanford University.}
*{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86. }
*{Faculty Author Development Team: Carolyn Lougee, Michael Carter, Ed McGuigan, }
*{Steve Fisher, and Tom Maliska. }

SunKing/4.0/Mail.RSRC

Type Menu
 ,256
 Main Menu
  Add Mail /A
  Delete Mail/D
  Edit Mail/E
  Save Data/S
  Quit/Q
  List/L

Type WIND
 ,256 (32)
 Display
 0 0 342 512
 Visible NoGoAway
 1
 0

Type DITL
 ,256 (32)
 4

  BtnItem Enabled
  50 13 70 83
OK

  BtnItem Enabled
  50 300 70 370
Cancel

  EditText Enabled
  10 170 25 230

  StatText Disabled
  10 5 25 160
Number to delete

 ,257 (32)
 22

  BtnItem Enabled
  215 330 235 400

OK

BtnItem Enabled
250 330 270 400
Cancel

EditText Enabled
10 10 110 422

RadioItem Enabled
125 10 140 90
Cornuel

RadioItem Enabled
125 100 140 190
Mazarin

RadioItem Enabled
125 200 140 290
Particelli

RadioItem Enabled
125 300 140 390
Conde

RadioItem Enabled
155 10 170 90
Fouquet

RadioItem Enabled
155 100 170 190
Colbert

RadioItem Enabled
155 200 170 290
Maintenon

RadioItem Enabled
155 300 170 420
Grand Dauphin

RadioItem Enabled
185 10 200 190
The Duke Of Burgundy

RadioItem Enabled
185 300 200 420
Generic

RadioItem Enabled
215 10 230 130
No Protector

StatText Disabled
250 10 265 40
Year

EditText Enabled
250 50 265 120

RadioItem Enabled
250 160 265 215
Fall

RadioItem Enabled
250 245 265 305
Spring

StatText Disabled
280 10 295 50
Cash

EditText Enabled
280 55 295 150

StatText Disabled
280 200 295 295
Mail Number

EditText Enabled
280 300 295 390

,258 (32)
4

BtnItem Enabled
50 13 70 83
OK

BtnItem Enabled
50 300 70 370
Cancel

EditText Enabled
10 170 25 230

StatText Disabled
10 5 25 160
Number to edit

Type DLOG
,256 (32)
50 40 140 472

Visible 1 NoGoAway 0
256

,257 (32)
28 40 333 472
Visible 1 NoGoAway 0
257

,258 (32)
50 40 140 472
Visible 1 NoGoAway 0
258

Type CODE
SunKing/4.0/MailL,0

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/MARRIAGE1.TEXT

Date: Thursday, September 18, 1986

Time: 11:43:41 PM

Printer: LaserWriter Plus

```
{$M+}    {mac code}
{$X-}    {no automatic stack expansion}
{$R-}    {no range checking, paslib is buggy}

{The Would-Be Gentleman, Faculty Author Development Program at Stanford University. }
{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86.                        }
{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan,      }
{Steve Fisher, and Tom Maliska.                                                      }
{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford      }
{Junior University.                                                                  }

program Marriage;
{ACTION: This program creates the Marriage data file for the first generation required by  }
{the simulation.  The Marriage program for the second generation differs only in the       }
{FILENAME constant "Marriage2".                                                            }


uses  {$U-}
    {$U obj/MemTypes}        MemTypes,
    {$U obj/QuickDraw}     QuickDraw,
    {$U obj/OSIntf}          OSIntf,
    {$U obj/ToolIntf}        ToolIntf,
    {$U Obj/PackIntf}        PackIntf,
    {$U Obj/MacPrint}        MacPrint,
    {$U Obj/PasLibIntf}    PasLibIntf;

const       AddItem = 1;
       DeleteItem = 2;
       SaveItem = 3;
       QuitItem = 4;
       NameTextItem = 3;
       DadTextItem = 5;
       AgeTextItem = 7;
       DowryTextItem = 9;
       GroupTextItem = 11;
       DelTextItem = 3;
       FileName = 'Marriage1';  {"Marriage2" for the second generation marriage list filename}
       NumMarrs = 30;
       FontSize = 9;

type MarrRec = record
       Father,
       Name : Str255;
       Age,
       Dowry,
       Group : longint;
       Deleted : boolean;
   end; (* MarrRec *)

   MarrArray = array [1..NumMarrs] of MarrRec;

   MarrType = record
```

```
        List : MarrArray;
        Number,              (* Deleted plus undeleted *)
        RealNum : integer;         (* Only undeleted *)
    end; (* MarrType *)

var  WhichWindow, TextWindow : WindowPtr;
    Done,
    Temp : boolean;
    myEvent : EventRecord;
    Code, VRefNum : integer;
    Letter : char;
    myMenu : MenuHandle;
    Marrs : MarrType;

procedure DebugDelay;

begin (* DebugDelay *)
    repeat
        SystemTask;
    until Button;
end; (* DebugDelay *)

procedure SetUpMenus(var myMenu:MenuHandle);

begin (* SetUpMenus *)
    InitMenus;
    myMenu := GetMenu(256);
    InsertMenu(myMenu, 0);
    DrawMenuBar;
end; (* SetUpMenus *)

procedure SetUpWindow(var TextWindow:WindowPtr);

begin (* SetUpWindow *)
    TextWindow := GetNewWindow(256, nil, Pointer(-1));
    SetPort(TextWindow);
    PLSetWrPort (TextWindow);
end; (* SetUpWindow *)

procedure Initialize(var myMenu:MenuHandle; var TextWindow:WindowPtr; var Marrs:MarrType);

begin (* Initialize *)
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyEvent, 0);
    InitWindows;
    SetUpMenus(myMenu);
    SetUpWindow(TextWindow);
    TEInit;
    InitDialogs(nil);
    InitCursor;
    Marrs.Number := 0;
```

```
    Marrs.RealNum := 0;
end; (* Initialize *)

procedure GetMarrs (var Marrs:MarrType);

var  ERR : OSErr;
    REFNUM, I, NUM : integer;
    STRLEN, LONGLEN, AGE, DOWRY, GROUP : longint;
    NAME, FATHER : Str255;

begin (* GetMarrs *)
    ERR := FSOpen(FileName, 0, REFNUM);
    if ERR = 0 then begin
        LONGLEN := SizeOf(Marrs.RealNum);
        ERR := FSRead(REFNUM, LONGLEN, @NUM);
        Marrs.RealNum := NUM;
        Marrs.Number := Marrs.RealNum;
        STRLEN := SizeOf(Str255);
        LONGLEN := SizeOf(longint);
        for I := 1 to Marrs.Number do begin
            ERR := FSRead(REFNUM, STRLEN, @NAME);
            ERR := FSREAD(REFNUM, STRLEN, @FATHER);
            ERR := FSRead(REFNUM, LONGLEN, @AGE);
            ERR := FSRead(REFNUM, LONGLEN, @DOWRY);
            ERR := FSRead(REFNUM, LONGLEN, @GROUP);
            Marrs.List[I].Name := NAME;
            Marrs.List[I].Father := FATHER;
            Marrs.List[I].Age := AGE;
            Marrs.List[I].Dowry := DOWRY;
            Marrs.List[I].Group := GROUP;
            Marrs.List[I].Deleted := false;
        end; (* For *)
        ERR := FSClose(REFNUM);
    end; (* If *)
end; (* GetMarrs *)

procedure DisplayMarrs (Marrs:MarrType; TextWindow:WindowPtr);

var  I : integer;
    DISPRECT : Rect;

begin (* DisplayMarrs *)
    SetPort(TextWindow);
    PLSetWrPort (TextWindow);
    SetRect(DISPRECT, 0, 0, 512, 342);
    EraseRect(DISPRECT);
    TextFont(Geneva);
    TextSize(FontSize);
    MoveTo(0,30);
    for I := 1 to Marrs.Number do begin
        with Marrs.List[I] do begin
            if not Marrs.List[I].Deleted then begin
```

```
                writeln('  ',I:0,'      ', Name, Age, Dowry, Group, Father);
            end; (* If *)
        end; (* With *)
    end; (* For *)
    TextFont(0);
    TextSize(0);
end; (* DisplayMarrs *)

function PowerOfTen(Num : integer) : longint;

var  I, TEMP : longint;

begin (* PowerOfTen *)
    TEMP := 1;
    for I := 1 to Num do begin
        TEMP := TEMP * 10;
    end; (* For *)
    PowerOfTen := TEMP;
end; (* PowerOfTen *)

function ConvertNum(StrNum : Str255) : longint;

var  I, TEMP : longint;

begin (* ConvertNum *)
    TEMP := 0;
    if Length(StrNum) <> 0 then begin
        for I := 1 to Length(StrNum) do begin
            TEMP := TEMP + (ord(StrNum[I]) - ord('0')) * PowerOfTen(Length(StrNum) - I);
        end (* For *)
    end; (* If *)
    ConvertNum := TEMP;
end; (* ConvertNum *)

procedure AddMarrs (var Marrs:MarrType);

var  MarrDIALOG : DialogPtr;
    ITEM, DUMMYTYPE : integer;
    DUMMYRECT : Rect;
    ITEMHDL : Handle;
    AMT : Str255;

begin (* AddMarrs *)
    MarrDIALOG := GetNewDialog (257, nil, Pointer(-1));
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
    until ITEM in [OK, Cancel];
    if ITEM = OK then begin
        Marrs.Number := Marrs.Number + 1;
        Marrs.RealNum := Marrs.RealNum + 1;
        GetDItem(MarrDIALOG, NameTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
```

```
        GetIText(ITEMHDL, Marrs.List[Marrs.Number].Name);
        GetDItem(MarrDIALOG, DadTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, Marrs.List[Marrs.Number].Father);
        GetDItem(MarrDIALOG, AgeTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        Marrs.List[Marrs.Number].Age := ConvertNum(AMT);
        GetDItem(MarrDIALOG, DowryTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        Marrs.List[Marrs.Number].Dowry := ConvertNum(AMT);
        GetDItem(MarrDIALOG, GroupTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        Marrs.List[Marrs.Number].Group := ConvertNum(AMT);
        Marrs.List[Marrs.Number].Deleted := false;
    end; (* If *)
    DisposDialog(MarrDIALOG);
end; (* AddMarrs *)

procedure DelMarrs (var Marrs:MarrType);

var  MARRDIALOG : DialogPtr;
    ITEM, DUMMYTYPE : integer;
    NUM : longint;
    DUMMYRECT : Rect;
    ITEMHDL : Handle;
    NUMSTR : Str255;

begin (* DelMarrs *)
    MARRDIALOG := GetNewDialog (256, nil, Pointer(-1));
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
    until ITEM in [OK, Cancel];
    if (ITEM = OK) then begin
        Marrs.RealNum := Marrs.RealNum - 1;
        GetDItem(MARRDIALOG, DelTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, NUMSTR);
        NUM := ConvertNum(NUMSTR);
        if NUM <= NumMarrs then begin
            Marrs.List[NUM].Deleted := true;
        end; (* If *)
    end; (* If *)
    DisposDialog(MARRDIALOG);
end; (* DelMarrs *)

procedure SaveMarrs (Marrs:MarrType);

var  STRLEN, LONGLEN, AGE, DOWRY, GROUP : longint;
    REFNUM, I, NUM : integer;
    ERR : OSErr;
    NAME, FATHER : Str255;

begin (* SaveMarrs *)
```

```
    LONGLEN := SizeOf(Marrs.RealNum);
    ERR := FSDelete(FileName, 0);
    ERR := Create(FileName, 0, '????', 'MAR1');
    ERR := FSOpen(FileName, 0, REFNUM);
    NUM := Marrs.RealNum;
    ERR := FSWrite(REFNUM, LONGLEN, @NUM);
    STRLEN := SizeOf(Str255);
    LONGLEN := SizeOf(longint);
    for I := 1 to Marrs.Number do begin
        if not Marrs.List[I]. Deleted then begin
            NAME := Marrs.List[I].Name;
            FATHER := Marrs.List[I].Father;
            AGE := Marrs.List[I].Age;
            DOWRY := Marrs.List[I].Dowry;
            GROUP := Marrs.List[I].GROUP;
            ERR := FSWrite(REFNUM, STRLEN, @NAME);
            ERR := FSWrite(REFNUM, STRLEN, @FATHER);
            ERR := FSWrite(REFNUM, LONGLEN, @AGE);
            ERR := FSWrite(REFNUM, LONGLEN, @DOWRY);
            ERR := FSWrite(REFNUM, LONGLEN, @GROUP);
        end; (* If *)
    end; (* For *)
    ERR := FSClose(REFNUM);
end; (* SaveMarrs *)

procedure DoCommand(mResult:longint; var Done:boolean; var Marrs:MarrType);

var  theItem : integer;

begin (* DoCommand *)
    Done := false;
    theItem := LoWord(mresult);
    case theItem of
        AddItem : AddMarrs(Marrs);
        DeleteItem : DelMarrs(Marrs);
        SaveItem : SaveMarrs(Marrs);
        QuitItem : Done := true;
    end; (* Case *)
    if not Done then begin
        HiliteMenu(0);
    end; (* If *)
end; (* DoCommand *)

begin (* Main *)
    Initialize(myMenu, TextWindow, Marrs);
    GetMarrs(Marrs);
    Done := false;
    repeat
        SystemTask;
        Temp := GetNextEvent(everyEvent, myEvent);

        case myEvent.what of
```

```
        MouseDown : begin
            Code := FindWindow(myEvent.where, WhichWindow);
            case Code of
                inMenuBar : DoCommand(MenuSelect(myEvent.where), Done, Marrs);
                inSysWindow : SystemClick(myEvent, WhichWindow);
            end; (* Case *)
        end; (* MouseDown *)

        KeyDown : begin
            Letter := chr(myEvent.message mod 256);
            if BitAnd(myEvent.modifiers, 256) <> 0 then begin
                DoCommand(MenuKey(Letter), Done, Marrs);
            end; (* If *)
        end; (* KeyDown *)

        UpdateEvt : begin
                BeginUpdate(TextWindow);
                EndUpdate(TextWindow);
                DisplayMarrs(Marrs, TextWindow);
            end; (* UpdateEvt *)

    end; (* Case *)
  until Done;
end. (* Main *)
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/MARRIAGE1R.TEXT

Date: Thursday, September 18, 1986

Time: 11:45:02 PM

Printer: LaserWriter Plus

*{The Would-Be Gentleman, Faculty Author Development Program at Stanford University.}
*{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86. }
*{Faculty Author Development Team: Carolyn Lougee, Michael Carter, Ed McGuigan, }
*{Steve Fisher, and Tom Maliska. }

SunKing/4.0/Marriage1.RSRC

Type Menu
,256
Main Menu
Add Bride /A
Delete Bride/D
Save Data/S
Quit/Q

Type WIND
,256 (4)
Display
0 0 342 512
Visible NoGoAway
1
0

Type DITL
,256
4

BtnItem Enabled
50 13 70 83
OK

BtnItem Enabled
50 300 70 370
Cancel

EditText Enabled
10 170 25 230

StatText Disabled
10 5 25 160
Number to delete

,257
12

BtnItem Enabled
160 13 180 83
OK

```
    BtnItem Enabled
    160 300 180 370
Cancel

    EditText Enabled
    10 60 25 390

    StatText Disabled
    10 5 25 50
Name

    EditText Enabled
    35 60 65 390

    StatText Disabled
    35 5 65 50
Father is...

    EditText Enabled
    75 60 90 120

    StatText Disabled
    75 5 90 50
Age

    EditText Enabled
    100 60 115 150

    StatText Disabled
    100 5 115 50
Dowry

    EditText Enabled
    125 60 140 120

    StatText Disabled
    125 5 140 50
Group

Type DLOG
 ,256
 50 40 140 472
 Visible 1 NoGoAway 0
 256

 ,257
 50 40 240 472
```

Visible 1 NoGoAway 0
257

Type CODE
SunKing/4.0/Marriage1L,0

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/MARRIAGE2.TEXT

Date: Thursday, September 18, 1986

Time: 11:46:01 PM

Printer: LaserWriter Plus

```
{$M+}    {mac code}
{$X-}    {no automatic stack expansion}
{$R-}    {no range checking, paslib is buggy}

{The Would-Be Gentleman, Faculty Author Development Program at Stanford University. }
{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86.                        }
{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan,      }
{Steve Fisher, and Tom Maliska.                                                      }
{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford      }
{Junior University.                                                                  }

program Marriage;
{ACTION: This program creates the Marriage data file for the second generation required  }
{by the simulation.  The Marriage program for the first generation differs only in the   }
{FILENAME constant "Marriage1".                                                          }

uses  {$U-}
    {$U obj/MemTypes}        MemTypes,
    {$U obj/QuickDraw}     QuickDraw,
    {$U obj/OSIntf}          OSIntf,
    {$U obj/ToolIntf}        ToolIntf,
    {$U Obj/PackIntf}        PackIntf,
    {$U Obj/MacPrint}        MacPrint,
    {$U Obj/PasLibIntf}    PasLibIntf;

const       AddItem = 1;
       DeleteItem = 2;
       SaveItem = 3;
       QuitItem = 4;
       NameTextItem = 3;
       DadTextItem = 5;
       AgeTextItem = 7;
       DowryTextItem = 9;
       GroupTextItem = 11;
       DelTextItem = 3;
       FileName = 'Marriage2';  {'Marriage1' for the first generation marriage list filename}
       NumMarrs = 30;
       FontSize = 9;

type MarrRec = record
       Father,
       Name : Str255;
       Age,
       Dowry,
       Group : longint;
       Deleted : boolean;
   end; (* MarrRec *)

   MarrArray = array [1..NumMarrs] of MarrRec;

   MarrType = record
      List : MarrArray;
```

```
        Number,              (* Deleted plus undeleted *)
        RealNum : integer;       (* Only undeleted *)
    end; (* MarrType *)

var  WhichWindow, TextWindow : WindowPtr;
    Done,
    Temp : boolean;
    myEvent : EventRecord;
    Code, VRefNum : integer;
    Letter : char;
    myMenu : MenuHandle;
    Marrs : MarrType;

procedure DebugDelay;

begin (* DebugDelay *)
    repeat
        SystemTask;
    until Button;
end; (* DebugDelay *)

procedure SetUpMenus(var myMenu:MenuHandle);

begin (* SetUpMenus *)
    InitMenus;
    myMenu := GetMenu(256);
    InsertMenu(myMenu, 0);
    DrawMenuBar;
end; (* SetUpMenus *)

procedure SetUpWindow(var TextWindow:WindowPtr);

begin (* SetUpWindow *)
    TextWindow := GetNewWindow(256, nil, Pointer(-1));
    SetPort(TextWindow);
    PLSetWrPort(TextWindow);
end; (* SetUpWindow *)

procedure Initialize(var myMenu:MenuHandle; var TextWindow:WindowPtr; var Marrs:MarrType);

begin (* Initialize *)
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyEvent, 0);
    InitWindows;
    SetUpMenus(myMenu);
    SetUpWindow(TextWindow);
    TEInit;
    InitDialogs(nil);
    InitCursor;
    Marrs.Number := 0;
    Marrs.RealNum := 0;
```

```
end; (* Initialize *)

procedure GetMarrs (var Marrs:MarrType);

var  ERR : OSErr;
     REFNUM, I, NUM : integer;
     STRLEN, LONGLEN, AGE, DOWRY, GROUP : longint;
     NAME, FATHER : Str255;

begin (* GetMarrs *)
     ERR := FSOpen(FileName, 0, REFNUM);
     if ERR = 0 then begin
          LONGLEN := SizeOf(Marrs.RealNum);
          ERR := FSRead(REFNUM, LONGLEN, @NUM);
          Marrs.RealNum := NUM;
          Marrs.Number := Marrs.RealNum;
          STRLEN := SizeOf(Str255);
          LONGLEN := SizeOf(longint);
          for I := 1 to Marrs.Number do begin
               ERR := FSRead(REFNUM, STRLEN, @NAME);
               ERR := FSREAD(REFNUM, STRLEN, @FATHER);
               ERR := FSRead(REFNUM, LONGLEN, @AGE);
               ERR := FSRead(REFNUM, LONGLEN, @DOWRY);
               ERR := FSRead(REFNUM, LONGLEN, @GROUP);
               Marrs.List[I].Name := NAME;
               Marrs.List[I].Father := FATHER;
               Marrs.List[I].Age := AGE;
               Marrs.List[I].Dowry := DOWRY;
               Marrs.List[I].Group := GROUP;
               Marrs.List[I].Deleted := false;
          end; (* For *)
          ERR := FSClose(REFNUM);
     end; (* If *)
end; (* GetMarrs *)

procedure DisplayMarrs (Marrs:MarrType; TextWindow:WindowPtr);

var  I : integer;
     DISPRECT : Rect;

begin (* DisplayMarrs *)
     SetPort(TextWindow);
     PLSetWrPort(TextWindow);
     SetRect(DISPRECT, 0, 0, 512, 342);
     EraseRect(DISPRECT);
     TextFont(Geneva);
     TextSize(FontSize);
     MoveTo(0,30);
     for I := 1 to Marrs.Number do begin
          with Marrs.List[I] do begin
               if not Marrs.List[I].Deleted then begin
                    writeln('  ',I:0,'      ', Name, Age, Dowry, Group, Father);
```

```
            end; (* If *)
        end; (* With *)
    end; (* For *)
    TextFont(0);
    TextSize(0);
end; (* DisplayMarrs *)

function PowerOfTen(Num : integer) : longint;

var  I, TEMP : longint;

begin (* PowerOfTen *)
    TEMP := 1;
    for I := 1 to Num do begin
        TEMP := TEMP * 10;
    end; (* For *)
    PowerOfTen := TEMP;
end; (* PowerOfTen *)

function ConvertNum(StrNum : Str255) : longint;

var  I, TEMP : longint;

begin (* ConvertNum *)
    TEMP := 0;
    if Length(StrNum) <> 0 then begin
        for I := 1 to Length(StrNum) do begin
            TEMP := TEMP + (ord(StrNum[I]) - ord('0')) * PowerOfTen(Length(StrNum) - I);
        end (* For *)
    end; (* If *)
    ConvertNum := TEMP;
end; (* ConvertNum *)

procedure AddMarrs (var Marrs:MarrType);

var  MarrDIALOG : DialogPtr;
    ITEM, DUMMYTYPE : integer;
    DUMMYRECT : Rect;
    ITEMHDL : Handle;
    AMT : Str255;

begin (* AddMarrs *)
    MarrDIALOG := GetNewDialog (257, nil, Pointer(-1));
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
    until ITEM in [OK, Cancel];
    if ITEM = OK then begin
        Marrs.Number := Marrs.Number + 1;
        Marrs.RealNum := Marrs.RealNum + 1;
        GetDItem(MarrDIALOG, NameTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, Marrs.List[Marrs.Number].Name);
```

```
        GetDItem(MarrDIALOG, DadTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, Marrs.List[Marrs.Number].Father);
        GetDItem(MarrDIALOG, AgeTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        Marrs.List[Marrs.Number].Age := ConvertNum(AMT);
        GetDItem(MarrDIALOG, DowryTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        Marrs.List[Marrs.Number].Dowry := ConvertNum(AMT);
        GetDItem(MarrDIALOG, GroupTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, AMT);
        Marrs.List[Marrs.Number].Group := ConvertNum(AMT);
        Marrs.List[Marrs.Number].Deleted := false;
    end; (* If *)
    DisposDialog(MarrDIALOG);
end; (* AddMarrs *)

procedure DelMarrs (var Marrs:MarrType);

var  MARRDIALOG : DialogPtr;
    ITEM, DUMMYTYPE : integer;
    NUM : longint;
    DUMMYRECT : Rect;
    ITEMHDL : Handle;
    NUMSTR : Str255;

begin (* DelMarrs *)
    MARRDIALOG := GetNewDialog (256, nil, Pointer(-1));
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
    until ITEM in [OK, Cancel];
    if (ITEM = OK) then begin
        Marrs.RealNum := Marrs.RealNum - 1;
        GetDItem(MarrDIALOG, DelTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
        GetIText(ITEMHDL, NUMSTR);
        NUM := ConvertNum(NUMSTR);
        if NUM <= NumMarrs then begin
            Marrs.List[NUM].Deleted := true;
        end; (* If *)
    end; (* If *)
    DisposDialog(MARRDIALOG);
end; (* DelMarrs *)

procedure SaveMarrs (Marrs:MarrType);

var  STRLEN, LONGLEN, AGE, DOWRY, GROUP : longint;
    REFNUM, I, NUM : integer;
    ERR : OSErr;
    NAME, FATHER : Str255;

begin (* SaveMarrs *)
    LONGLEN := SizeOf(Marrs.RealNum);
```

```
    ERR := FSDelete(FileName, 0);
    ERR := Create(FileName, 0, '????', 'MAR1');
    ERR := FSOpen(FileName, 0, REFNUM);
    NUM := Marrs.RealNum;
    ERR := FSWrite(REFNUM, LONGLEN, @NUM);
    STRLEN := SizeOf(Str255);
    LONGLEN := SizeOf(longint);
    for I := 1 to Marrs.Number do begin
        if not Marrs.List[I]. Deleted then begin
            NAME := Marrs.List[I].Name;
            FATHER := Marrs.List[I].Father;
            AGE := Marrs.List[I].Age;
            DOWRY := Marrs.List[I].Dowry;
            GROUP := Marrs.List[I].GROUP;
            ERR := FSWrite(REFNUM, STRLEN, @NAME);
            ERR := FSWrite(REFNUM, STRLEN, @FATHER);
            ERR := FSWrite(REFNUM, LONGLEN, @AGE);
            ERR := FSWrite(REFNUM, LONGLEN, @DOWRY);
            ERR := FSWrite(REFNUM, LONGLEN, @GROUP);
        end; (* If *)
    end; (* For *)
    ERR := FSClose(REFNUM);
end; (* SaveMarrs *)

procedure DoCommand(mResult:longint; var Done:boolean; var Marrs:MarrType);

var  theItem : integer;

begin (* DoCommand *)
    Done := false;
    theItem := LoWord(mresult);
    case theItem of
        AddItem : AddMarrs(Marrs);
        DeleteItem : DelMarrs(Marrs);
        SaveItem : SaveMarrs(Marrs);
        QuitItem : Done := true;
    end; (* Case *)
    if not Done then begin
        HiliteMenu(0);
    end; (* If *)
end; (* DoCommand *)

begin (* Main *)
    Initialize(myMenu, TextWindow, Marrs);
    GetMarrs(Marrs);
    Done := false;
    repeat
        SystemTask;
        Temp := GetNextEvent(everyEvent, myEvent);

        case myEvent.what of
```

```
        MouseDown : begin
            Code := FindWindow(myEvent.where, WhichWindow);
            case Code of
                inMenuBar : DoCommand(MenuSelect(myEvent.where), Done, Marrs);
                inSysWindow : SystemClick(myEvent, WhichWindow);
            end; (* Case *)
        end; (* MouseDown *)

        KeyDown : begin
            Letter := chr(myEvent.message mod 256);
            if BitAnd(myEvent.modifiers, 256) <> 0 then begin
                DoCommand(MenuKey(Letter), Done, Marrs);
            end; (* If *)
        end; (* KeyDown *)

        UpdateEvt : begin
                BeginUpdate(TextWindow);
                EndUpdate(TextWindow);
                DisplayMarrs(Marrs, TextWindow);
            end; (* UpdateEvt *)

    end; (* Case *)
  until Done;
end. (* Main *)
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/MARRIAGE2R.TEXT

Date: Thursday, September 18, 1986

Time: 11:47:14 PM

Printer: LaserWriter Plus

SunKing/4.0/Marriage2.RSRC

```
Type Menu
 ,256
 Main Menu
   Add Bride/A
   Delete Bride/D
   Save Data/S
   Quit/Q

Type WIND
 ,256 (4)
 Display
 0 0 342 512
 Visible NoGoAway
 1
 0

Type DITL
 ,256
 4

   BtnItem Enabled
   50 13 70 83
OK

   BtnItem Enabled
   50 300 70 370
Cancel

   EditText Enabled
   10 170 25 230


   StatText Disabled
   10 5 25 160
Number to delete

 ,257
 12

   BtnItem Enabled
   160 13 180 83
OK
```

BtnItem Enabled
160 300 180 370
Cancel

EditText Enabled
10 60 25 390

StatText Disabled
10 5 25 50
Name

EditText Enabled
35 60 65 390

StatText Disabled
35 5 65 50
Father is...

EditText Enabled
75 60 90 120

StatText Disabled
75 5 90 50
Age

EditText Enabled
100 60 115 150

StatText Disabled
100 5 115 50
Dowry

EditText Enabled
125 60 140 120

StatText Disabled
125 5 140 50
Group

Type DLOG
,256
50 40 140 472
Visible 1 NoGoAway 0
256

,257
50 40 240 472

Visible 1 NoGoAway 0
257

Type CODE
SunKing/4.0/Marriage2L,0

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/OFFICE.TEXT

Date: Thursday, September 18, 1986

Time: 11:48:24 PM

Printer: LaserWriter Plus

```
{$M+}    {mac code}
{$X-}    {no automatic stack expansion}
{$R-}    {no range checking, paslib is buggy}

{The Would-Be Gentleman, Faculty Author Development Program at Stanford University. }
{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86.                        }
{Faculty Author Development Team:  Carolyn Lougee, Michael Carter, Ed McGuigan,      }
{Steve Fisher, and Tom Maliska.                                                      }
{Copyright 1986 Carolyn Lougee and the Board of Trustees of the Leland Stanford      }
{Junior University.                                                                  }

program Office;
{ACTION: This program creates the Office list data file required by the simulation.  }

uses  {$U-}
    {$U obj/MemTypes}        MemTypes,
    {$U obj/QuickDraw}     QuickDraw,
    {$U obj/OSIntf}          OSIntf,
    {$U obj/ToolIntf}        ToolIntf,
    {$U Obj/PackIntf}        PackIntf,
    {$U Obj/MacPrint}        MacPrint,
    {$U Obj/PasLibIntf}    PasLibIntf;

const       AddItem = 1;
       DeleteItem = 2;
       SaveItem = 3;
       QuitItem = 4;
       DelTextItem = 3;
       AddTextItem = 3;
       CostTextItem = 5;
       PresTextItem = 7;
       DriveNum = 0;
       FileName = 'Office.Dat';
       NumOffices = 100;
       FontSize = 9;
       NobItem = 9;
       TitleItem = 10;

type OfficeRec = record
       Title : Str255;
       Cost,
       Prestige : longint;
       Nobility,
       TitAndNob,
       Deleted : boolean;
   end; (* OfficeRec *)

   OfficeArray = array [1..NumOffices] of OfficeRec;

   OfficeType = record
      List : OfficeArray;
       Number,              (* Deleted plus undeleted *)
```

```
        RealNum : integer;          (* Only undeleted *)
    end; (* OfficeType *)

var  WhichWindow, TextWindow : WindowPtr;
    Done,
    Temp : boolean;
    myEvent : EventRecord;
    Code, VRefNum : integer;
    Letter : char;
    myMenu : MenuHandle;
    Offices : OfficeType;

procedure DebugDelay;

begin (* DebugDelay *)
    repeat
        SystemTask;
    until Button;
end; (* DebugDelay *)

procedure SetUpMenus(var myMenu:MenuHandle);

begin (* SetUpMenus *)
    InitMenus;
    myMenu := GetMenu(256);
    InsertMenu(myMenu, 0);
    DrawMenuBar;
end; (* SetUpMenus *)

procedure SetUpWindow(var TextWindow:WindowPtr);

begin (* SetUpWindow *)
    TextWindow := GetNewWindow(256, nil, Pointer(-1));
    SetPort(TextWindow);
    PLSetWrPort (TextWindow);
end; (* SetUpWindow *)

procedure Initialize(var myMenu:MenuHandle; var TextWindow:WindowPtr; var Offices:OfficeType);

begin (* Initialize *)
    InitGraf(@thePort);
    InitFonts;
    FlushEvents(everyEvent, 0);
    InitWindows;
    SetUpMenus(myMenu);
    SetUpWindow(TextWindow);
    TEInit;
    InitDialogs(nil);
    InitCursor;
    Offices.Number := 0;
    Offices.RealNum := 0;
end; (* Initialize *)
```

```
procedure GetOffices (var Offices:OfficeType);

var  ERR : OSErr;
     REFNUM, I, NUM : integer;
     RECLEN, COSTLEN, COST, PRESTIGE, BOOLLEN : longint;
     TITLE : Str255;
     TITANDNOB,
     NOBILITY : boolean;

begin (* GetOffices *)
     ERR := FSOpen(FileName, 0, REFNUM);
     if ERR = 0 then begin
          RECLEN := SizeOf(Offices.RealNum);
          ERR := FSRead(REFNUM, RECLEN, @NUM);
          Offices.RealNum := NUM;
          Offices.Number := Offices.RealNum;
          RECLEN := SizeOf(Str255);
          COSTLEN := SizeOf(COST);
          BOOLLEN := SizeOf(boolean);
          for I := 1 to Offices.Number do begin
               ERR := FSRead(REFNUM, RECLEN, @TITLE);
               ERR := FSREAD(REFNUM, COSTLEN, @COST);
               ERR := FSRead(REFNUM, COSTLEN, @PRESTIGE);
               ERR := FSRead(REFNUM, BOOLLEN, @TITANDNOB);
               ERR := FSRead(REFNUM, BOOLLEN, @NOBILITY);
               Offices.List[I].Title := TITLE;
               Offices.List[I].Cost := COST;
               Offices.List[I].Prestige := PRESTIGE;
               Offices.List[I].Nobility := NOBILITY;
               Offices.List[I].TitAndNob := TITANDNOB;
               Offices.List[I].Deleted := false;
          end; (* For *)
          ERR := FSClose(REFNUM);
     end; (* If *)
end; (* GetOffices *)

procedure DisplayOffices (Offices:OfficeType; TextWindow:WindowPtr);

var  I : integer;
     DISPRECT : Rect;

begin (* DisplayOffices *)
     SetPort(TextWindow);
     PLSetWrPort(TextWindow);
     SetRect(DISPRECT, 0, 0, 512, 342);
     EraseRect(DISPRECT);
     TextFont(Geneva);
     TextSize(FontSize);
     MoveTo(0,30);
     for I := 1 to Offices.Number do begin
          if not Offices.List[I].Deleted then begin
```

```
            writeln('  ',I:0,'     ',Offices.List[I].Title,Offices.List[I].Cost,' ',Offices.List[I].Prestige,'  ',Offices.List
        end; (* If *)
    end; (* For *)
    TextFont(0);
    TextSize(0);
end; (* DisplayOffices *)

function PowerOfTen(Num : integer) : longint;

var  I, TEMP : longint;

begin (* PowerOfTen *)
    TEMP := 1;
    for I := 1 to Num do begin
        TEMP := TEMP * 10;
    end; (* For *)
    PowerOfTen := TEMP;
end; (* PowerOfTen *)

function ConvertNum(StrNum : Str255) : longint;

var  I, TEMP : longint;

begin (* ConvertNum *)
    TEMP := 0;
    if Length(StrNum) <> 0 then begin
        for I := 1 to Length(StrNum) do begin
            TEMP := TEMP + (ord(StrNum[I]) - ord('0')) * PowerOfTen(Length(StrNum) - I);
        end (* For *)
    end; (* If *)
    ConvertNum := TEMP;
end; (* ConvertNum *)

procedure AddOffices (var Offices:OfficeType);

var  OFFICEDIALOG : DialogPtr;
    ITEM, DUMMYTYPE, VAL : integer;
    DUMMYRECT : Rect;
    ITEMHDL, ITEM1HDL, ITEM2HDL : Handle;
    AMT : Str255;

begin (* AddOffices *)
    OFFICEDIALOG := GetNewDialog (257, nil, Pointer(-1));
    GetDItem(OFFICEDIALOG, NOBITEM, DUMMYTYPE, ITEM1HDL, DUMMYRECT);
    SetCtlValue(Pointer(ITEM1HDL), 0);
    GetDItem(OFFICEDIALOG, TITLEITEM, DUMMYTYPE, ITEM2HDL, DUMMYRECT);
    SetCtlValue(Pointer(ITEM2HDL), 0);
    repeat
        SystemTask;
        ModalDialog(nil, ITEM);
        if (ITEM = NOBITEM) or (ITEM = TITLEITEM) then begin
            case ITEM of
```

```
            NOBITEM : ITEMHDL := ITEM1HDL;
            TITLEITEM : ITEMHDL := ITEM2HDL;
         end; (* Case *)
         VAL := GetCtlValue(Pointer(ITEMHDL));
         if VAL = 0 then begin
            SetCtlValue(Pointer(ITEMHDL), 1);
         end else begin
            SetCtlValue(Pointer(ITEMHDL), 0);
         end; (* If *)
      end; (* If *)
   until ITEM in [OK, Cancel];
   if ITEM = OK then begin
      Offices.Number := Offices.Number + 1;
      Offices.RealNum := Offices.RealNum + 1;
      VAL := GetCtlValue(Pointer(ITEM1HDL));
      if VAL = 0 then begin
         Offices.List[Offices.Number].Nobility := false;
      end else begin
         Offices.List[Offices.Number].Nobility := true;
      end; (* If *)
      VAL := GetCtlValue(Pointer(ITEM2HDL));
      if VAL = 0 then begin
         Offices.List[Offices.Number].TitAndNob := false;
      end else begin
         Offices.List[Offices.Number].TitAndNob := true;
      end; (* If *)
      GetDItem(OFFICEDIALOG, AddTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      GetIText(ITEMHDL, Offices.List[Offices.Number].Title);
      GetDItem(OFFICEDIALOG, CostTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      GetIText(ITEMHDL, AMT);
      Offices.List[Offices.Number].Cost := ConvertNum(AMT);
      GetDItem(OFFICEDIALOG, PresTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      GetIText(ITEMHDL, AMT);
      Offices.List[Offices.Number].Prestige := ConvertNum(AMT);
      Offices.List[Offices.Number].Deleted := false;
   end; (* If *)
   DisposDialog(OFFICEDIALOG);
end; (* AddOffices *)

procedure DelOffices (var Offices:OfficeType);

var  OFFICEDIALOG : DialogPtr;
   ITEM, DUMMYTYPE : integer;
   NUM : longint;
   DUMMYRECT : Rect;
   ITEMHDL : Handle;
   NUMSTR : Str255;

begin (* DelOffices *)
   OFFICEDIALOG := GetNewDialog (256, nil, Pointer(-1));
   repeat
      SystemTask;
```

```
      ModalDialog(nil, ITEM);
    until ITEM in [OK, Cancel];
    if (ITEM = OK) then begin
      Offices.RealNum := Offices.RealNum - 1;
      GetDItem(OFFICEDIALOG, AddTextItem, DUMMYTYPE, ITEMHDL, DUMMYRECT);
      GetIText(ITEMHDL, NUMSTR);
      NUM := ConvertNum(NUMSTR);
      if NUM <= NumOffices then begin
        Offices.List[NUM].Deleted := true;
      end; (* If *)
    end; (* If *)
    DisposDialog(OFFICEDIALOG);
end; (* DelOffices *)

procedure SaveOffices (Offices:OfficeType);

var  RECLEN, COSTLEN, BOOLLEN, COST, PRESTIGE : longint;
    REFNUM, I, NUM : integer;
    ERR : OSErr;
    TITLE : Str255;
    TITANDNOB,
    NOBILITY : boolean;

begin (* SaveOffices *)
    RECLEN := SizeOf(Offices.RealNum);
    ERR := FSDelete(FileName, 0);
     ERR := Create(FileName, 0, '????', 'LEAS');
    ERR := FSOpen(FileName, 0, REFNUM);
    NUM := Offices.RealNum;
    ERR := FSWrite(REFNUM, RECLEN, @NUM);
    RECLEN := SizeOf(TITLE);
    COSTLEN := SizeOf(COST);
    BOOLLEN := SizeOf(boolean);
    for I := 1 to Offices.Number do begin
      if not Offices.List[I]. Deleted then begin
        TITLE := Offices.List[I].Title;
        COST := Offices.List[I].Cost;
        PRESTIGE := Offices.List[I].Prestige;
        NOBILITY := Offices.List[I].Nobility;
        TITANDNOB := Offices.List[I].TitAndNob;
        ERR := FSWrite(REFNUM, RECLEN, @TITLE);
        ERR := FSWrite(REFNUM, COSTLEN, @COST);
        ERR := FSWrite(REFNUM, COSTLEN, @PRESTIGE);
        ERR := FSWrite(REFNUM, BOOLLEN, @TITANDNOB);
        ERR := FSWrite(REFNUM, BOOLLEN, @NOBILITY);
      end; (* If *)
    end; (* For *)
    ERR := FSClose(REFNUM);
end; (* SaveOffices *)

procedure DoCommand(mResult:longint; var Done:boolean; var Offices:OfficeType);
```

```
var theItem : integer;

begin (* DoCommand *)
   Done := false;
   theItem := LoWord(mresult);
   case theItem of
      AddItem : AddOffice(Offices);
      DeleteItem : DelOffice(Offices);
      SaveItem : SaveOffices(Offices);
      QuitItem : Done := true;
   end; (* Case *)
   if not Done then begin
      HiliteMenu(0);
   end; (* If *)
end; (* DoCommand *)

begin (* Main *)
   Initialize(myMenu, TextWindow, Offices);
   GetOffices(Offices);
   Done := false;
   repeat
      SystemTask;
      Temp := GetNextEvent(everyEvent, myEvent);

      case myEvent.what of

         MouseDown : begin
            Code := FindWindow(myEvent.where, WhichWindow);
            case Code of
               inMenuBar : DoCommand(MenuSelect(myEvent.where), Done, Offices);
               inSysWindow : SystemClick(myEvent, WhichWindow);
            end; (* Case *)
         end; (* MouseDown *)

         KeyDown : begin
            Letter := chr(myEvent.message mod 256);
            if BitAnd(myEvent.modifiers, 256) <> 0 then begin
               DoCommand(MenuKey(Letter), Done, Offices);
            end; (* If *)
         end; (* KeyDown *)

         UpdateEvt : begin
                  BeginUpdate(TextWindow);
                  EndUpdate(TextWindow);
                  DisplayOffices(Offices, TextWindow);
               end; (* UpdateEvt *)

      end; (* Case *)
   until Done;
end. (* Main *)
```

User: Tom Maliska, FAD Program

Application: Edit

Document: Support Programs:SUNKING/4.0/OFFICER.TEXT

Date: Thursday, September 18, 1986

Time: 11:49:22 PM

Printer: LaserWriter Plus

*{The Would-Be Gentleman, Faculty Author Development Program at Stanford University.}
*{Version 4.1, Steve Fisher 12/20/84 and Tom Maliska, 3/12/86. }
*{Faculty Author Development Team: Carolyn Lougee, Michael Carter, Ed McGuigan, }
*{Steve Fisher, and Tom Maliska. }

SunKing/4.0/Office.RSRC

Type Menu
 ,256
 Main Menu
  Add Office /A
  Delete Office/D
  Save Data/S
  Quit/Q

Type WIND
 ,256 (32)
 Display
 0 0 342 512
 Visible NoGoAway
 1
 0

Type DITL
 ,256 (32)
 4

  BtnItem Enabled
  50 13 70 83
OK

  BtnItem Enabled
  50 300 70 370
Cancel

  EditText Enabled
  10 170 25 230

  StatText Disabled
  10 5 25 160
Number to delete

 ,257 (32)
 10

  BtnItem Enabled
  130 13 150 83
OK

BtnItem Enabled
130 300 150 370
Cancel

EditText Enabled
10 70 40 390

StatText Disabled
10 5 25 65
Office

EditText Enabled
50 70 65 140

StatText Disabled
50 5 65 65
Cost

EditText Enabled
75 70 90 120

StatText Disabled
75 5 90 65
Prestige

ChkItem Enabled
100 5 115 170
Nobility Required

ChkItem Enabled
100 200 115 450
Nobility and Title Required

Type DLOG
,256 (32)
50 40 140 472
Visible 1 NoGoAway 0
256

,257 (32)
50 40 220 472
Visible 1 NoGoAway 0
257

Type CODE
SunKing/4.0/OfficeL,0

User: Tom Maliska, FAD Program

Application: MacDraw 1.9

Document: Cover, User's Manual

Date: Friday, October 3, 1986

Time: 6:09:31 PM

Printer: LaserWriter Plus

The Would-Be Gentleman

A simulation of social mobility set during the life and reign
of Louis XIV of France, 1638 -1715

User's Manual

The Would-Be Gentleman User's Manual
version 4.1 (a)

Created at Stanford University by
the Faculty Author Development Program.

User: Tom Maliska, FAD Program

Application: MacDraw 1.9

Document: Credits/Startup Screen

Date: Friday, September 19, 1986

Time: 1:09:16 AM

Printer: LaserWriter Plus

THE WOULD-BE GENTLEMAN

HISTORY 31S

PROF. CAROLYN LOUGEE

Created by the Faculty Author Development Program, Stanford University

Digitized art for the Would-Be Gentleman. This woodcut is displayed when the simulation is started.

User: Tom Maliska, FAD Program

Application: MacDraw 1.9

Document: Beggar

Date: Friday, September 19, 1986

Time: 12:53:38 AM

Printer: LaserWriter Plus

Beggar with Large Rosary, 1623

Jacques Callot

Digitized art for the Would-Be Gentleman. This woodcut is displayed when a player goes bankrupt.

User: Tom Maliska, FAD Program

Application: MacDraw 1.9

Document: Painter and Son

Date: Friday, September 19, 1986

Time: 12:55:23 AM

Printer: LaserWriter Plus

Portrait of the Painter Claude Deruet and His Son, 1632

Jacques Callot

Digitized art for the Would-Be Gentleman. This woodcut is displayed at the birth of the firstborn son, and at the start of play with the credits for the simulation.