#!/usr/bin/env python

# converted to Python file, changed = to == and := to =

# “ then” to “:” commented out end, added changes for math and random libraries

# checked == for floats, convert every else if to elif, add: to else

# **need to handle with and remove begin statements**

# while and for, procedure with def and add return if necessary

# unit MyWW2RulesU;

#\*\*\*\* debug on debugTime check for 0 or > 9 #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# DESIGN PHILOSOPHY: #

# Routines which react to changes in the form will only change other items #

# in the form, they will not update the database. #

# Only when the appropriate button is pressed, eg Assign Targets or Move #

# will the database be updated. #

# This means for example that SetupFirerDetails will not be called when #

# one of the target combo-boxes is changed, only when a new firer is #

# selected #

# #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

interface

Import Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,\

math, random, Menus, StdCtrls, ExtCtrls, #System, #ComCtrls, Grids, ExtDlgs, Mask;

type

TBandName == String[4];

TFormationRecord == class(TObject)

FormationLeadershipDice: Boolean;

end;

WW2Rules == class(TForm)

MainMenu: TMainMenu;

File1: TMenuItem;

OpenDatabase: TMenuItem;

SaveDatabase: TMenuItem;

CloseDatabase: TMenuItem; #Coded#

N1: TMenuItem;

Exit1: TMenuItem; #Coded#

Movement1: TMenuItem;

SpecifyMovement: TMenuItem;

MakeMovement: TMenuItem;

Firing: TMenuItem;

DisplayResults: TMenuItem;

panTabbedArea: TPanel;

ObtainTargets: TMenuItem;

PerformFiring: TMenuItem;

OpenDBDialog: TOpenDialog;

pagCtrl: TPageControl;

tabMap: TTabSheet;

tabRanges: TTabSheet;

RichEdit1: TRichEdit;

OpenMapDialog: TOpenPictureDialog;

TabSheet1: TTabSheet;

mmoTargChg: TMemo;

tabEntry: TTabSheet;

panButtonBar: TPanel;

lblMoveNum: TLabel;

btnMakeMove: TButton;

btnPerformFiring: TButton;

btnSpecifyMove: TButton;

btnObtainTargets: TButton;

btnExtraDamage: TButton;

panExtraDamage: TPanel;

Label16: TLabel;

Label17: TLabel;

cboExtraDamageTarget: TComboBox;

btnAssignExtraDamage: TButton;

medtAmountExtraDamage: TMaskEdit;

btnApplyExtraDamage: TButton;

panAssignMovement: TPanel;

lblVesselMaxSpd: TLabel;

Label8: TLabel;

Label11: TLabel;

cboMoveVessel: TComboBox;

edtVesselCurrSpd: TEdit;

cboVesselDirn: TComboBox;

btnAssignMovement: TButton;

updCurrSpd: TUpDown;

panTarg1: TPanel;

Label2: TLabel;

Label3: TLabel;

lblTarg1Range: TLabel;

label12: TLabel;

lblTarg1BandName: TLabel;

lblTarg1TimeTN: TLabel;

lblTarg1BearingNum: TLabel;

lblTarg1Arc: TLabel;

lblTarg1Control: TLabel;

lblTarg1TimeStep: TLabel;

edtTarg1Num: TEdit;

cboTarg1Wpns: TComboBox;

cboTarg1Target: TComboBox;

rdgTarg1Smoke: TRadioGroup;

panTarg2: TPanel;

Label4: TLabel;

Label5: TLabel;

lblTarg2Range: TLabel;

label13: TLabel;

lblTarg2BandName: TLabel;

lblTarg2TimeTN: TLabel;

lblTarg2BearingNum: TLabel;

lblTarg2Arc: TLabel;

lblTarg2Control: TLabel;

lblTarg2TimeStep: TLabel;

edtTarg2Num: TEdit;

cboTarg2Wpns: TComboBox;

cboTarg2Target: TComboBox;

rdgTarg2Smoke: TRadioGroup;

panTarg3: TPanel;

Label6: TLabel;

Label7: TLabel;

lblTarg3Range: TLabel;

label14: TLabel;

lblTarg3BandName: TLabel;

lblTarg3TimeTN: TLabel;

lblTarg3BearingNum: TLabel;

lblTarg3Arc: TLabel;

lblTarg3Control: TLabel;

lblTarg3TimeStep: TLabel;

edtTarg3Num: TEdit;

cboTarg3Wpns: TComboBox;

cboTarg3Target: TComboBox;

rdgTarg3Smoke: TRadioGroup;

panTarg4: TPanel;

Label9: TLabel;

Label10: TLabel;

lblTarg4Range: TLabel;

label15: TLabel;

lblTarg4BandName: TLabel;

lblTarg4TimeTN: TLabel;

lblTarg4BearingNum: TLabel;

lblTarg4Arc: TLabel;

lblTarg4Control: TLabel;

lblTarg4TimeStep: TLabel;

edtTarg4Num: TEdit;

cboTarg4Wpns: TComboBox;

cboTarg4Target: TComboBox;

rdgTarg4Smoke: TRadioGroup;

panFirer: TPanel;

Label1: TLabel;

lblFirerSpdDmgTN: TLabel;

cboFirer: TComboBox;

btnAssignTargets: TButton;

chbLeadershipDice: TCheckBox;

chbSquadronLeadershipDice: TCheckBox;

chbFleetLeadershipDice: TCheckBox;

updLabelMult: TUpDown;

edtLabelMult: TEdit;

OpenInitDialog: TOpenDialog;

OpenInit: TMenuItem;

OpenGraphics: TMenuItem;

memResults: TMemo;

sgrdState: TStringGrid;

panSea: TPanel;

ptbSea: TPaintBox;

pumBaseMenu: TPopupMenu;

IncrementMagnification1: TMenuItem;

DecrementMagnification1: TMenuItem;

RichEdit2: TRichEdit;

RichEdit3: TRichEdit;

lblDebug: TLabel;

mmoNotes: TMemo;

FiringAtSunkVessels: TMemo;

btnNil1: TButton;

btnNil2: TButton;

btnNil3: TButton;

btnNil4: TButton;

#--------------------------------------------#

# Standard Form Procedures #

#--------------------------------------------#

# Don't need to pre-announce procedures in Python:

#procedure FormCreate(Sender: TObject);

#procedure FormDestroy(Sender: TObject);

#--------------------------------------------#

# Menu Response Procedures #

#--------------------------------------------#

#procedure OpenInitClick(Sender: TObject);

#procedure OpenDatabaseClick(Sender: TObject);

#procedure OpenGraphicsClick(Sender: TObject);

#procedure CloseDatabaseClick(Sender: TObject);

#procedure SaveDatabaseClick(Sender: TObject);

#--------------------------------------------#

#procedure Exit1Click(Sender: TObject);

#--------------------------------------------#

#procedure SpecifyMovementClick(Sender: TObject);

#procedure MakeMovementClick(Sender: TObject);

#procedure ObtainTargetsClick(Sender: TObject);

#procedure PerformFiringClick(Sender: TObject);

#procedure DisplayResultsClick(Sender: TObject);

#--------------------------------------------#

# Utility Procedures #

#--------------------------------------------#

#procedure DestroyObjectsInVessels; ##Destroy all objects stored in VesselList#

#procedure GetRange(Targ1, Targ2:integer; Wpn: integer;

# out Range: Real; out RangeBand: TBandName; out Bearing: Real);

# Get the range difference between Target1 and Target2, and the Range Band associated with Wpn and

# Bearing from Targ1 to Targ2#

#procedure TargTargetChanged(TargNum: integer);

# called when one of the 4 cboTargTarget combo boxes is changed, the param is the number of the combo box changed#

#procedure TargWpnChanged(TargNum: integer);

# called when one of the 4 cboTargWpn combo boxes is changed, the param is the number of the combo box changed#

#procedure TargNumChanged(TargNum: integer; DirectChange: boolean);

# called when one of the 4 edtTargNum Edit boxes is changed, the param is the number of the Edit box changed,

# the boolean indicates if the user changed the number or the programme (FALSE)#

#procedure TargSmokeChanged(TargNum: integer);

# called when one of the 4 rdgTargSmoke Radio Groups changes#

#procedure SetupFirerDetails(FirerNum: integer);

# called when index changes for cboFirer to show the details for that new firer - Max/Curr spd etc#

#procedure SetupMoveDetails(MoverNum: integer);

# called when index changes for cboMoveVessel to show the details for the new mover - curr spd and angle#

#function PositiveAngle(Angle: real; PiOffset: real): real;

# adds Angle to Pi \* PiOffset, and if negative adds 2Pi - angles in radians#

#procedure PlacePanels(ShowMovementPanel: integer);

# ShowMovementPanel == 0 or 1 to either not show or show the movement panel

# sets the appropriate panels visible and places them in the correct locations#

#procedure ApplyDamage(SubMove: string);

# apply all of the damage in DmgRcvdThisMove and ExtraDamage to all vessels

# used by PerformFiring and ApplyExtraDamage#

#procedure RefreshBackground;

# Refresh the Background based on the Label Multiplier#

#procedure RefreshAllGraphics;

# Paint the map by copying the background bitmap if appropriate to current offset

# and vessel/smoke positions#

#procedure SetupItemIndex(ComboBox: TComboBox; SuggestedIndex: integer);

# called to select the next available ie non-sunk index#

#procedure CheckFiringAtSunkVessels(out Firing: boolean);

# checks each target to see if firing is being performed against a sunk target#

#procedure NullifyTarget(TargNum: integer);

# or that TargNum set the NumWpns to zero and the Index for WpnType and Target to -1#

#--------------------------------------------#

# Event Response Procedures #

#--------------------------------------------#

#procedure cboMoveVesselChange(Sender: TObject);

#procedure btnAssignMovementClick(Sender: TObject);

#procedure cboFirerChange(Sender: TObject);

#procedure cboTarg1WpnsChange(Sender: TObject);

#procedure cboTarg2WpnsChange(Sender: TObject);

#procedure cboTarg3WpnsChange(Sender: TObject);

#procedure cboTarg4WpnsChange(Sender: TObject);

#procedure cboTarg1TargetChange(Sender: TObject);

#procedure cboTarg2TargetChange(Sender: TObject);

#procedure cboTarg3TargetChange(Sender: TObject);

#procedure cboTarg4TargetChange(Sender: TObject);

#procedure edtTarg1NumChange(Sender: TObject);

#procedure edtTarg2NumChange(Sender: TObject);

#procedure edtTarg3NumChange(Sender: TObject);

#procedure edtTarg4NumChange(Sender: TObject);

#procedure btnAssignTargetsClick(Sender: TObject);

#procedure cboVesselDirnChange(Sender: TObject);

#procedure scrlChange(Sender: TObject);

#procedure scrlHorizChange(Sender: TObject);

#procedure btnAssignExtraDamageClick(Sender: TObject);

#procedure btnApplyExtraDamageClick(Sender: TObject);

#procedure btnExtraDamageClick(Sender: TObject);

#procedure rdgTarg1SmokeClick(Sender: TObject);

#procedure rdgTarg2SmokeClick(Sender: TObject);

#procedure rdgTarg3SmokeClick(Sender: TObject);

#procedure rdgTarg4SmokeClick(Sender: TObject);

#procedure edtLabelMultChange(Sender: TObject);

#procedure pagCtrlChange(Sender: TObject);

#procedure FormPaint(Sender: TObject);

#procedure ptbSeaMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);

#procedure ptbSeaMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

#procedure IncrementMagnification1Click(Sender: TObject);

#procedure DecrementMagnification1Click(Sender: TObject);

#procedure btnNil1Click(Sender: TObject);

#procedure btnNil2Click(Sender: TObject);

#procedure btnNil3Click(Sender: TObject);

#procedure btnNil4Click(Sender: TObject);

private

# Private declarations #

public

# Public declarations #

# end;

DamageRecord = class(Object)

""" Stores data relative to Damage Blocks

:ivar size the remaining size of the block

speed the maximum speed when damage has occurred in this block

TN the Target Number mod when damage has occurred in this block

.. methods none

"""

Size = 0

Spd = 0

TN = 0

TargetRecord = class(Object)

Target = "" # number of the target within string list#

# may have to change this to name if we delete Sunk ships from the list

WpnNum = 0 # Number of Wpns firing at this target#

WpnIndex = 0 # Index (0..3) into Wpns for Wpn firing at this target#

TargDist = 0 # Distance of Target from Firer#

TimeStep = 0 # Time Step for this moves firing at this

# target ranges from 1 to 9 is 2nd index into FireControlValues array#

ControlType = 0 # the control type for this weapons firing on this target, will usually be

# the same as the equivalent weapons Ctrl value but if it is Fire or Director

# Control firing at an adjacent or other vessel in the same formation: it

# will be 2 or 1 less respectively #

BandName = "bynd" # The range band name for this target, PB, clos, mid, long, extr, bynd#

Bearing = 0.0 # radian angle of bearing from Firer to Target#

Arc = "Side" # the arc that the target is in from the firer, one of Fore, Side, Aft#

Smoke = 0 # set to ItemIndex of the relevant Smoke Radio Group - note the smoke could be

# blocking line of sight in a different way for a different firer, eg a firer in front

# of the target making smoke will see most of the vessel, one behind will see

# only the smoke#

#\*\*\*\* need to update the other smoke areas#

WpnRecord == class(Object)

WpnNum = (0, 0, 0) #Number of Wpns of this type that can fire into the Fore side and aft areas#

WpnType = " " # Type of Wpn in string form#

WpnCode = 0 # index (0..14) into WpnTypes for WpnType above#

WpnSubType = 0 # sub-type of this wpn, either 0 or 1#

WpnCtrlType = 0 # Fire Control type of this weapon - can be 0=local ctrl from Turret/TT, 1=local

# control from other position, 2..4=Fire Ctrl, 5..7=Director Ctrl, 8=Poor Radar,

# 9=Medium Radar, 10=Good Radar. Note 2..4 is because 2=New Target is Adjacent ,

# 3=New Target in Same Formation, 4=New Target in New Formation, similar 5..7#

VesselRecord == class(Object)

Name = ""

VesselClass = ""

Size = "XZ" # Size of Vessel, 1 of XZ, HZ, MZ, LZ#

SizeTN = 0 # TN to hit this vessel due to size#

DmgTN = 0 # TN for this vessel to hit another vessel

# based on the amount of Damage it has taken#

DmgRcvdThisMove = 0 # all damage this vessel receives this move used to apply damage once all firing is

# complete#

ExtraDamage = 0 #Extra Damage assigned this move before movement - ie torpedo or air attacks#

Belt = 0.0 # Thickness of Belt in inches#

Deck = 0.0 # Thickness of Deck in inches#

Movement = "" # setup by movement routines, current moves movement#

# Probably only 1 or 2 chars long, eg 5 or P5 but maybe

# we want to show turn as 45 degrees so 180 deg is PPPP4#

X = 0.0 # Distance in cms Eastwards from some Origin#

Y = 0.0 # Distance in cms Northwards from some Origin#

ShipLabel =" " # A 2 letter label to display the relative location of the ship#

MaxSpd = 0 # Maximum distance this vessel can move in cms dependant

# on the amount of damage it has taken so far may need to set this to real for 0.5 cms#

CurrSpd = 0 # Current distance this vessel is moving in cms#

Heading = 0 # Direction the vessel is moving, a number associated with

# the list NN, NE, EE, SE, SS, SW, WW, NW#

Sunk = 0 # -1=not suitable target (eg merchantman), 0=not sunk, 1=sunk#

OwnLeadershipDice = 1 # 0=used, 1=still available, adds 1 dice to calcs#

OwnLeadershipDiceSelected = False # Set if selected this move#

SquadronLeadershipDice = 0 # 0=used or this is not Sqn Ldr of 4 vessel or more sqn,

# 1=still avail and this is Sqn Ldr, adds 1 dice to calcs#

SquadronLeadershipDiceSelected = False # Set if selected this move#

FleetLeadershipDice = 0 # 0=used or this is not Fleet Adm, 1=still avail

# and this is the Fleet Adm, adds 1 dice to calcs#

FleetLeadershipDiceSelected = False # Set if selected this move#

Wpns: Array[0..3] of WpnRecord;

Target: Array[0..3] of TargetRecord;

Block: Array[0..7] of DamageRecord;

BlockSize = 0 # the size of each block for this target - used to compare

# with each damaged reduced block#

Formation = "" # a string that holds the formation name that the vessel is in

# changing to a target in the same formation has a smaller

# TimeTN to changing to one in a new formation#

Side = 0 # which side a vessel is on, 0 or 1 XXXX add 2 #

Smoke = 0 # The smoke situation, -1=not selected, shouldn't happen, 0=no smoke effect,

# 1=making smoke, 2=in smoke, 3=behind smoke#

VesselsFiringAtThisOne = (0,0,0,0,0,0,0,0) # the number of vessels firing

# each weapon type at this vessel - if>1,: TN penulties

# ignores Torpedoes and AA guns#

DmgLeftInCurrentBlock = 0

# Contains the amount of damage left in the block currently being damaged – to show in String Grid#

# end;

TPolyRec == Class(TObject)

NumPoints = 0 #the number of points actually in the polygon#

Points = 40\*((0,0)) # the actual Points, only the first NumPoints of which are valid#

# end;

const

AngleList = ['NN','NE','EE','SE','SS','SW','WW','NW']

XMult = [0.0, 0.707, 1.0, 0.707, 0.0, -0.707, -1.0, -0.707]

YMult = [1.0, 0.707, 0.0, -0.707, -1.0, -0.707, 0.0, 0.707]

Main = 0; # Index for Main Wpn into WpnRecord#

Sec = 1; # Index for Secondary Wpn into WpnRecord#

Tert = 2; # Index for Tertiary Wpn into WpnRecord#

TT = 3; # Index for TT Wpn into WpnRecord#

maxDuplicateFirers = 4; # number of Duplicate Firers that can be displayed#

RangeBands = [[42,42,42,42,31,31,21,10,8,6,6,4,11,5,1],

[94,84,84,94,63,63,42,31,20,12,12,8,21,16,3],

[178,157,125,167,105,94,73,63,36,20,18,12,37,32,4],

[230,209,209,219,146,125,94,73,56,32,30,18,47,28,6],

[251,230,230,230,154,136,105,84,84,48,44,26,53,42,8],

[999,999,999,999,999,999,999,999,999,999,999,999,999,999,999]]

BandNames = ['n/a ', 'PB ', 'Clos', 'Mid ', 'Long', 'Extr', 'Bynd']

ArcNames = ['Fore', 'Side', 'Aft ']

AngleNumToRadians = [0.0, math.pi/4, math.pi/2, 3\* math.pi/4, math.pi, \

5 \* Math.pi/4, 3 \* math.pi/2, 7 \* math.pi/4]

# Yards Cms Move1 Move2 Move3 Move4 Move5 Move6 Move7 Move8 Move9 #

FireControlValues = [ \

[ 7000, 43, 20, 18, 16, 15, 14, 13, 12, 11, 10] \ #Local Control – Turret #

[ 5000, 30, 22, 20, 18, 17, 16, 15, 14, 13, 12] \ #Local Control – Other #

[28000, 251, 7, 5, 4, 3, 2, 1, 0, 0, 0] \ #Fire Control - Adj Target #

[28000, 251, 9, 7, 5, 4, 3, 2, 1, 0, 0] \ #Fire Control - Same Frmtn #

[28000, 251, 10, 8, 6, 5, 4, 3, 2, 1, 0] \ #Fire Control - New Frmtn #

[28000, 251, 2, 0, -1, -2, -3 -4, -5, -5, -5] \ #Director Control - Adj Target#

[28000, 251, 4, 2, 0, -1, -2, -3, -4, -5, -5] \ #Director Control - Same Frmtn#

[28000, 251, 5, 3, 1, 0, -1, -2, -3, -4, -5] \ #Director Control - New Frmtn #

[99999, 610, 5, 3, 1, -1, -3, -5, -5, -5, -5] \ #Radar Control - Poor #

[99999, 610, 3, 0, -3, -6, -6, -6, -6, -6, -6] \ #Radar Control - Medium #

[99999, 610, 2, -2, -6, -8, -8, -8, -8, -8, -8]]\ #Radar Control - Good #

numBlocks = 8 # the number of blocks before a vessel is sunk (once all are full)

numWpns = 4 # the number of different weapons that each vessel may fire

numTargs = 4 # the number of targets that a vessel may fire at

var

WW2Rules: WW2Rules;

Firer, Target, Vessel = class(VesselRecord)

VesselList: TStringList;

# slstLabels: TStringList; #

lblShipLabel: TLabel;

FormationList: TStringList;

FormationRec: TFormationRecord;

moveNum = 0 # contains the number of the move since the beginning for logging purposes,

# incremented in Perform Movement Routine or obtain movement routine #

moveNumSet = False # set if Move Num updated in Obtain Movement#

logfileIsDirty = False # set if something has been written to either of the log files #

saveDatabaseIsDirty = False # Set if something has been written to the Vessels list #

havePolygons = False # set if Polygons have been read in from Init File #

haveBitMap = False # set if a Map has been read in from Graphics File #

noFileOpened = False # set if no file could be opened to stop error on close #

openDatabaseName = "" # name of the file that is currently open – used to create next Save File#

FiringLogFileName = "" # Name of file that will store the logs of all firing #

ShipLogFileName = "" # Name of file that will store the logs of current ship status #

FiringLogFile: TextFile; # File handle for Firing Log File #

ShipLogFile: TextFile; # File handle for Ship Log File #

headerLine = "" # Contains the Headers Line ready for output again #

sideLeadershipDice = (False,False,False) # contains an indicator of whether

# that sides Fleet Leadership Dice is still available

# it can only be used once and if the Fleet leader is

# sunk: it is cancelled too #

labelMult = 1 # multiplier for Label positioning #

bitmapTop = 0 # holds the Top value for the BitMaps #

bitmapLeft = 0 # holds the Left value for the BitMaps #

cursorX = 0 # holds the X value of the Cursor #

cursorY = 0 # holds the Y value of the Cursor #

unmultipliedWindowTop = 0 # holds the nominal top of window before the value is multiplied by LabelMult #

windForce = 0 # holds the current Wind Strength #

windDirection = 0 # holds the direction number for the wind 0=North,

# 0=NN, 1=NE, ... 7=NW as per AngleList #

PolygonList = [] # holds the list of Polygons if any have been read in -

# havePolygons will be set if there are any #

polygonRec: TPolyRec; # used to set up and read entries in PolygonList #

background: TBitmap; # bitmap used to store any background picture when displaying a map #

duplicateFirers = [] # contains a pointer for each vessel as to how many Firers

# a vessel has engaging it - used in btnAssignTargetsClick #

edtTargNums: Array [0..3] of TEdit;

cboTargWpns: Array [0..3] of TComboBox;

cboTargTargets: Array [0..3] of TComboBox;

lblTargRanges: Array [0..3] of TLabel;

lblTargTimeTN: Array [0..3] of TLabel;

lblTargTimeStep: Array [0..3] of TLabel;

lblTargControl: Array [0..3] of TLabel;

lblTargBandName: Array [0..3] of TLabel;

lblTargBearingNum: Array [0..3] of TLabel;

lblTargArc: Array [0..3] of TLabel;

rdgTargSmoke: Array [0..3] of TRadioGroup;

btnNilTarg: Array [0..3] of TButton;

debugTime: integer;

implementation

#$R \*.DFM#

#--------------------------------------------#

# Standard Form Procedures #

#--------------------------------------------#

#--------------------------------------------#

# Form Create #

# Create the string list to contain all #

# the vessels, & set Sorted to False #

# Set flag to show database is not open #

# Set the Random Number Seed #

# Point all the arrays of controls to #

# actual controls so that they can #

# be indexed with the same index #

# value as in the database #

#--------------------------------------------#

def WW2Rules.FormCreate(Sender: TObject):

begin

VesselList = TStringList.Create;

VesselList.Sorted = False; # do not sort strings, keep in order input #

# slstLabels = TStringList.Create; why has this been commented out? Proly couldn’t get it to work

slstLabels.Sorted = False; # need to keep them in the same order as VesselList #

FormationList = TStringList.Create;

FormationList.Sorted = True; # this time we do want the formations to be sorted #

PolygonList = TStringList.Create;

PolygonList.Sorted = FALSE; # shouldn't matter but set it unsorted anyway #

background = TBitmap.Create;

background.Height = ptbSea.Height;

background.Width = ptbSea.Width;

background.Canvas.Brush.Color = clInactiveCaptionText;

background.Canvas.Brush.Style = bsSolid;

background.Canvas.Rectangle(0, 0, background.Width, background.Height);

noFileOpened = True;

havePolygons = FALSE;

haveBitMap = FALSE;

moveNum = 0; # initialise the Move Num to 0, may be overwritten by InitFile #

moveNumSet = FALSE; # clear the Move Num Set logical so that the correct action will

# be taken in Specify/Perform Movement #

edtLabelMult.Text = '3' # set the default graphical multiplier to be 3 #

labelMult = 3; # to doubly check that the value is reasonable #

bitmapTop = 0; # set start offset to be top left of the background #

bitmapLeft = 0;

cursorX = 0; # initialise the Cursor position#

cursorY = 0;

unmultipliedWindowTop = 787; # initialise to known value in case InitFile not read

# this is the value used in all the early simulations #

# load the random number seed from the system clock - only needs to be done once#

# not necessary in Python ?

# Randomize;

edtTargNums[0] = edtTarg1Num;

edtTargNums[1] = edtTarg2Num;

edtTargNums[2] = edtTarg3Num;

edtTargNums[3] = edtTarg4Num;

cboTargWpns[0] = cboTarg1Wpns;

cboTargWpns[1] = cboTarg2Wpns;

cboTargWpns[2] = cboTarg3Wpns;

cboTargWpns[3] = cboTarg4Wpns;

cboTargTargets[0] = cboTarg1Target;

cboTargTargets[1] = cboTarg2Target;

cboTargTargets[2] = cboTarg3Target;

cboTargTargets[3] = cboTarg4Target;

lblTargRanges[0] = lblTarg1Range;

lblTargRanges[1] = lblTarg2Range;

lblTargRanges[2] = lblTarg3Range;

lblTargRanges[3] = lblTarg4Range;

lblTargBandName[0] = lblTarg1BandName;

lblTargBandName[1] = lblTarg2BandName;

lblTargBandName[2] = lblTarg3BandName;

lblTargBandName[3] = lblTarg4BandName;

lblTargTimeTN[0] = lblTarg1TimeTN;

lblTargTimeTN[1] = lblTarg2TimeTN;

lblTargTimeTN[2] = lblTarg3TimeTN;

lblTargTimeTN[3] = lblTarg4TimeTN;

lblTargTimeStep[0] = lblTarg1TimeStep;

lblTargTimeStep[1] = lblTarg2TimeStep;

lblTargTimeStep[2] = lblTarg3TimeStep;

lblTargTimeStep[3] = lblTarg4TimeStep;

lblTargControl[0] = lblTarg1Control;

lblTargControl[1] = lblTarg2Control;

lblTargControl[2] = lblTarg3Control;

lblTargControl[3] = lblTarg4Control;

lblTargBearingNum[0] = lblTarg1BearingNum;

lblTargBearingNum[1] = lblTarg2BearingNum;

lblTargBearingNum[2] = lblTarg3BearingNum;

lblTargBearingNum[3] = lblTarg4BearingNum;

lblTargArc[0] = lblTarg1Arc;

lblTargArc[1] = lblTarg2Arc;

lblTargArc[2] = lblTarg3Arc;

lblTargArc[3] = lblTarg4Arc;

rdgTargSmoke[0] = rdgTArg1Smoke;

rdgTargSmoke[1] = rdgTArg2Smoke;

rdgTargSmoke[2] = rdgTArg3Smoke;

rdgTargSmoke[3] = rdgTArg4Smoke;

btnNilTarg[0] = btnNil1;

btnNilTarg[1] = btnNil2;

btnNilTarg[2] = btnNil3;

btnNilTarg[3] = btnNil4;

btnSpecifyMove.Enabled = FALSE;

btnMakeMove.Enabled = FALSE;

btnObtainTargets.Enabled = FALSE;

btnPerformFiring.Enabled = FALSE;

sgrdState.Cells[0,0] = 'Name';

sgrdState.Cells[1,0] = 'Class';

sgrdState.Cells[2,0] = 'Curr Spd';

sgrdState.Cells[3,0] = 'Max Spd';

sgrdState.Cells[4,0] = 'Time TN';

sgrdState.Cells[5,0] = 'Dmg';

sgrdState.Cells[6,0] = 'Sunk';

sgrdState.Cells[7,0] = 'Fire1';

sgrdState.Cells[8,0] = 'Fire2';

sgrdState.Cells[9,0] = 'Fire3';

sgrdState.Cells[10,0] = 'Fire4';

mmoTargChg.Lines.add('#,NewTarg,Firer,CurTarg,CurWpn,CurCtrl,NewForm,OldForm,NewStep,Targ,'

+ 'OldForm,NewStep,Targ,OldForm,NewStep,Targ,OldForm,NewStep,Targ,FinalStep,FinalCtrl');

# scrlVert.Position = 0;

# scrlHoriz.Position = 0;#

panExtraDamage.Visible = FALSE;

panAssignMovement.Visible = FALSE;

panFirer.Visible = FALSE;

panTarg1.Visible = FALSE;

panTarg2.Visible = FALSE;

panTarg3.Visible = FALSE;

panTarg4.Visible = FALSE;

end;

#--------------------------------------------#

# Form Destroy #

# Call the routine to destroy all the #

# sub-records in each record,: the #

# record itself (work from last to first #

# so that the records still exist when #

# they are deleted #

# Destroy the String List that holds #

# vessel records #

#--------------------------------------------#

def WW2Rules.FormDestroy(Sender: TObject):

var

i: integer;

begin

DestroyObjectsInVessels; #Destroy all the objects in the VesselList variable#

VesselList.Free;

# slstLabels.Free; #

FormationList.Free;

for i in range(PolygonList.Count):

begin

polygonRec = TObject(PolygonList.Objects[i]) as TPolyRec;

polygonRec.Free;

# end;

PolygonList.Free;

background.Free;

if not(noFileOpened):

#flush the last entries to the log files#

CloseFile(FiringLogFile);

CloseFile(ShipLogFile);

# end;

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Menu Response Procedures #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

#--------------------------------------------#

# Open Init Click #

# Only perform procedure if we haven't #

# already read in a BitMap #

# Obtain Init File Name and open file #

# Read the Width, Height, GraphicsMult #

# Read the Top/Left of the BitMaps to #

# show interesting bit of area #

# Read the Wind Force and Direction #

# Read the number of points in the first #

# Polygon, if > 0: #

# Set HavePolygons boolean #

# Empty and Destroy String List to #

# hold polygons #

# For each Point in the Polygon, #

# Read the X and Y #

# Add to Polygon #

# Add the Polygon to the string list #

# If Next Number of Points > 0: #

# Read in Next Polygon, etc. #

# Close the Database file #

# #

#--------------------------------------------#

def WW2Rules.OpenInitClick(Sender: TObject):

var

FileHandle: TextFile;

lineRead = "" # holds the line read - one value per line #

polygonPointsNum = 0 # holds the number of points in the polygon #

i = 0

polygonName = "" # holds the name of the current Polygon #

polygonNum = 0 # holds the number of the current Polygon, 0 based #

begin

if not(haveBitMap): #XXXX maybe create subprocs and leave this in superproc?

begin

if OpenInitDialog.Execute:

begin

AssignFile(FileHandle, OpenInitDialog.FileName);

Reset(FileHandle);

ReadLn(FileHandle, lineRead);

moveNum = int(lineRead);

lblMoveNum.Caption = "Move Number = %i" % MoveNum

ReadLn(FileHandle, lineRead);

background.Width = int(lineRead);

ReadLn(FileHandle, lineRead);

background.Height = int(lineRead);

ReadLN(FileHandle, lineRead);

unmultipliedWindowTop = int(lineRead);

ReadLn(FileHandle, lineRead);

edtLabelMult.Text = lineRead; #set the graphical multiplier#

#XXXX do we need to set the internal multiplier?

ReadLn(FileHandle, lineRead);

bitmapLeft = int(lineRead);

ReadLn(FileHandle, lineRead);

bitmapTop = int(lineRead);

ReadLn(FileHandle, lineRead);

windForce = int(lineRead);

if (windForce < 0) or (windForce > 7):

Application.MessageBox( 'Invalid Wind Force (0..7) ' , 'Init Wind Error', MB\_OKCancel);

ReadLn(FileHandle, lineRead);

windDirection = int(lineRead);

if (windDirection < 0) or (windDirection > 7):

Application.MessageBox( 'Invalid Wind Direction (0..7) ' , 'Init Wind Error', MB\_OKCancel);

ReadLn(FileHandle, lineRead);

polygonPointsNum = int(lineRead);

if polygonPointsNum > 0:

begin

havePolygons = TRUE;

PolygonList.Clear;

polygonNum = -1; #will be incremented to 0 on first pass through while loop#

while polygonPointsNum > 0:

begin

if polygonPointsNum > 40:

Application.MessageBox('Too Many Polygon Points (Limit 40)','Init Error',MB\_OK);

return #XXXX is this correct?

polygonRec = TPolyRec.Create;

for i in range(polygonPointsNum):

ReadLn(FileHandle, lineRead);

polygonRec.Points[i].X = int(lineRead);

ReadLn(FileHandle, lineRead);

# Note, the Y values are currently counted from the bottom of the window, to assign

# a value based on the top of the window, subtract the Y value from a value which is

# the nominal top of the window (before multiplying by the LabelMult) #

polygonRec.Points[i].Y = UnmultipliedWindowTop - int(lineRead);

# Clear the rest of the array #

for i in range(polygonPointsNum, 40):

begin

polygonRec.Points[i].X = 0;

polygonRec.Points[i].Y = 0;

# end;

polygonRec.NumPoints = polygonPointsNum;

polygonNum = polygonNum + 1;

polygonName = 'Polygon' + Str(polygonNum);

polygonList.AddObject(polygonName, polygonRec);

ReadLn(FileHandle, lineRead);

polygonPointsNum = int(lineRead);

# end;

RefreshBackground; # setup the background using polygons if any were imported #

# end;

CloseFile(FileHandle); # close the opened database file as we have no more use for it#

# end;

# end;

RefreshAllGraphics; # Paint the whole map #

# end of OpenInitClick;

#--------------------------------------------#

# Open Database Click #

# Subprocedure to separate each line #

# Fields #

# Obtain Database Name and open file #

# Empty and Destroy String List to hold #

# vessels, in case a second Database #

# is opened #

# Read the header line in the database #

# For each data line in the database, #

# read the line, separate into fields #

# create a record and copy the fields #

# to the record. Add the record to #

# the String List. #

# Close the Database file #

# Initialise the Move Number #

# Open the Log Files, write the header #

# lines, set the Dirty Flag #

# Clear the comboboxes for the Firer, #

# and all 4 targets #

# Add each vessel to each of these cbos #

# #

#--------------------------------------------#

def WW2Rules.OpenDatabaseClick(Sender: TObject):

var

FileHandle: TextFile;

lineRead = ""

fieldValuesList = []

numFieldsRead = 0

i = 0

angleNum = 0

currentDateTime: string;

description: string; # holds the name and class of the vessel for the combo boxes #

#- - - - - - - - - - - - - - - - - - - - - - #

# Subprocedure to split a string into #

# Separate fields, based upon the #

# Separator - usually comma, but can be #

# semi-colon if there are subs-fields #

# Returns the passed String List and the #

# integer number of fields read #

# #

#- - - - - - - - - - - - - - - - - - - - - - #

def SeparateFields (record: String; separator: Char;

var slstFields: TStringList; var numFieldsRead: integer):

var startPos, sepPos = 0, 0

begin

numFieldsRead = 0;

startPos = 1;

sepPos = Pos(separator, record);

while sepPos <> 0:

begin

slstFields.Add (Copy(record, startPos, sepPos - startPos));

record = Copy(record, sepPos+1, length(record)-sepPos + 1);

numFieldsRead = numFieldsRead + 1;

sepPos = Pos(separator, record);

# end;

if length(record) > 0:

begin

slstFields.Add (record);

numFieldsRead = numFieldsRead + 1;

# end;

# end; of Separate\_Fields #

begin

noFileOpened = False;

if OpenDBDialog.Execute:

AssignFile(FileHandle, OpenDBDialog.FileName)

else:

# begin

noFileOpened = True;

return #XXXX is this correct?

# end;

openDatabaseName = OpenDBDialog.FileName; # store the name of the open file

# so it can be used as template for save #

Reset(FileHandle);

sideLeadershipDice[0] = FALSE;

sideLeadershipDice[1] = FALSE;

DestroyObjectsInVessels; # if there are any objects already in VesselList: destroy them #

VesselList.Clear;

fieldValuesList = TStringList.Create;

# Read the Header Line which has titles but no Data #

Readln(FileHandle, headerLine);

While not SeekEOF(FileHandle):

begin

Readln(FileHandle, lineRead);

fieldValuesList.Clear;

SeparateFields(lineRead, ',', fieldValuesList, numFieldsRead);

Vessel = VesselRecord.Create;

with Vessel do

begin

Name = fieldValuesList[0];

VesselClass = fieldValuesList[1];

Size = fieldValuesList[2];

SizeTN = int(fieldValuesList[3]);

DmgTN = int(fieldValuesList[4]);

DmgRcvdThisMove = 0;

Belt = StrToFloat(fieldValuesList[6]);

Deck = StrToFloat(fieldValuesList[7]);

Movement = ''; #not needed when importing#

X = StrToFloat(fieldValuesList[8]);

Y = StrToFloat(fieldValuesList[9]);

ShipLabel = fieldValuesList[10];

MaxSpd = int(fieldValuesList[11]);

CurrSpd = int(fieldValuesList[12]);

angleNum = 0;

while (fieldValuesList[13] <> AngleList[angleNum]) and (angleNum < numAngles):

Inc(angleNum);

if angleNum == 8:

Application.MessageBox( 'Invalid Angle ' , 'Input Angle Error', MB\_OKCancel);

Heading = angleNum;

Sunk = int(fieldValuesList[14]);

OwnLeadershipDice = int(fieldValuesList[15]);

OwnLeadershipDiceSelected = FALSE;

SquadronLeadershipDice = int(fieldValuesList[16]);

SquadronLeadershipDiceSelected = FALSE;

FleetLeadershipDice = int(fieldValuesList[17]);

FleetLeadershipDiceSelected = FALSE;

for i in range(numWpns):

begin

Vessel.Wpns[i] = WpnRecord.Create;

Vessel.Wpns[i].WpnNum[0] = int(fieldValuesList[18 + i \* 7]);

Vessel.Wpns[i].WpnNum[1] = int(fieldValuesList[18 + i \* 7 + 1]);

Vessel.Wpns[i].WpnNum[2] = int(fieldValuesList[18 + i \* 7 + 2]);

Vessel.Wpns[i].WpnType = fieldValuesList[18 + i \* 7 + 3];

Vessel.Wpns[i].WpnCode = int(fieldValuesList[18 + i \* 7 + 4]);

Vessel.Wpns[i].WpnSubType = int(fieldValuesList[18 + i \* 7 + 5]);

Vessel.Wpns[i].WpnCtrlType = int(fieldValuesList[18 + i \* 7 + 6]);

# end;

for i in range(numTargs):

begin

Vessel.Target[i] = TargetRecord.Create;

Target[i].Target = int(fieldValuesList[46 + i \* 4]);

Target[i].WpnNum = int(fieldValuesList[46 + i \* 4 + 1]);

Target[i].WpnIndex = int(fieldValuesList[46 + i \* 4 + 2]);

Target[i].TimeStep = int(fieldValuesList[46 + i \* 4 + 3]);

debugTime = Target[i].TimeStep;

# Select Control Type, will always be firing against a New Formation at start of battle #

if Target[i].WpnIndex == -1:

Target[i].ControlType = 1 #set to Local - Other if no control type assigned#

else:

Target[i].ControlType = Vessel.Wpns[Target[i].WpnIndex].WpnCtrlType;

Target[i].BandName = 'Bynd';

Target[i].Bearing = 0.0;

Target[i].Arc = 'Side';

Target[i].Smoke = 0; #no smoke effect#

# end;

BlockSize = int(fieldValuesList[86]);

DmgLeftInCurrentBlock = BlockSize; #takes care of the situation when no damage has been received at all#

for i in range(numBlocks):

begin

Vessel.Block[i] = DamageRecord.Create;

Block[i].Size = int(fieldValuesList[62 + i \* 3]);

Block[i].Spd = int(fieldValuesList[62 + i \* 3 + 1]);

Block[i].TN = int(fieldValuesList[62 + i \* 3 + 2]);

if Block[i].Size < BlockSize:

DmgLeftInCurrentBlock = Block[i].Size;

# end;

Formation = fieldValuesList[87];

if SquadronLeadershipDice == 1:

begin

FormationRec = TFormationRecord.Create;

FormationRec.FormationLeadershipDice = TRUE;

FormationList.AddObject(Formation, FormationRec);

# end;

Side = int(fieldValuesList[88]);

Smoke = 0;

if FleetLeadershipDice == 1:

sideLeadershipDice[Side] = TRUE;

# end of with Vessel;

VesselList.AddObject (Vessel.Name, Vessel);

# lblShipLabel = TLabel.Create(panSea);

lblShipLabel.Parent = panSea;

lblShipLabel.Caption = Vessel.ShipLabel;

lblShipLabel.Top = panSea.Height – math.floor(Vessel.Y) \* labelMult;

lblShipLabel.Left = math.floor(Vessel.X) \* labelMult;

sColourLetter = copy(Vessel.Name,0,1);

if sColourLetter == 'G':

lblShipLabel.Font.Color = clGreen

elif sColourLetter == 'R':

lblShipLabel.Font.Color = clRed

elif sColourLetter == 'B':

lblShipLabel.Font.Color = clBlue;

# any other colour: leave it as default #

# slstLabels.AddObject (Vessel.ShipLabel, lblShipLabel); #

# end; #of While not SeekEOF(FileHandle) do#

CloseFile(FileHandle); # close the opened database file as we have no more use for it #

RefreshAllGraphics; # paint the map - to show the labels that we have just added #

FiringLogFileName = ExtractFilePath(openDatabaseName) + 'Firing\_Log\_Move';

DateTimeToString(currentDateTime, '\_yyyy\_mm\_dd\_hh\_mm', Now);

FiringLogFileName = "%s%i%s.csv" % FiringLogFileName, moveNum, currentDateTime

AssignFile(FiringLogFile, FiringLogFileName);

Rewrite(FiringLogFile);

WriteLN(FiringLogFile, 'Move #,Firer,Move,Target,Wpn Type,Wpn #,Range,Band,Spd,'

+ 'Base TN,Size Mod,Dmg Mod,Time Mod,XT TN,Mult Wpn,Total TN,Dice,' +

'Total Dice,Hits,Belt Deck,Actual Arm,Arm Pen,Success,Dmg');

ShipLogFileName = ExtractFilePath(openDatabaseName) + 'Ship\_Log\_Move';

ShipLogFileName = "%s%i%s.csv" % ShipLogFileName, moveNum, currentDateTime

AssignFile(ShipLogFile, ShipLogFileName);

Rewrite(ShipLogFile);

WriteLN(ShipLogFile, 'Move #,Name,Spd,Dmg TN,Sunk,Block1,Block2,Block3,Block4,' \

+ 'Block5,Block6,Block7,Block8');

logfileIsDirty = True;

cboFirer.Clear;

for i in range(numTargs):

cboTargTargets[i].Clear;

for i in range(VesselList.Count):

with TObject(VesselList.Objects[i]) as VesselRecord do

begin

description = Name + ' - ' + VesselClass;

if Sunk == 1:

description = 'SUNK-' + description;

cboMoveVessel.Items.Add(description);

cboFirer.Items.Add(description);

cboTargTargets[0].Items.Add(description);

cboTargTargets[1].Items.Add(description);

cboTargTargets[2].Items.Add(description);

cboTargTargets[3].Items.Add(description);

cboExtraDamageTarget.Items.Add(description);

# end;

SetupItemIndex(cboFirer, 0); #select the first non-Sunk name in the list as firer#

SetupFirerDetails(0);

sgrdState.RowCount = VesselList.Count + 2;

for i in range(VesselList.Count):

begin

Vessel = (TObject(VesselList.Objects[i]) as VesselRecord);

with Vessel do

begin

sgrdState.Cells[0, i + 1] = Name;

sgrdState.Cells[1, i + 1] = VesselClass;

sgrdState.Cells[5, i + 1] = IntToStr(DmgLeftInCurrentBlock);

if Sunk == 1:

sgrdState.Cells[6, i + 1] = 'Sunk'

elif DmgLeftInCurrentBlock <> Blocksize:

sgrdState.Cells[6, i + 1] = 'Dmgd'

else:

sgrdState.Cells[6, i + 1] = 'OK';

# end;

# end;

SetLength(duplicateFirers,VesselList.Count); # set the length of the dynamic array to be

# large enough for all of the vessels #

for i in range(VesselList.Count): # point each entry to the first row in sgrdState #

duplicateFirers[i] = 0;

btnSpecifyMove.Enabled = TRUE;

btnMakeMove.Enabled = FALSE;

btnObtainTargets.Enabled = FALSE;

btnPerformFiring.Enabled = FALSE;

# end;

#--------------------------------------------#

# Open Graphics Click #

# If no polygons have been read so far #

# If the user selects a file and clicks #

# OK: #

# Create a BitMap #

# Load the BitMap from the file #

# Set Transparent to True and select #

# bottom left bit as the Transparent #

# Bit #

# Draw the BitMap on to the ptbSea #

# canvas #

# Set the HaveBitMap boolean #

# Free the BitMap #

# #

#--------------------------------------------#

def WW2Rules.OpenGraphicsClick(Sender: TObject):

var

MapBitmap: TBitMap;

begin

if not(havePolygons):

begin

if OpenMapDialog.Execute:

begin

MapBitmap = TBitmap.Create;

try:

# might have EOFError - not currently handled XXXX

with MapBitmap do

begin

LoadFromFile(OpenMapDialog.Filename);

Transparent = True;

TransParentColor = MapBitMap.canvas.pixels[0,MapBitmap.Height - 1];

background.Width = Width;

background.Height = Height;

background.Canvas.Draw(0,0,MapBitMap);

haveBitMap = TRUE;

# end;

except EOFError, read\_fail:

if ApplicationMessage("Read Fail in OpenGraphicsClick","Quit","Yes or No"):

# check if we need to save database

if database\_is\_dirty:

if ApplicationMessage("SaveDatabase?","Save","Yes or No"):

# XXXX call save the database

else

# XXXX quit

finally:

MapBitmap.Free;

# end;

# end;

# end;

# end of OpenGraphicsClick;

#--------------------------------------------#

# Close Database Click #

# Set the No Database Open Flag #

# Close the Log files and clear their #

# Dirty Flags #

# Clear the Database Dirty Flag #

# Clear the Log File Names #

# #

#--------------------------------------------#

def WW2Rules.CloseDatabaseClick(Sender: TObject):

begin

noFileOpened = True;

CloseFile(FiringLogFile);

CloseFile(ShipLogFile);

logfileIsDirty = False;

saveDatabaseIsDirty = False;

FiringLogFileName = ''; # Clear the name out so as not to cause confusion later #

ShipLogFileName = '';

# end;

#--------------------------------------------#

# Save Database Click #

# Create the Save filename from the #

# Database name, and open it #

# For each Vessel #

# Build the record from the String #

# List entry and write it to the #

# Save File #

# Close the file & clear the dirty flag #

# #

#--------------------------------------------#

def WW2Rules.SaveDatabaseClick(Sender: TObject):

var

SaveDatabaseFile: TextFile;

SaveDatabaseName: TFilename;

currentDateTime ""

i, j, k = 0, 0, 0

lineRead = ""

begin

DateTimeToString(currentDateTime, '\_yyyy\_mm\_dd\_hh\_mm', Now);

SaveDatabaseName = "%sShip\_Data+Move%i%s.csv" % ExtractFilePath(openDatabaseName), moveNum, currentDateTime

AssignFile(SaveDatabaseFile, SaveDatabaseName);

Rewrite(SaveDatabaseFile);

WriteLN(SaveDatabaseFile, headerLine); #write the HeaderLine Back out#

for i in range(VesselList.Count):

begin

Vessel = TObject(VesselList.Objects[i]) as VesselRecord;

with Vessel do

begin

lineRead = "%s,%s,%s,%i,%i," % Name, VesselClass, Size, SizeTN, DmgTN

lineRead += "%i,%f6.2,%f6.2,%i,%i,%s" % DmgRcvdThisMove, Belt, Deck, X, Y, ShipLabel

# note, Movement doesn't need to be saved it will always be empty on input #

lineRead += "%i,%i,%s,%i," % MaxSpd, CurrSpd, AngleList[Heading], Sunk

lineRead += "%i,%i,%i," % OwnLeadershipDice, SquadronLeadershipDice, FleetLeadershipDice

for j in range(numWpns):

for k in range(3):

lineRead += "%i," % Wpns[j].WpnNum[k]

lineRead += "%i,%i,%i,%i," % Wpns[j].WpnType, Wpns[j].WpnCode, Wpns[j].WpnSubType, \ Wpns[j].WpnCtrlType

for j in range(numTargs):

lineRead += "%i,%i,%i,%i," % Target[j].Target), Target[j].WpnNum, Target[j].WpnIndex), \ Target[j].TimeStep

For j in range(numBlocks):

lineRead += "%i,%i,%i," % Block[j].Size), Block[j].Spd, Block[j].TN

lineRead += "%i,%s,%i," % BlockSize, Formation, Side

# end; of with vessel do#

WriteLN(SaveDatabaseFile, lineRead);

# end; of for i = 0 to VesselList.Count - 1 do#

CloseFile(SaveDatabaseFile);

SaveDatabaseIsDirty = False;

# empty the buffers of the log files so that they can be opened read-only to check the latest updates #

Flush(FiringLogFile);

Flush(ShipLogFile);

# end; of SaveDatabaseClick

#--------------------------------------------#

# Exit Click #

# If the Database is Dirty: check #

# that the programme should exit #

# without saving #

# If a Database has been opened #

# close the log files #

# Terminate the application #

# Else #

# If a database has been opened #

# close the log files #

# Terminate the application #

# #

#--------------------------------------------#

def WW2Rules.Exit1Click(Sender: TObject):

begin

if SaveDatabaseIsDirty:

begin

if Application.MessageBox('Really Exit without Saving Database?', 'Exit Programme', MB\_OKCANCEL) == IDOK:

begin

if not(noFileOpened):

begin

CloseFile(FiringLogFile);

CloseFile(ShipLogFile);

# end;

Application.Terminate;

# end;

# end

else:

begin

if not(noFileOpened):

begin

CloseFile(FiringLogFile);

CloseFile(ShipLogFile);

# end;

Application.Terminate;

# end;

# end; of ExitClick

#--------------------------------------------#

# Specify Movement Click #

# If the Database hasn't been opened #

# Output an error message and exit #

# Fill the Move Vessel combo-box - is #

# this really necessary? #

# Select the first entry in the Move #

# Vessel Combo Box #

# #

#--------------------------------------------#

def WW2Rules.SpecifyMovementClick(Sender: TObject):

var

i: integer;

begin

if noFileOpened:

begin

if Application.MessageBox('Database Not Opened Yet', 'Specify Movement', MB\_OK) == IDOK:

return #XXXX is this correct?

# end;

# Increment the Move Number#

moveNum = moveNum + 1;

lblMoveNum.Caption = 'Move Number = ' + IntToStr(moveNum);

moveNumSet = TRUE;

# Display the Form to get the Movement for each Vessel#

btnAssignMovement.Enabled = TRUE;

for i in range(VesselList.Count):

begin

with TObject(VesselList.Objects[i]) as VesselRecord do

begin

# can't ignore any vessels otherwise can't link back to VesselList #

# moved the setting up of this combo box to open database with the other combo box setups #

# clear the movement field so that it can be refilled afresh #

Movement = '';

# end;

# end; for i = 0 to slstVesselList.Count - 1 do #

SetupItemIndex(cboMoveVessel, 0); # select the first non-sunk vessel #

SetupMoveDetails(cboMoveVessel.ItemIndex);

SetupItemIndex(cboFirer, 0); # select the first non-sunk name in the list #

if cboFirer.ItemIndex > - 1:

begin

cboFirerChange(self);

# end;

# note, cboFirerChange can reset the ItemIndex to -1#

# if cboFirer.ItemIndex > -1:

SetupFirerDetails(0); #

btnMakeMove.Enabled = TRUE;

btnSpecifyMove.Enabled = FALSE;

PlacePanels(1); # shows the movement panel #

# end; of SpecifyMovementClick

#--------------------------------------------#

# Make Movement Click #

# If the Database hasn't been opened #

# Output an error message and exit #

# Increment the Move Number #

# For each vessel in the String List #

# Apply the X & Y movement #

# #

#--------------------------------------------#

def WW2Rules.MakeMovementClick(Sender: TObject):

var

i: integer;

begin

if noFileOpened:

begin

if Application.MessageBox('Database Not Opened Yet', 'Make Movement', MB\_OK) == IDOK:

return #XXXX is this correct?

# end;

if not(moveNumSet):

begin

# Increment the Move Number#

moveNum = moveNum + 1;

lblMoveNum.Caption = 'Move Number = ' + IntToStr(moveNum);

# end;

moveNumSet = FALSE;

# Make the movements specified in SpecifyMovement and all the routines called as a result of making changes #

for i in range(VesselList.Count):

begin

with TObject(VesselList.objects[i]) as VesselRecord do

begin

X = X + CurrSpd \* XMult[Heading];

Y = Y + CurrSpd \* YMult[Heading];

Movement = AngleList[Angle] + IntToStr(CurrSpd);

# end;

# end;

RefreshAllGraphics; # Paint the map to the graphics tab #

# enable/disable the buttons as appropriate #

btnSpecifyMove.Enabled = FALSE;

btnMakeMove.Enabled = FALSE;

btnObtainTargets.Enabled = TRUE;

btnPerformFiring.Enabled = TRUE;

btnAssignMovement.Enabled = FALSE;

# end;

#--------------------------------------------#

# Obtain Targets Click #

# If the Database hasn't been opened #

# Output an error message and exit #

# Select the first entry in the Firer #

# Combo Box #

# Setup the remaining details for that #

# Firer #

# #

#--------------------------------------------#

def WW2Rules.ObtainTargetsClick(Sender: TObject):

var i: integer;

begin

if noFileOpened:

begin

if Application.MessageBox('Database Not Opened Yet', 'Obtain Targets', MB\_OK) == IDOK:

return #XXXX is this correct?

# end;

btnAssignMovement.Enabled = FALSE;

btnAssignTargets.Enabled = TRUE;

PlacePanels(0); # shows the firing panels but not the movement one #

# Note all vessels are loaded into the Firer and Target Lists when opening a Database #

SetupItemIndex(cboFirer, 0); # Select the first non-Sunk vessel in the list #

cboFirerChange(self);

# call the routine to handle change of selected firer #

#SetupFirerDetails(0); \*\*\*\*\*\*\*\*\*\*\*\*\*\*??? #

# set DuplicateFirers to point to first entry, may need to clear the sgrdState

# entries too, but will leave them for now so that it is possible to see the

# other firers from last turn \*\*\*\*#

for i in range(VesselList.Count):

duplicateFirers[i] = 0;

# enable/disable the buttons as appropriate #

btnSpecifyMove.Enabled = FALSE;

btnMakeMove.Enabled = FALSE;

btnObtainTargets.Enabled = FALSE;

btnPerformFiring.Enabled = TRUE;

# end;

#---------------------------------------------------------------------#

# Perform Firing Click #

# If any vessel is targeting a sunk vessel: warn the user #

# and maybe exit #

# For each firer #

# Clear the DmgReceivedThisMove and VesselsFiringAtThisOne for #

# all 8 main gun types. #

# Count the number of each weapon type that is firing at this #

# vessel. eg if 3 ships are firing MM at this vessel: the #

# count will be 3. If 1 ship is also firing LM: the count #

# for that will be 1, the combined effect will be to reduce #

# their chance to hit quite considerably #

# For each Firer #

# For each Target of that Firer, if it has one #

# (note all data is logged to the FiringLog File) #

# Compute TargDist, WpnType, RangeBand, BaseTN, SizeTN, #

# modification for Large/XS/HS guns firing at Light ships #

# FireControlTN based on ControlType and Time Step #

# If TimeStep is not max, increment it #

# If Firer has no Radar: compute SmokeTN effects #

# Compute CrossingTTN based on bearing of target and angle of #

# target #

# Compute MultiWpnTN by adding the number of vessels firing #

# at this target with WpnTypes between 1 lower and 1 higher #

# than this WpnType and multiply by 2 (note if only one #

# ship firing these types of wpns (ie this ship): set #

# TN to zero).: add 1 for each Wpn that is 2 less or 2 #

# more than this type. #

# If Firer is not being fired at: reduce MultiWpnTN #

# #

# Part 2 #

# Compute Number of Dice to roll and roll that number of dice #

# Note add 1 dice for each of the LeadershipDice selected #

# Compute NumHits from TotalTN and DiceResults #

# Compute potential Belt and Deck Penetration if any hits #

# For each hit #

# Compute if Belt or Deck hit #

# If Belt hit: check if target's belt is unarmoured, #

# half the penetratable armour, or penetratable, or #

# not penetratable at all, add damage caused as approp #

# If Deck hit: ditto but there is no half penetratable #

# value #

# Part 3 #

# When all hits are computed, apply all damage to all vessels #

# note no letter is added to move number, this is only #

# used when Extra damage is applied (with an X) #

# Disable Fire Buttons and Enable Move Buttons #

# #

#---------------------------------------------------------------------#

def WW2Rules.PerformFiringClick(Sender: TObject):

const WpnTypes: Array [0..14] of string[2] == ('XM','HM','MM','LM','XS','HS','MS',

'LS','XT','HT','MT','LT','MA','LA','AA');

BaseTNByRangeAndSpeed: Array [0..4,0..7] of integer == ((0, 2, 2, 4, 6, 9, 10, 11),

(3, 5, 5, 7, 10, 13, 14, 16),

(7, 9, 9, 11, 14, 17, 18, 21),

(11, 13, 13, 15, 18, 21, 23, 26),

(14, 16, 16, 19, 22, 25, 28, 30));

TNforArmourTypePen: Array [0..4] of integer == (2, 4, 6, 8, 10);

BeltPenetration: Array [0..17] of real == (14.0, 13.0, 12.5, 12.0, 11.5, 11.0, 10.5, 10.0,

6.0, 5.5, 5.0, 4.5, 4.0, 3.5, 3.0, 2.5, 2.0, 0.0);

#XM HM MM LM XS HS#

RangeOfBeltPen: Array [0..16, 0..5] of real == (

(246,186,105, 97, -1, -1),

(249,197,112,107, 31, -1),

(250,202,114,112, 34, -1),

(251,207,120,117, 38, -1),

(251,211,125,123, 41, -1),

(251,214,130,129, 44, -1),

(251,217,137,136, 48, -1),

(251,219,143,142, 52, -1),

(251,230,207,190, 98, 57),

(251,230,213,196,107, 63),

(251,230,218,201,118, 72),

(251,230,223,208,132, 85),

(251,230,227,213,142,100),

(251,230,230,219,148,116),

(251,230,230,227,152,129),

(251,230,230,230,153,135),

(251,230,230,230,154,136));

DeckPenetration: Array [0..12] of real == (6.0, 5.5, 5.0, 4.5, 4.0, 3.5, 3.0, 2.5, 2.0,

1.5, 1.0, 0.5, 0.0);

RangeOfDeckPen: Array [0..11, 0..5] of real == (

# XM HM MM LM XS HS#

(198,210,213,221, -1, -1),

(194,203,195,207, -1, -1),

(190,195,177,195, -1, -1),

(186,187,158,181, -1, -1),

(182,180,142,167, -1, -1),

(178,168,125,150, -1, -1),

(174,153,110,135,156, -1),

(169,114, 96,117,148, -1),

(162,114, 83,100,130,120),

(156,114, 62, 80,105, 94),

(150,114, 40, 50, 84, 78),

(133,114, 40, 50, 63, 63));

ArmourPen == 0; # used to index into Damage Array #

MediumArmour == 1;

UnarmouredBelt == 2;

UnarmouredDeck == 3;

NoPenetration == 4;

Damage: Array [0..4, 0..21] of integer == (

(18, 0, 15, 0, 12, 11, 10, 0, 8, 0, 6, 0, 0, 0, 0, 0, 48, 0, 36, 0, 24, 0),

( 9, 0, 7, 0, 6, 5, 4, 0, 3, 0, 2, 0, 0, 0, 0, 0, 60, 0, 45, 0, 30, 0),

( 6, 0, 5, 0, 4, 3, 3, 0, 2, 0, 2, 12, 9, 0, 6, 4, 72, 0, 54, 0, 36, 0),

(18, 0, 15, 0, 12, 11, 10, 0, 8, 0, 6, 12, 9, 0, 6, 4, 0, 0, 0, 0, 0, 0),

( 4, 0, 3, 0, 2, 2, 2, 0, 1, 0, 1, 0, 0, 0, 0, 0, 24, 0, 18, 0, 12, 0));

var

i, j, k, l: integer;

sFiringLogData: string;

iWpnType: integer;

iRangeBand: integer;

iBaseTN: integer;

iSizeTN: integer;

iNumDice: integer;

iNumDiceRolled: integer;

iDiceResults: integer;

sDiceRolled: string;

iRandomNumber: integer;

iNumHits: integer;

sBeltDeck: string;

rBeltPenetration: real;

rDeckPenetration: real;

sPenetrationSuccess: string;

iDamage: integer;

MsgString: PChar;

iCrossingTTN: integer; # TN modifier for Crossing the targets "T", -5 or 0 #

iMultiWpnTN: integer; # TN modifier for number of wpns on different firers of similar size firing at Target #

rAngleInRadians: real; # the angle in radians of the target's track converted from a number 0..7 #

smmoTargChgLine: string; # used to output a line to mmoTargChg #

bFiringAtSunkVessel: boolean; # set by CheckFiringAtSunkVessel if any firer has a target that is sunk #

iVesselsTargetingFirer: integer; # number of vessels targeting Firer #

begin

# look through each non-sunk firer to see if they have a target that is sunk #

CheckFiringAtSunkVessels(bFiringAtSunkVessel);

if bFiringAtSunkVessel:

begin

if Application.MessageBox('Vessels Firing at Sunk Vessels, Abandon Firing?', 'Perform Firing', \

MB\_YESNO) == IDYES:

begin

exit;

# end;

# end;

# initialise DmgRcvdThisMove to zero - it should be anyway because all damage received in the previous move will

# have been assigned #

# also clear VesselsFiringAtThisOne and load it for this move #

for i in range(VesselList.Count):

begin

Vessel = TObject(VesselList.Objects[i]) as VesselRecord;

Vessel.DmgRcvdThisMove = 0;

for j in range(8): #XXXX change this to be open ended - have larger guns now?

Vessel.VesselsFiringAtThisOne[j] = 0;

for j in range(VesselList.Count):

begin

Firer = TObject(VesselList.Objects[j]) as VesselRecord;

If (Vessel.Side <> Firer.Side) and (Vessel.Sunk <> 1) and (Firer.Sunk <> 1):

begin

lblDebug.Caption = Vessel.Name + ',';

lblDebug.Caption = lblDebug.Caption + Firer.Name + ',';

for k in range(numWpns):

begin

lblDebug.Caption = lblDebug.Caption + IntToStr(k) + ',';

lblDebug.Caption = lblDebug.Caption + IntToStr(Firer.Target[k].WpnNum) + ',';

if Firer.Target[k].WpnNum > 0:

# this firer is really shooting #

begin

# Target of this firer is the current vessel #

if (Firer.Target[k].Target == i) and (Firer.Wpns[Firer.Target[k].WpnIndex].WpnCode < 8):

# and the wpn is a Main or Sec one, ie not AA or Torp #

begin

Vessel.VesselsFiringAtThisOne[Firer.Wpns[Firer.Target[k].WpnIndex].WpnCode] += 1

# end;

# end;

# end;

# end;

# end;

# end;

# for each potential firer#

for i in range(VesselList.Count):

begin

Firer = TObject(VesselList.Objects[i]) as VesselRecord;

# don't bother to check if vessel is sunk, it will have no targets if this is the case#

for j in range(numTargs):

begin

if (Firer.Target[j].WpnNum > 0) and (Firer.Target[j].Target > -1): # this is a real target #

begin

SaveDatabaseIsDirty = True; # changed something in the Vessels DB #

Target = TObject(VesselList.Objects[Firer.Target[j].Target]) as VesselRecord;

# Move ,Firer,Movement,Target #

sFiringLogData = "%i,%s,%s,%s," % (moveNum, Firer.Name. Firer.Movement. Target.Name)

# calculate TN #

Firer.Target[j].TargDist = SQRT((Firer.X-Target.X)\*(Firer.X-Target.X) + \

(Firer.Y-Target.Y)\*(Firer.Y-Target.Y));

iWpnType = Firer.Wpns[Firer.Target[j].WpnIndex].WpnCode;

iRangeBand = 0;

while (Firer.Target[j].TargDist > RangeBands[iRangeBand, iWpnType]) and (iRangeBand < 5):

iRangeBand = iRangeBand + 1;

if iRangeBand == 5:

begin

MsgString = PChar('Distance Too Great ' + Firer.Name + ' ' + Target.Name);

Application.MessageBox( MsgString, 'Range Error', MB\_OKCancel);

Continue;

# end;

# Wpn,,Range,Band #

sFiringLogData += "%s,%i,%i,%s" % (Firer.Wpns[Firer.Target[j].WpnIndex].WpnType, \

Firer.Target[j].WpnNum, math.ceil(Firer.Target[j].TargDist), BandNames[iRangeBand])

iBaseTN = BaseTNByRangeAndSpeed[iRangeBand,Target.CurrSpd];

iSizeTN = Target.SizeTN;

if Target.Size == 'MZ':

begin

# handle large guns firing on MZ sized vessels #

if copy(Firer.Wpns[Firer.Target[j].WpnIndex].WpnType,0,1) == 'M':

iSizeTN = iSizeTN + 8

elif (Firer.Wpns[Firer.Target[j].WpnIndex].WpnType == 'XS') \

or (Firer.Wpns[Firer.Target[j].WpnIndex].WpnType == 'HS'):

# handle XS/HS guns firing on MZ sized vessels #

iSizeTN = iSizeTN + 4;

# end;

# Spd,Base TN,Size Mod,Dmg Mod,Time Mod #

sFiringLogData += "%i,%i,%i,%i,%i," % (Target.CurrSpd, iBaseTN, iSizeTN, Firer.DmgTN)

sFiringLogData += "%i,%i," % (FireControlValues[Firer.Target[j].ControlType, \ Firer.Target[j].TimeStep])

iBaseTN = iBaseTN + iSizeTN;

iBaseTN = iBaseTN + Firer.DmgTN;

iBaseTN = iBaseTN + FireControlValues[Firer.Target[j].ControlType,Firer.Target[j].TimeStep];

if Firer.Target[j].TimeStep < 9:

begin

Firer.Target[j].TimeStep = Firer.Target[j].TimeStep + 1;

debugTime = Firer.Target[j].TimeStep;

# end;

# Only apply Smoke conditions if Firer has no Radar #

if Firer.Target[j].ControlType < 8:

begin

# handle smoke conditions #

if Firer.Target[j].Smoke == 1:

# target making smoke #

iBaseTN = iBaseTN + 2

elif Firer.Target[j].Smoke == 2:

# target in smoke #

iBaseTN = iBaseTN + 6

elif Firer.Target[j].Smoke == 3:

# target behind smoke #

begin

iBaseTN = iBaseTN + 10;

# loose ranging bonus when behind smoke - reset to first time step #

Firer.Target[j].TimeStep = 1;

debugTime = Firer.Target[j].TimeStep;

# end;

# end;

# handle crossing T #

#XXXX is this the correct bearing? want bearing of firer from target

rAngleInRadians = AngleNumToRadians[Target.Heading];

if ((Firer.Target[j].Bearing >= PositiveAngle(rAngleInRadians, -1/12)) \

and (Firer.Target[j].Bearing <= PositiveAngle(rAngleInRadians, 1/12))) \

or ((Firer.Target[j].Bearing >= PositiveAngle(rAngleInRadians, 5/12)) \

and (Firer.Target[j].Bearing <= PositiveAngle(rAngleInRadians, 7/12))):

iCrossingTTN = -5

else:

iCrossingTTN = 0;

iBaseTN = iBaseTN + iCrossingTTN;

# handle multiple wpn sizes firing on the same target #

iMultiWpnTN = 0;

for k in range(MAX(0, iWpnType - 1), MIN(iWpnType + 1, 7)):

iMultiWpnTN += Target.VesselsFiringAtThisOne[k]

if iMultiWpnTN == 1:

iMultiWpnTN = 0 # only 1 wpn firing: no TN modifier #

else:

iMultiWpnTN = iMultiWpnTN \* 2; # otherwise penalty = double num vessels firing #

if iWpnType > 1:

iMultiWpnTN += Target.VesselsFiringAtThisOne[iWpnType - 2]

if iWpnType < 6:

iMultiWpnTN += Target.VesselsFiringAtThisOne[iWpnType + 2]

iVesselsTargetingFirer = 0;

for k in range(8): #XXXX may need to change this

iVesselsTargetingFirer += Firer.VesselsFiringAtThisOne[k]

if iVesselsTargetingFirer == 0:

iMultiWpnTN = iMultiWpnTN - 5; # Nobody firing at this Firer,then get a bonus #

iBaseTN += iMultiWpnTN

# CrossingT TN,MultiWpnTN #

sFiringLogData += "%i,%i,%i," % (iCrossingTTN, iMultiWpnTN, iBaseTN)

# Calculate Hits #

# Calculate the number of dice to roll #

if Firer.Target[j].WpnNum < 4:

iNumDice = 1

elif Firer.Target[j].WpnNum < 7:

iNumDice = 2

elif Firer.Target[j].WpnNum < 10:

iNumDice = 3

elif Firer.Target[j].WpnNum < 14:

iNumDice = 4

elif Firer.Target[j].WpnNum < 19:

iNumDice = 5

else:

iNumDice = 5 + math.ceil((Firer.Target[j].WpnNum - 18)/6);

if Firer.OwnLeadershipDiceSelected:

iNumDice = iNumDice + 1;

if Firer.SquadronLeadershipDiceSelected:

iNumDice = iNumDice + 1;

if Firer.FleetLeadershipDiceSelected:

iNumDice = iNumDice + 1;

# roll that number of dice #

iDiceResults = 0;

iNumDiceRolled = 0;

sDiceRolled = ''; # holds a string list of the dice rolled, with A representing 10 #

while iNumDiceRolled < iNumDice:

begin

# Get a RandomNumber from 1 to 10 #

iRandomNumber = random.randrange(10) + 1; # Randrange produces an integer 0<=r<10 #

iDiceResults = iDiceResults + iRandomNumber;

if iRandomNumber == 10:

sDiceRolled += 'A'

else:

begin

sDiceRolled += IntToStr(iRandomNumber);

# Note if a 10 is rolled: roll another dice-achieved by not counting the 10 just rolled #

iNumDiceRolled += 1;

# end;

# end;

# Check if at least one hit occurs - another hit occurs for every 5 points beyond the TN #

iNumHits = 0 if iDiceResults < iBaseTN else math.ceil((iDiceResults-iBaseTN + 1)/5)

# note if the difference is exactly a multiple of 5: 1 hit less than the correct number was being set

# - adding 1 should fix this#

# ,Dice,Total Dice,Hits #

sFiringLogData += "%s,%i,%i" % (sDiceRolled, iDiceResults, iNumHits)

sBeltDeck = '';

sPenetrationSuccess = '';

iDamage = 0;

if iNumHits > 0:

begin

# calculate Penetration depending on Belt and Range #

rBeltPenetration = BeltPenetration[0];

rDeckPenetration = DeckPenetration[0];

if iWpnType < 6: # wpns of type 6 and higher cannot penetrate armour #

begin

for l in range(17):

if Firer.Target[j].TargDist > RangeOfBeltPen[l, iWpnType]:

rBeltPenetration = BeltPenetration[l+1];

for l in range(12):

if Firer.Target[j].TargDist < RangeOfDeckPen[l, iWpnType]:

rDeckPenetration = DeckPenetration[l+1];

# end

else:

begin

rBeltPenetration = 0.0;

rDeckPenetration = 0.0;

# end;

# end;

for k in range(iNumHits):

begin

# Get a RandomNumber from 1 to 10 #

iRandomNumber = random.randrange(10) + 1; # randrange produces an integer 0<=r<10 #

if iRandomNumber >= TNforArmourTypePen[iRangeBand]:

begin

# this is a BELT hit #

sBeltDeck = sBeltDeck + 'B';

if math.abs(Target.Belt) < 0.001: # defensive coding in case of rounding errors #

begin

# add Dmg for UnarmouredBelt based on Firer Type and Sub #

sPenetrationSuccess += 'U';

iDamage = iDamage + Damage[UnarmouredBelt, iWpnType\*2 + \

Firer.Wpns[Firer.Target[j].WpnIndex].WpnSubType];

# end

elif rBeltPenetration > 2\*Target.Belt + 1:

begin

# add Dmg for Medium Armour based on Firer Type and Sub #

sPenetrationSuccess += 'M';

iDamage += Damage[MediumArmour, iWpnType\*2 + \

Firer.Wpns[Firer.Target[j].WpnIndex].WpnSubType];

# end

elif rBeltPenetration >= Target.Belt:

begin

# add Dmg for Armoured based on Firer Type and Sub #

sPenetrationSuccess += 'Y';

iDamage += Damage[ArmourPen, iWpnType\*2 + \ Firer.Wpns[Firer.Target[j].WpnIndex].WpnSubType];

# end

else:

begin

# add Dmg for NoPenetration based on FirerType/Sub #

sPenetrationSuccess += 'N';

iDamage += Damage[NoPenetration, iWpnType\*2 + \

Firer.Wpns[Firer.Target[j].WpnIndex].WpnSubType];

# end;

# end if iRandomNumber >= TNforArmourTypePen[iRangeBand]#

else

begin

# this is a DECK hit #

sBeltDeck += 'D';

if math.abs(Target.Deck) < 0.001: # defensive coding in case of rounding errors #

begin

# add Dmg for Un-ArmouredDeck based on FirerType/Sub #

sPenetrationSuccess += 'u';

iDamage += Damage[UnarmouredDeck, iWpnType\*2 + \

Firer.Wpns[Firer.Target[j].WpnIndex].WpnSubType];

# end

elif rDeckPenetration >= Target.Deck:

begin

# add Dmg for Armoured based on FirerType/Sub #

sPenetrationSuccess += 'y';

iDamage += Damage[ArmourPen, iWpnType\*2 + \

Firer.Wpns[Firer.Target[j].WpnIndex].WpnSubType];

# end

else:

begin

# add Dmg for NoPenetration based on FirerType/Sub #

sPenetrationSuccess += 'n';

iDamage += Damage[NoPenetration, iWpnType\*2 + \

Firer.Wpns[Firer.Target[j].WpnIndex].WpnSubType];

# end;

# end; If RandomNumber < TN for Range (deck hit)#

# end; of for each hit#

# Belt Deck,Actual Arm,Arm Pen,Success,Damage #

sFiringLogData += "%s,%f6.2,%f6.2,%f6.2;%f6.2,%s,%i," % (sBeltDeck, \

Target.Belt, Target.Deck, rBeltPenetration, rDeckPenetration, sPenetrationSuccess, iDamage)

Target.DmgRcvdThisMove += iDamage;

# Write details of firing to log #

# Move ,Firer,Movement,Target,Wpn ,Range,Band,Spd,Base TN,Time Mod,Dmg Mod,Size Mod,Total TN,Dice,Total

# Dice,Hits,Belt Deck,Actual Arm,Arm Pen,Success,Damage #

writeln(FiringLogFile, sFiringLogData);

# end; #if WpnNum > 0#

# end; #for each target#

end; #for each Firer#

# Note, damage cannot be assigned to a vessel until all firing has completed -

# otherwise a vessel's fire could be reduced by the damage it took during this move #

ApplyDamage(''); # Apply all of the damage assigned this move and

# don't add any extra character after the move number #

# enable/disable the buttons as appropriate #

btnSpecifyMove.Enabled = TRUE;

btnMakeMove.Enabled = TRUE;

btnObtainTargets.Enabled = FALSE;

btnPerformFiring.Enabled = FALSE;

btnAssignTargets.Enabled = FALSE;

# end; of PerformFiringClick

def WW2Rules.DisplayResultsClick(Sender: TObject):

begin

#Display the results of the last moves firing to the User#

end;

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Utility Procedures #

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

#---------------------------------------------------------------------#

# Destroy Objects In Vessels #

# For each vessel, starting with the highest and working down #

# Free each Block (Damage Record) #

# Free each WpnRecord #

# Free each TargetRecord #

# Free the VesselRecord #

# #

#---------------------------------------------------------------------#

def WW2Rules.DestroyObjectsInVessels:

var

i, j: integer;

iOrigVesselCount: integer;

begin

iOrigVesselCount = VesselList.Count;

for i in range(iOrigVesselCount - 1, 0, -1): #XXXX

begin

with TObject(VesselList.Objects[i]) as VesselRecord do

begin

for j in range(numBlocks):

(TObject(Block[j]) as DamageRecord).Free;

for j in range(numWpns):

begin

(TObject(Wpns[j]) as WpnRecord).Free;

(TObject(Target[j]) as TargetRecord).Free;

# end;

# end;

(TObject(VesselList.Objects[i]) as VesselRecord).Free;

# (TObject(slstLabels.Objects[i]) as TLabel).Free; #

# end;

# end; of DestroyAllObjects

#---------------------------------------------------------------------#

# Get Range #

# Save the X and Y values for both vessels #

# Compute the Range as the sum of the squares of the differences #

# Compute the Range Band from the Range and the Wpn Type #

# Compute the Bearing, handling the special cases first where #

# the ships are exactly on the same axis, either horizontally #

# or vertically #

# Note bearing is the angle that Targ2 is from Targ1, ie Targ1 #

# is the firer. Bearing is calculated with 0 being vertically #

# upwards #

# #

#---------------------------------------------------------------------#

def WW2Rules.GetRange(Targ1, Targ2: integer; Wpn: integer;

out Range: Real; out RangeBand: TBandName;

out Bearing: Real):

var

iRangeBand: integer;

X1, Y1, X2, Y2, DiffX, DiffY: real;

Name1, Name2: string;

begin

with TObject(VesselList.Objects[Targ1]) as VesselRecord do

begin

X1 = X;

Y1 = Y;

Name1 = Name;

# end;

with TObject(VesselList.Objects[Targ2]) as VesselRecord do

begin

X2 = X;

Y2 = Y;

Name2 = Name;

# end;

DiffX = X1 - X2;

DiffY = Y1 - Y2;

Range = SQRT(DiffX \* DiffX + DiffY \* DiffY);

iRangeBand = 0;

while (Range > RangeBands[iRangeBand, Wpn]) and (iRangeBand < 5):

iRangeBand += 1;

# Note, if iRangeBand is 5, BandNames[5] is correctly Bynd=Beyond #

RangeBand = BandNames[iRangeBand];

if math.abs(X1-X2) < 0.001: # defensive coding in case of rounding errors #

Bearing = 0.0 if Y2 > Y1 else math.pi

# else option Y2 <= Y1 - assume two vessels are not in same location #

elif math.abs(Y1-Y2) < 0.001: # defensive coding in case of rounding errors #

Bearing = math.pi / 2.0 if X2 > X1 else 3.0 \* math.pi / 2.0

# else option X2 < X1 - case for equal positions already catered for #

elif (X2 > X1) and (Y2 > Y1): #top right quadrant#

Bearing = ARCTAN((X2-X1)/(Y2-Y1))

elif (X2 > X1) and (Y2 < Y1): #bottom right quadrant#

Bearing = math.pi - ARCTAN((X2-X1)/(Y1-Y2))

elif (X2 < X1) and (Y2 > Y1): #top left quadrant#

Bearing = 2.0 \* math.pi - ARCTAN((X1-X2)/(Y2-Y1))

elif (X2 < X1) and (Y2 < Y1): #bottom left quadrant#

Bearing = math.pi + ARCTAN((X1-X2)/(Y1-Y2))

else:

Application.MessageBox( 'Cannot find Bearing ' , 'Bearing Error', MB\_OKCancel);

# end; of GetRange

#---------------------------------------------------------------------#

# Targ Target Changed #

# Compute new Target Index from ItemIndex of specified combobox #

# Select Firer from Firer ComboBox Item Index #

# Set New Time Step to 0 and New Control to Control Type of #

# associated Wpn or 1 if no Wpn selected #

# If new target has really changed from last selected target #

# and a real target has been selected #

# if the control type is not Local or Radar: #

# Set New Formation to the formation of the newly selected #

# target #

# Check through the old targets to see if any of them are #

# in the same formation as the new target #

# If it is the same target and the NewTimeStep is less #

# the old TimeStep: set the NewTimeStep and Control #

# from the old ones #

# If it is an adjacent target: and TimeStep is 0 (ie #

# it has not been changed yet): set TimeStep to 1 #

# and Control to WpnTypeCtrl – 2 #

# else if it is in same formation but not adjacent: #

# set Control to WpnTypeCtrl – 1 #

# If TimeStep is still 0 (either not firing at any target yet #

# or at a non-adjacent one in the same formation): set #

# TimeStep to 1, in both cases Control has already been set #

# Call routine to Change Wpn Type #

# #

#---------------------------------------------------------------------#

def WW2Rules.TargTargetChanged(TargNum: integer):

var

iTargetIndex: integer;

i: integer;

sNewFormation: string;

iNewStep: integer; #holds time step value prior to storing it in non-visible label#

iNewCtrl: integer; #holds control value prior to storing it in non-visible label#

sTargChgLine: string; #holds details to be displayed in mmoTargChg#

begin

iTargetIndex = cboTargTargets[TargNum].ItemIndex;

Firer = (TObject(VesselList.Objects[cboFirer.ItemIndex]) as VesselRecord);

sTargChgLine = "%i,%i,%s,%i," % (moveNum, iTargetIndex, Firer.Name, Firer.Target[TargNum].Target)

with Firer do

begin

iNewStep = 0; # to show that the Time Step has not been set up yet#

iNewCtrl = Wpns[Target[TargNum].WpnIndex].WpnCtrlType if Target[TargNum].WpnIndex > -1 else 1

#local control not in turrets - defensive code when target is not assigned#

if (iTargetIndex <> Target[TargNum].Target) and (iTargetIndex > -1) \

and (cboTargWpns[TargNum].ItemIndex > -1): # defensive code when selecting a target when there

# wasn't a previous one, or the target has been de-selected #

begin

sTargChgLine += "%i," % (cboTargWpns[TargNum].ItemIndex)

sTargChgLine += "%i," % (Wpns[max(cboTargWpns[TargNum].ItemIndex,0)].WpnCtrlType)

if (Wpns[cboTargWpns[TargNum].ItemIndex].WpnCtrlType > 1) \ #Local Control#

and (Wpns[cboTargWpns[TargNum].ItemIndex].WpnCtrlType < 8): #Radar Control#

begin

# Fire Control is either "Fire Control"=2..4 or "Director Control"=5..7 #

# note this is the formation of the target selected in the UI not the target selected last move of firing #

sNewFormation = (TObject(VesselList.Objects[iTargetIndex]) as VesselRecord).Formation;

sTargChgLine += sNewFormation + ',';

# Note iNewStep and iNewCtrl already initialised #

# check each vessel currently being fired at to see if there is one in the same formation

# if there is, and it is the adjacent one: at worst the TN should be 2, otherwise 4 #

for i in range(numTargs):

begin

if (Target[i].Target > -1):

begin

sTargChgLine += (TObject(VesselList.Objects[Target[i].Target]) as \

VesselRecord).Formation + ',';

if sNewFormation == (TObject(VesselList.Objects[Target[i].Target]) \

as VesselRecord).Formation:

begin

# New Target is in a Formation that is already under fire #

sTargChgLine += "%i,%i,%i," % (iNewStep, Target[i].Target, Target[i].TimeStep)

if (iTargetIndex == Target[i].Target) and (iNewStep < Target[i].TimeStep):

begin

# already firing a different weapon at the new target #

iNewStep = Target[i].TimeStep;

debugTime = Target[i].TimeStep;

iNewCtrl = Target[i].ControlType;

# end

elif (math.abs(iTargetIndex - Target[i].Target) == 1) and (iNewStep == 0):

# need to cater for the situation when firing on the same + adjacent ship #

begin

# New Target is an Adjacent vessel in the same formation#

iNewStep = 1;

# set to 1 so that if also firing at non-adjacent target the NewCtrl won't be changed again #

iNewCtrl = Wpns[Target[TargNum].WpnIndex].WpnCtrlType - 2;

# WpnCtrlType is either 4 or 7 #

# end

elif iNewStep == 0:

# need to cater for the situation when firing on the same + another ship #

begin

# New Target is a non-Adjacent vessel in the same formation #

iNewCtrl = Wpns[Target[TargNum].WpnIndex].WpnCtrlType - 1;

# WpnCtrlType is either 4 or 7 #

# end;

# end; part of the same formation#

# warning - while it may not be in the same formation this time round the loop it may have been in a previous loop #

# end; a real target if non negative#

# end; for each target currently being fired at#

# end; of not Local or Radar Control#

if iNewStep == 0: # only firing at non-adjacent target, or no target in same formation #

iNewStep = 1; # iNewCtrl will already have been setup #

sTargChgLine += "%i,%i" % (iNewStep, iNewCtrl)

mmoTargChg.Lines.Add(sTargChgLine);

# end; of checking Target Num

lblTargTimeStep[TargNum].Caption = IntToStr(iNewStep);

debugTime = iNewStep;

lblTargControl[TargNum].Caption = IntToStr(iNewCtrl);

lblTargTimeTN[TargNum].Caption = IntToStr(FireControlValues[iNewCtrl, iNewStep]);

# Procedure call to setup the Target Weapon boxes

TargWpnChanged(TargNum);

# end;

# end;

#----------------------------------------------------------------------#

# Targ Wpn Changed – the Wpn type has changed for the spec Target #

# Set the firer index to the selected cboFirer #

# Set the Wpn index to the selected wpn for specified TargNum #

# Point Firer at the VesselList entry pointed to by FirerIndex #

# If the Wpn index has a real value (<>-1) #

# Set the Firer's Target Control Type for this TargNum from #

# the Firer's Wpns Control Type for the Wpn Index #

# else #

# Set the Firer's Target Control Type for this TargNum to 1 #

# (defaults to local control not in turrets #

# Call TargNumChanged for this TargNum, indicating that it has #

# been called from software not directly by the user #

# Parameters: TargNum – indicates which Target of the current Firer #

# has been changed #

# #

#----------------------------------------------------------------------#

def WW2Rules.TargWpnChanged(TargNum: integer):

var

iWpnIndex: integer;

iFirerIndex: integer;

begin

iFirerIndex = cboFirer.ItemIndex;

iWpnIndex = cboTargWpns[TargNum].ItemIndex;

Firer = (TObject(VesselList.Objects[iFirerIndex]) as VesselRecord);

if iWpnIndex > -1:

Firer.Target[TargNum].ControlType = Firer.Wpns[iWpnIndex].WpnCtrlType

else

Firer.Target[TargNum].ControlType = 1; # local control not in turrets #

TargNumChanged(TargNum, FALSE);

# end;

#--------------------------------------------------------------------------#

# Targ Num Changed – the number of wpns has changed for the spec Target #

# Set the Wpn index to the selected wpn for specified TargNum #

# Set the target index to the Target of this Targ group #

# Point Firer at the VesselList entry pointed to by FirerIndex #

# If the Num Wpns edit box has been modified and its value is zero #

# Deselect the current Target and set the time step back to 1 #

# If the Target index points to a real target (not -1) #

# and the Wpn Type is real too (not -1) #

# Set the Wpn Code from the selected Wpns Wpn Code #

# Get the range band between the Firer and Target for this wpn #

# code. #

# Set Targets Bearing from the value returned from GetRange #

# Set up the Targ Ranges label from the returned Range #

# If the target is sunk #

# Set the Range caption to SUNK, Wpn Index to -1 Wpn Num to 0 #

# Target Index to -1, Disable the Wpn Type and Num #

# else Enable the Wpn Type and Num #

# if the range band is beyond, or the range is greater than max #

# for that control type #

# Set the Wpn Nums to zero (to invalidate the target #

# else #

# if the new Range Band Name is not "beyond" #

# Setup the ArcNum based on the Arc Label #

# if the Wpn Num is greater than max for this ArcNum #

# reduce the Wpn Num to this max value #

# Set the labels for the Range Band and Bearing #

# Set the label for the Arc that the firer is firing into #

# Parameters: TargNum – indicates which Target of the current Firer #

# has been changed #

# DirectChange – indicates if the TargNumChanged has been #

# called directly by the user changing the Wpn Num edit #

# box in the UI(True), or if it is called #

# programmatically (False) #

# #

#--------------------------------------------------------------------------#

def WW2Rules.TargNumChanged(TargNum: integer; DirectChange: boolean):

var

iWpnIndex, iTargetIndex, iWpnCode: integer;

rRange: real;

sBandName: TBandName;

rBearing: real;

i: integer;

iArcNum: integer;

iBearingBand: integer;

begin

iWpnIndex = cboTargWpns[TargNum].ItemIndex;

iTargetIndex = cboTargTargets[TargNum].ItemIndex;

Firer = (TObject(VesselList.Objects[cboFirer.ItemIndex]) as VesselRecord);

with Firer do

begin

if (edtTargNums[TargNum].Modified) and (edtTargNums[TargNum].Text == '0'):

begin

cboTargTargets[TargNum].ItemIndex = -1; # Deselect the current target #

Target[TargNum].TimeStep = 1; # Reset the time step #

debugTime = Target[TargNum].TimeStep;

# end;

if (iTargetIndex > -1) and (cboTargWpns[TargNum].ItemIndex > -1):

# defensive code in case target is now null #

begin

iWpnCode = Wpns[iWpnIndex].WpnCode;

GetRange(cboFirer.ItemIndex, iTargetIndex, iWpnCode, rRange, sBandName, rBearing);

Target[TargNum].Bearing = PositiveAngle(rBearing, 0);

lblTargRanges[TargNum].Caption = 'Range = ' + FloatToStrF(rRange, ffFixed, 6, 2) + ' - ';

if ((TObject(VesselList.Objects[iTargetIndex]) as VesselRecord).Sunk == 1):

begin

lblTargRanges[TargNum].Caption = 'SUNK !!!';

cboTargWpns[TargNum].ItemIndex = -1;

edtTargNums[TargNum].Text = '0';

cboTargTargets[TargNum].ItemIndex = -1;

cboTargWpns[TargNum].Enabled = FALSE;

edtTargNums[TargNum].Enabled = FALSE;

# end

else:

begin

cboTargWpns[TargNum].Enabled = TRUE;

edtTargNums[TargNum].Enabled = TRUE;

# end;

if (sBandName == 'Bynd') or (rRange > FireControlValues[Target[TargNum].ControlType, 0]):

# check for exceeding maximum range for Fire Control Type #

edtTargNums[TargNum].Text = '0' # set target invalid if out of range #

else:

begin

if #(Target[TargNum].BandName == 'Bynd') and# (sBandName <> 'Bynd'):

begin

iArcNum = -1;

for i in range(3):

if lblTargArc[TargNum].Caption == ArcNames[i]:

iArcNum = i;

if iArcNum < 0:

begin

Application.MessageBox( 'Invalid Arc ' , 'Arc Error', MB\_OKCancel);

edtTargNums[TargNum].Text = '0';

# end

else:

if int(edtTargNums[TargNum].Text) > Wpns[iWpnIndex].WpnNum[iArcNum]:

edtTargNums[TargNum].Text = IntToStr(Wpns[iWpnIndex].WpnNum[iArcNum]);

# end;

# end;

lblTargBandName[TargNum].Caption = sBandName;

lblTargBearingNum[TargNum].Caption = "%f6.2" % (rBearing)

# XXXX add code to load Arc Labels #

iBearingBand = math.floor(Target[TargNum].Bearing \* 4 / math.pi);

if Heading == 0:

begin

if (iBearingBand == 0) or (iBearingBand == 7):

lblTargArc[TargNum].Caption = 'Fore'

else:

if (iBearingBand == 4) or (iBearingBand == 3):

lblTargArc[TargNum].Caption = 'Aft '

else:

lblTargArc[TargNum].Caption = 'Side';

# end

elif Heading == 4:

begin

if (iBearingBand == 4) or (iBearingBand == 3):

lblTargArc[TargNum].Caption = 'Fore'

elif (iBearingBand == 0) or (iBearingBand == 7):

lblTargArc[TargNum].Caption = 'Aft '

else:

lblTargArc[TargNum].Caption = 'Side';

# end

else:

begin

if (iBearingBand == Heading) or (iBearingBand == Heading - 1):

lblTargArc[TargNum].Caption = 'Fore'

elif (iBearingBand == Heading + 4) or (iBearingBand == Heading + 3):

lblTargArc[TargNum].Caption = 'Aft '

else:

lblTargArc[TargNum].Caption = 'Side';

# end;

# end if (iTargetIndex > -1) and (cboTargWpns[TargNum].ItemIndex > -1)

else:

begin

cboTargWpns[TargNum].Enabled = TRUE;

edtTargNums[TargNum].Enabled = TRUE;

#end; of if iTargetIndex > -1 #

# end; of with Firer do

# end; of TargNumChanged

def WW2Rules.TargSmokeChanged(TargNum: integer):

begin

Firer = TObject(VesselList.Objects[cboFirer.ItemIndex]) as VesselRecord;

if Firer.Target[TargNum].Target > -1:

# (TObject(VesselList.Objects[Firer.Target[TargNum].Target]) as VesselRecord).Smoke =#

Firer.Target[TargNum].Smoke = rdgTargSmoke[TargNum].ItemIndex;

end;

def WW2Rules.SetupFirerDetails(FirerNum: integer):

var

i, j: integer;

begin

Vessel = TObject(VesselList.Objects[FirerNum]) as VesselRecord;

with Vessel do

begin

for i in range(numTargs):

begin

lblTargRanges[i].Caption = '';

lblTargBandName[i].Caption = Target[i].BandName;

lblTargTimeStep[i].Caption = IntToStr(Target[i].TimeStep);

debugTime = Target[i].TimeStep;

lblTargControl[i].Caption = IntToStr(Target[i].ControlType);

lblTargTimeTN[i].Caption = IntToStr(FireControlValues[Target[i].ControlType, Target[i].TimeStep]);

lblTargBearingNum[i].Caption = '0.0';

lblTargArc[i].Caption = 'Side';

cboTargTargets[i].ItemIndex = Target[i].Target;

cboTargWpns[i].ItemIndex = Target[i].WpnIndex;

edtTargNums[i].Text = IntToStr(Target[i].WpnNum);

# Set all of the Smoke Radio Groups to show the last moves Smoke State #

for j in range(numTargs):

rdgTargSmoke[j].ItemIndex = 0 if cboTargTargets[j].ItemIndex == -1 else Target[j].Smoke;

TargNumChanged(i, FALSE);

# call routine to handle number of wpns changes but note the change has been made by software not the user directly#

# end;

# end;

# end;

def WW2Rules.SetupMoveDetails(MoverNum: integer):

begin

with Tobject(VesselList.Objects[MoverNum]) as VesselRecord do

begin

CurrSpd = Max(MaxSpd, CurrSpd) # defensive coding in case Max Spd has been reduced to less than Curr Spd #

lblVesselMaxSpd.Caption = 'Max Speed is: %i' % (MaxSpd)

edtVesselCurrSpd.Text = "%i" % (CurrSpd)

updCurrSpd.Max = MaxSpd;

updCurrSpd.Position = CurrSpd; #XXXX should this be += something wrong here?

cboVesselDirn.ItemIndex = Heading;

# end;

# end;

#--------------------------------------------#

# Positive Angle #

# Adds Angle to Pi \* PiOffset, #

# if negative adds 2Pi #

# if > 2Pi subtracts 2Pi #

# All angles in radians #

# #

#--------------------------------------------#

def WW2Rules.PositiveAngle(Angle: real; PiOffset: real):

var

ModAngle: real;

begin

ModAngle = Angle + math.pi \* PiOffset;

while ModAngle < 0.0:

ModAngle = ModAngle + 2.0 \* math.pi;

while ModAngle > Pi \* 2:

ModAngle = ModAngle - 2.0 \* math.pi;

return ModAngle;

end;

#--------------------------------------------#

# Place Panels #

# Set the following Panels Visible #

# and place them in the correct #

# locations #

# #

#--------------------------------------------#

def WW2Rules.PlacePanels(ShowMovementPanel: integer):

# ShowMovementPanel == 0 or 1 to either not show or show the movement panel #

begin

panExtraDamage.Visible = FALSE;

if ShowMovementPanel == 0:

begin

panAssignMovement.Visible = FALSE;

panFirer.Top = panButtonBar.Height;

# end

else

begin

panAssignMovement.Visible = TRUE;

panAssignMovement.Top = panButtonBar.Height;

panAssignMovement.Left = 0;

panFirer.Top = panAssignMovement.Top + panAssignMovement.Height;

# end;

panFirer.Visible = TRUE;

panFirer.Left = 0;

panTarg1.Visible = TRUE;

panTarg1.Top = panFirer.Top + panFirer.Height;

panTarg1.Left = 0;

panTarg2.Visible = TRUE;

panTarg2.Top = panTarg1.Top + panTarg1.Height;

panTarg2.Left = 0;

panTarg3.Visible = TRUE;

panTarg3.Top = panTarg2.Top + panTarg2.Height;

panTarg3.Left = 0;

panTarg4.Visible = TRUE;

panTarg4.Top = panTarg3.Top + panTarg3.Height;

panTarg4.Left = 0;

# end;

#--------------------------------------------#

# Apply Damage #

# Apply all of the damage received #

# so far to all of the vessels #

# This could be either damage from #

# firing or extra damage. #

# The paramater is meant to take a #

# single letter showing whether #

# move is a normal firing move #

# or eXtra damage received at the #

# start of a move from torpedos #

# or aircraft. Normally '' or 'x' #

# #

#--------------------------------------------#

def WW2Rules.ApplyDamage(SubMove: string):

var

i, j: integer;

iCurrentBlock: integer;

sShipLogData: string;

begin

memResults.Lines.Clear;

for i in range(VesselList.Count):

begin

iCurrentBlock = 0;

Vessel = TObject(VesselList.Objects[i]) as VesselRecord;

# Add the extra damage so that it doesn't have to be accounted for each time that damage is applied #

Vessel.DmgRcvdThisMove = Vessel.DmgRcvdThisMove + Vessel.ExtraDamage;

if Vessel.DmgRcvdThisMove > 0:

begin

memResults.Lines.Add(Vessel.Name +' Received '+ str(Vessel.DmgRcvdThisMove) +' this move ');

# end;

while (Vessel.DmgRcvdThisMove > 0) and (iCurrentBlock < numBlocks):

begin

if Vessel.Block[iCurrentBlock].Size > 0:

begin

if Vessel.DmgRcvdThisMove > Vessel.Block[iCurrentBlock].Size:

begin

Vessel.DmgRcvdThisMove -= Vessel.Block[iCurrentBlock].Size;

Vessel.Block[iCurrentBlock].Size = 0;

iCurrentBlock += 1;

# end

else:

# remaining Damage Received this move is <= remaining damage in this block #

begin

# there was a bug here if exactly filled a block last move and haven't moved on to next block this move #

if Vessel.Block[iCurrentBlock].Size == Vessel.BlockSize:

begin

Vessel.MaxSpd = Vessel.Block[iCurrentBlock].Spd;

if Vessel.CurrSpd > Vessel.MaxSpd:

Vessel.CurrSpd = Vessel.MaxSpd;

Vessel.DmgTN = Vessel.Block[iCurrentBlock].TN;

memResults.Lines.Add(Vessel.Name +' Speed is: %i' % (Vessel.MaxSpd) + \

' Dmg TN is: %i' % (Vessel.DmgTN))

# end;

Vessel.Block[iCurrentBlock].Size = Vessel.Block[iCurrentBlock].Size - Vessel.DmgRcvdThisMove;

sgrdState.Cells[5, i + 1] = "%i" % (Vessel.Block[iCurrentBlock].Size)

Vessel.DmgRcvdThisMove = 0;

# end;

# end

else

# This block is already empty - move on to the next one #

iCurrentBlock = iCurrentBlock + 1;

# end; of While Vessel.DmgRcvdThisMove >0 and iCurrentBlock < 8#

If Vessel.Block[7].Size < 1:

# vessel has been sunk #

begin

if Vessel.Sunk <> 1:

begin

Vessel.Sunk = 1;

writeln(FiringLogFile, "%i%s,%s,Sunk" % (moveNum, SubMove, Vessel.Name)

memResults.Lines.Add(Vessel.Name + ' SUNK this move.');

cboMoveVessel.Items[i] = 'SUNK-' + cboMoveVessel.Items[i];

cboFirer.Items[i] = 'SUNK-' + cboFirer.Items[i];

# Clear the targets from this vessel so that no attempt is made to fire from it #

for j in range(numTargs):

begin

Vessel.Target[j].Target = -1;

Vessel.Target[j].WpnNum = 0;

Vessel.Target[j].WpnIndex = -1;

# potential to cause out of bounds errors #

Vessel.Target[j].TimeStep = 1;

debugTime = Vessel.Target[j].TimeStep;

# end;

# for k in range(VesselList.Count):

# begin

# Firer = TObject(VesselList.Objects[k]) as VesselRecord;

# for j in range(numTargs):

# begin

# if Firer.Target[j].Target == i:

# Firing at the Sunk Vessel #

# begin

# Firer.Target[j].Target = -1;

# Firer.Target[j].WpnNum = 0;

# Firer.Target[j].WpnIndex = -1;

# potential to cause out of bounds errors #

# Firer.Target[j].TimeStep = 1;

# end;

# end;

# end; of check each firer#

# end; Vessel was not already sunk#

# end; Vessel is now sunk

# Clear the Damage stores so that they don't get counted twice, especially the ExtraDamage #

Vessel.DmgRcvdThisMove = 0;

Vessel.ExtraDamage= 0;

#Move ,Name,Spd,Dmg TN,Block1,Block2,Block3,Block4,Block5,Block6,Block7,Block8 #

sShipLogData = "%i%s,%s,%i,%i,%i" % (moveNum, SubMove, Vessel.Name, Vessel.MaxSpd, Vessel.DmgTN, \ Vessel.Sunk)

for j in range(\*):

sShipLogData += ",%i" % (Vessel.Block[j].Size)

WriteLN(ShipLogFile, sShipLogData);

sgrdState.Cells[2, i + 1] = IntToStr(Vessel.CurrSpd);

sgrdState.Cells[3, i + 1] = IntToStr(Vessel.MaxSpd);

sgrdState.Cells[4, i + 1] = IntToStr(Vessel.DmgTN);

if Vessel.Sunk == 1:

begin

sgrdState.Cells[5, i + 1] = '0';

sgrdState.Cells[6, i + 1] = 'SUNK';

# end

else:

# Cell[5,... has already been setup with the correct value#

sgrdState.Cells[6, i + 1] = 'OK';

# end; #for i = 0 to slstVesselList.Count - 1 do#

# end; of Apply Damage

#--------------------------------------------#

# Refresh Background #

# Colour the background bitmap #

# if there are polygons #

# draw the polygons based on the #

# label multiplier #

#\*\*\* add code for imported bitmap #

# #

#--------------------------------------------#

def WW2Rules.RefreshBackground:

var

i, j: integer;

myPolygon: Array of TPoint;

begin

background.Canvas.Pen.Color = clRed;

background.Canvas.Pen.Width = 4;

background.Canvas.Brush.Color = clInactiveCaptionText;

background.Canvas.Brush.Style = bsSolid;

background.Canvas.Rectangle(Rect(0, 0, background.Width, background.Height));

if havePolygons:

begin

background.Canvas.Pen.Color = clOlive;

background.Canvas.Pen.Width = 1;

background.Canvas.Brush.Color = clOlive;

for i in range(PolygonList.Count):

begin

polygonRec = TObject(PolygonList.Objects[i]) as TPolyRec;

SetLength(myPolygon, polygonRec.NumPoints);

for j in range(polygonRec.NumPoints):

begin

myPolygon[j].X = polygonRec.Points[j].X \* labelMult;

myPolygon[j].Y = polygonRec.Points[j].Y \* labelMult;

# end;

background.Canvas.Polygon(myPolygon);

# end;

# end;

# end; of RefreshBackground

#--------------------------------------------#

# Refresh All Graphics #

# For each vessel #

# if the vessel has not been sunk #

# redraw label in new position #

# else #

# set the visible flag to false #

# #

#--------------------------------------------#

def WW2Rules.RefreshAllGraphics:

var

i: integer;

bmpTemp: TBitmap;

sColourLetter: string;

iTextLeft, iTextTop: integer;

begin

bmpTemp = TBitmap.Create;

bmpTemp.Height = background#ptbSea#.Height;

bmpTemp.Width = background#ptbSea#.Width;

bmpTemp.Canvas.Draw(0#bitmapLeft#, 0#bitmapTop#, background);

for i in range(VesselList.Count):

begin

Vessel = (TObject(VesselList.objects[i]) as VesselRecord);

with Vessel do

begin

if Sunk <> 1:

begin

sColourLetter = copy(Vessel.Name,0,1);

if sColourLetter == 'G':

bmpTemp.Canvas.Font.Color = clGreen

elif sColourLetter == 'R':

bmpTemp.Canvas.Font.Color = clRed

elif sColourLetter == 'B':

bmpTemp.Canvas.Font.Color = clBlue;

# any other colour: leave it as default #

bmpTemp.Canvas.Brush.Color = clInactiveCaptionText;

bmpTemp.Canvas.TextFlags = bmpTemp.Canvas.TextFlags and not(ETO\_OPAQUE);

iTextLeft = #bitmapLeft +# math.floor(X) \* labelMult;

iTextTop = #bitmapTop +# (unmultipliedWindowTop – math.floor(Y)) \* labelMult;

bmpTemp.Canvas.TextOut(iTextLeft, # background.Height # iTextTop, ShipLabel);

# lblShipLabel.Top = panSea.Height – math.floor(Y) \* labelMult;

lblShipLabel.Left = math.floor(X) \* labelMult;

# end

else:

lblShipLabel.Visible = False; #

# end; of with Vessel do

# end;

ptbSea.Canvas.Draw(bitmapLeft #0#, bitmapTop #0#, bmpTemp);

bmpTemp.Free;

# end; of RefreshAllGraphics

#--------------------------------------------#

# Setup Item Index #

# With the specified Combo Box #

# while the Item is sunk step on #

# to the next item #

# if the end of the list is reached #

# set the item index to -1 #

# ie, none selected #

# #

#--------------------------------------------#

def WW2Rules.SetupItemIndex(ComboBox: TComboBox; SuggestedIndex: integer):

var

iNextIndex: integer;

begin

iNextIndex = SuggestedIndex;

while (copy(ComboBox.Items[iNextIndex],0,5) == 'SUNK-') and (iNextIndex < ComboBox.Items.Count):

iNextIndex += 1;

if iNextIndex == ComboBox.Items.Count:

ComboBox.ItemIndex = -1 #select no entry#

else:

ComboBox.ItemIndex = iNextIndex;

# end; of SetupItemIndex

#--------------------------------------------#

# checks each target to see if firing #

# is being performed against a sunk #

# target #

# For each vessel #

# if it is not sunk: #

# for each target #

# if the target is sunk #

# set the return flag #

# add firer/target details to #

# the memo #

# #

#--------------------------------------------#

def WW2Rules.CheckFiringAtSunkVessels(out Firing: boolean):

var

i, j: integer;

begin

Firing = FALSE;

FiringAtSunkVessels.Clear;

for i in range(VesselList.Count):

begin

Firer = TObject(VesselList.Objects[i]) as VesselRecord;

if Firer.Sunk == 0:

begin

for j in range(numTargs):

begin

Target = TObject(VesselList.Objects[j]) as VesselRecord;

if Target.Sunk == 1:

begin

Firing = TRUE;

FiringAtSunkVessels.Lines.Add(Firer.Name+' '+Firer.VesselClass+' firing at '+ \

Target.Name+' '+Target.VesselClass);

# end;

# end;

# end;

# end;

return firing

# end; of CheckFiringAtSunkVessels

#--------------------------------------------#

# Nullify the Selected Target #

# deselect the Index for this Target #

# deselect the Index for this WpnType #

# set the number of weapons to zero #

# #

#--------------------------------------------#

def WW2Rules.NullifyTarget(TargNum: integer):

begin

cboTargTargets[TargNum].ItemIndex = -1;

cboTargWpns[TargNum].ItemIndex = -1;

edtTargNums[TargNum].Text = '0';

# end;

#--------------------------------------------#

# Event Response Procedures #

#--------------------------------------------#

def WW2Rules.cboMoveVesselChange(Sender: TObject):

var

iVesselNumber: integer;

# holds the selected vessels number - also an index into the list of vessels #

begin

SetupItemIndex(cboMoveVessel, cboMoveVessel.ItemIndex); # ensure that vessel is not SUNK #

iVesselNumber = cboMoveVessel.ItemIndex;

SetupMoveDetails(iVesselNumber);

SetupItemIndex(cboFirer, iVesselNumber); # match the Firer index to the Mover #

if cboFirer.ItemIndex <> - 1:

begin

cboFirerChange(self);

# end;

# note, cboFirerChange can reset the ItemIndex to -1 #

# if cboFirer.ItemIndex > -1:

SetupFirerDetails(iVesselNumber); #

# end;

def WW2Rules.btnAssignMovementClick(Sender: TObject):

var

iVesselNumber: integer;

# holds the selected vessels number - also an index into the list of vessels #

begin

# User has requested to assign the movement to the selected vessel based upon

# the values in the other components in this panel #

iVesselNumber = cboMoveVessel.ItemIndex;

with Tobject(VesselList.Objects[iVesselNumber]) as VesselRecord do

begin

CurrSpd = int(edtVesselCurrSpd.Text);

Heading = cboVesselDirn.ItemIndex;

# end;

SetupItemIndex(cboMoveVessel, cboMoveVessel.ItemIndex + 1);

# Point to the next non-sunk vessel in the list #

# while ((TObject(VesselList.Objects[cboMoveVessel.ItemIndex]) as VesselRecord).Sunk == 1) \

# and (cboMoveVessel.ItemIndex <> cboMoveVessel.Items.Count - 1):

# cboMoveVessel.ItemIndex += 1; #

if cboMoveVessel.ItemIndex > -1:

SetupMoveDetails(cboMoveVessel.ItemIndex);

SetupItemIndex(cboFirer, cboMoveVessel.ItemIndex); #ensure firer is not sunk#

if cboFirer.ItemIndex <> - 1:

begin

cboFirerChange(self);

# end;

SaveDatabaseIsDirty = True;

# end;

def WW2Rules.cboFirerChange(Sender: TObject):

var

iFirerNum: integer; # holds the selected Firer's number - also an index into the list of vessels #

i: integer;

# rRange: Real;

# sBandName: string[4]; #

bFormationFound: boolean; # set if the current formation has been found #

iFormationIndex: integer; # holds the index of the current formation in the FormationList record #

begin

# Load the Target information for that firer if the target is not == -1

# Note for each target, need to list the possible targets and select the one that is currently selected

#

iFirerNum = cboFirer.ItemIndex;

with Tobject(VesselList.Objects[iFirerNum]) as VesselRecord do

begin

if Sunk == 1: # Note, should never happen now - routine picks next non-sunk vessel #

begin

# Firer has been sunk - deselect Firer #

cboFirer.ItemIndex = -1;

lblFirerSpdDmgTN.Caption = 'SUNK !!!';

btnAssignTargets.Enabled = FALSE;

# end

else:

begin

# Firer has not been sunk#

btnAssignTargets.Enabled = TRUE;

lblFirerSpdDmgTN.Caption = "Speed: %i Damage TN: %i" % (CurrSpd, IntToStr(DmgTN)

cboTarg1Wpns.Clear;

cboTarg2Wpns.Clear;

cboTarg3Wpns.Clear;

cboTarg4Wpns.Clear;

for i in range(numWpns):

begin

cboTargWpns[0].Items.Add(Wpns[i].WpnType);

cboTargWpns[1].Items.Add(Wpns[i].WpnType);

cboTargWpns[2].Items.Add(Wpns[i].WpnType);

cboTargWpns[3].Items.Add(Wpns[i].WpnType);

# end;

chbLeadershipDice.Checked = FALSE;

if OwnLeadershipDice == 1:

chbLeadershipDice.Enabled = TRUE

else:

chbLeadershipDice.Enabled = FALSE;

chbSquadronLeadershipDice.Checked = FALSE;

bFormationFound = FormationList.Find(Formation, iFormationIndex);

if bFormationFound:

with (TObject(FormationList.Objects[iFormationIndex]) as TFormationRecord) do

chbSquadronLeadershipDice.Enabled = TRUE if FormationLeadershipDice else FALSE

else:

chbSquadronLeadershipDice.Enabled = FALSE;

chbFleetLeadershipDice.Checked = FALSE;

chbFleetLeadershipDice.Enabled = TRUE if sideLeadershipDice[Side] else FALSE

SetupFirerDetails(iFirerNum);

# end; Vessel not Sunk

# end; of with Objects as Vessel Record

# end;

def WW2Rules.cboTarg1WpnsChange(Sender: TObject):

begin

TargWpnChanged(0);

# end;

def WW2Rules.cboTarg2WpnsChange(Sender: TObject):

begin

TargWpnChanged(1);

# end;

def WW2Rules.cboTarg3WpnsChange(Sender: TObject):

begin

TargWpnChanged(2);

# end;

def WW2Rules.cboTarg4WpnsChange(Sender: TObject):

begin

TargWpnChanged(3);

# end;

def WW2Rules.cboTarg1TargetChange(Sender: TObject):

begin

TargTargetChanged(0):

# end;

def WW2Rules.cboTarg2TargetChange(Sender: TObject):

begin

TargTargetChanged(1);

# end;

def WW2Rules.cboTarg3TargetChange(Sender: TObject):

begin

TargTargetChanged(2);

# end;

def WW2Rules.cboTarg4TargetChange(Sender: TObject):

begin

TargTargetChanged(3);

# end;

def WW2Rules.edtTarg1NumChange(Sender: TObject);

begin

# if this is called even when the user has made no selection,: look at modified property of TEdit #

TargNumChanged(0, TRUE);

# end;

def WW2Rules.edtTarg2NumChange(Sender: TObject):

begin

TargNumChanged(1, TRUE);

# end;

def WW2Rules.edtTarg3NumChange(Sender: TObject):

begin

TargNumChanged(2, TRUE);

# end;

def WW2Rules.edtTarg4NumChange(Sender: TObject):

begin

TargNumChanged(3, TRUE);

# end;

def WW2Rules.btnAssignTargetsClick(Sender: TObject):

var

i: integer;

iFormationIndex: integer;

begin

with TObject(VesselList.Objects[cboFirer.ItemIndex]) as VesselRecord do

begin

if chbLeadershipDice.Checked:

begin

OwnLeadershipDiceSelected = TRUE;

OwnLeadershipDice = 0; # deselect the used extra dice #

# end

else:

OwnLeadershipDiceSelected = FALSE;

if chbSquadronLeadershipDice.Checked:

begin

SquadronLeadershipDiceSelected = TRUE;

FormationList.Find(Formation, iFormationIndex);

# set the formation record for this formation to false - it has been used #

with (TObject(FormationList.Objects[iFormationIndex]) as TFormationRecord) do

FormationLeadershipDice = FALSE;

# end

else:

SquadronLeadershipDiceSelected = FALSE;

if chbFleetLeadershipDice.Checked:

begin

FleetLeadershipDiceSelected = TRUE;

# set the side record for this side to false - it has been used #

sideLeadershipDice[Side] = FALSE;

# end

else:

FleetLeadershipDiceSelected = FALSE;

for i in range(numTargs):

begin

Target[i].Target = cboTargTargets[i].ItemIndex;

Target[i].WpnNum = int(edtTargNums[i].Text);

Target[i].WpnIndex = cboTargWpns[i].ItemIndex;

Target[i].TimeStep = int(lblTargTimeStep[i].Caption);

debugTime = Target[i].TimeStep;

Target[i].ControlType = int(lblTargControl[i].Caption);

Target[i].BandName = lblTargBandName[i].Caption;

Target[i].Bearing = float(lblTargBearingNum[i].Caption); # or should this be .float

if Target[i].Target <> -1: # pointing to a real target #

begin

if duplicateFirers[Target[i].Target] < maxDuplicateFirers:

begin

sgrdState.Cells[7 + duplicateFirers[Target[i].Target],Target[i].Target + 1] = copy(Name,1,4);

duplicateFirers[Target[i].Target] += 1

# end;

# end;

# end;

# end;

SetupItemIndex(cboFirer, cboFirer.ItemIndex + 1);

if cboFirer.ItemIndex > -1:

begin

btnAssignTargets.Enabled = TRUE;

cboFirerChange(self);

# end

else:

btnAssignTargets.Enabled = FALSE;

# while ((TObject(VesselList.Objects[cboFirer.ItemIndex]) as VesselRecord).Sunk == 1)

# and (cboFirer.ItemIndex <> cboFirer.Items.Count - 1):

# cboFirer.ItemIndex = cboFirer.ItemIndex + 1;

#

# if cboFirer.ItemIndex <> - 1:

# begin

# end;

# note, cboFirerChange can reset the ItemIndex to -1 #

# if cboFirer.ItemIndex > -1:

# begin

# SetupFirerDetails(cboFirer.ItemIndex);

# end; #

# end; of btnAssignTargetsClive

def WW2Rules.cboVesselDirnChange(Sender: TObject);

begin

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#

# Need to check that it has only changed by 1 angle and update movement#

# movement also needs to be zeroed when the new movement is obtained#

# end;

return

def WW2Rules.scrlChange(Sender: TObject):

begin

# panSea.Top = -panSea.Height- 128 - 17 - scrlVert.Position;

# -17 to move the panel above the Horizontal scroll bar #

# end;

return

def WW2Rules.scrlHorizChange(Sender: TObject):

begin

# panSea.Left = 0 - scrlHoriz.Position; #

# end;

return

def WW2Rules.btnExtraDamageClick(Sender: TObject):

begin

panExtraDamage.Visible = TRUE;

panExtraDamage.Top = 25;

panExtraDamage.Left = 0;

panAssignMovement.Visible = FALSE;

panFirer.Visible = FALSE;

panTarg1.Visible = FALSE;

panTarg2.Visible = FALSE;

panTarg3.Visible = FALSE;

panTarg4.Visible = FALSE;

btnApplyExtraDamage.Enabled = FALSE;

# select first target in the combo box #

cboExtraDamageTarget.ItemIndex = 0;

# end;

def WW2Rules.btnAssignExtraDamageClick(Sender: TObject):

begin

if cboExtraDamageTarget.ItemIndex == -1:

begin

Application.MessageBox( 'Please Select Target for Extra Damage' , 'Extra Damage', MB\_OKCancel);

# end

else:

begin

if medtAmountExtraDamage.Text == '0':

Application.MessageBox( 'Please Enter Valid Extra Damage Value' , 'Extra Damage', MB\_OKCancel)

else:

begin

Vessel = TObject(VesselList.Objects[cboExtraDamageTarget.ItemIndex]) as VesselRecord;

with Vessel do

begin

ExtraDamage = int(medtAmountExtraDamage.Text);

btnApplyExtraDamage.Enabled = TRUE;

# end;

# end;

# end;

# end;

def WW2Rules.btnApplyExtraDamageClick(Sender: TObject):

begin

# Apply all of the Extra Damage Assigned So Far #

ApplyDamage('x');

# Add an X to the end of the move number so that it is obvious that the damage received is EXtra Damage #

btnAssignExtraDamage.Enabled = FALSE;

btnSpecifyMove.Enabled = TRUE;

btnMakeMove.Enabled = TRUE;

# Place the Movement and Firing Panels back in sight #

PlacePanels(1); # Shows the movement panel #

# end;

def WW2Rules.rdgTarg1SmokeClick(Sender: TObject):

begin

TargSmokeChanged(0);

# end;

def WW2Rules.rdgTarg2SmokeClick(Sender: TObject):

begin

TargSmokeChanged(1);

# end;

def WW2Rules.rdgTarg3SmokeClick(Sender: TObject):

begin

TargSmokeChanged(2);

# end;

def WW2Rules.rdgTarg4SmokeClick(Sender: TObject):

begin

TargSmokeChanged(3);

# end;

def WW2Rules.edtLabelMultChange(Sender: TObject):

begin

labelMult = IntDef(edtLabelMult.Text, 3);

# set the Graphical Multiplier to the new value in the edit box, defaulting to 3 #

RefreshBackground;

RefreshAllGraphics;

# end;

def WW2Rules.pagCtrlChange(Sender: TObject):

begin

if pagCtrl.ActivePage == tabMap:

RefreshAllGraphics;

# end;

def WW2Rules.FormPaint(Sender: TObject):

begin

if pagCtrl.ActivePage == tabMap:

RefreshAllGraphics;

# end;

def WW2Rules.ptbSeaMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer):

begin

if pagCtrl.ActivePage == tabMap:

if ssLeft in Shift:

begin

bitmapLeft = max(min(0, bitmapLeft + X - cursorX), ptbSea.Width - background.Width);

bitmapTop = max(min(0, bitmapTop + Y - cursorY), ptbSea.Height - background.Height);

RefreshAllGraphics;

cursorX = X;

cursorY = Y;

# end;

# end;

def WW2Rules.ptbSeaMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer):

begin

if pagCtrl.ActivePage == tabMap:

if ssLeft in Shift:

begin

cursorX = X;

cursorY = Y;

# end;

# end;

def WW2Rules.IncrementMagnification1Click(Sender: TObject):

var

iUnmultTop, iUnmultLeft: integer;

# contains the top and left positions when divided by the labelMult #

begin

iUnmultTop = (ptbSea.Height div 2 - bitmapTop) div labelMult;

iUnmultLeft = (ptbSea.Width div 2 - bitmapLeft) div labelMult;

if labelMult < 7:

begin

labelMult = labelMult + 1;

# code resets the old middle of the screen to be the new middle of the screen #

bitmapTop = max(min(0, (ptbSea.Height div 2 - iUnmultTop) \* labelMult), ptbSea.Height - \

background.Height);

bitmapLeft = max(min(0, (ptbSea.Width div 2 - iUnmultLeft) \* labelMult), ptbSea.Width - \

background.Width);

RefreshBackground;

RefreshAllGraphics;

# end;

# end;

def WW2Rules.DecrementMagnification1Click(Sender: TObject):

var

iUnmultTop, iUnmultLeft: integer;

# contains the top and left positions when divided by the labelMult#

begin

iUnmultTop = (ptbSea.Height div 2 - bitmapTop) div labelMult;

iUnmultLeft = (ptbSea.Width div 2 - bitmapLeft) div labelMult;

if labelMult > 1:

begin

labelMult = labelMult - 1;

# code resets the old middle of the screen to be the new middle of the screen #

bitmapTop = max(min(0, (ptbSea.Height div 2 - iUnmultTop) \* labelMult), ptbSea.Height - \

background.Height);

bitmapLeft = max(min(0, (ptbSea.Width div 2 - iUnmultLeft) \* labelMult), ptbSea.Width - \

background.Width);

RefreshBackground;

RefreshAllGraphics;

# end;

# end;

def WW2Rules.btnNil1Click(Sender: TObject):

begin

NullifyTarget(0);

# end;

def WW2Rules.btnNil2Click(Sender: TObject):

begin

NullifyTarget(1);

# end;

def WW2Rules.btnNil3Click(Sender: TObject):

begin

NullifyTarget(2);

# end;

def WW2Rules.btnNil4Click(Sender: TObject):

begin

NullifyTarget(3);

# end;

# end.