Applying Visual Basic Express 2005 to
Game Programming with Visual Basic.NET

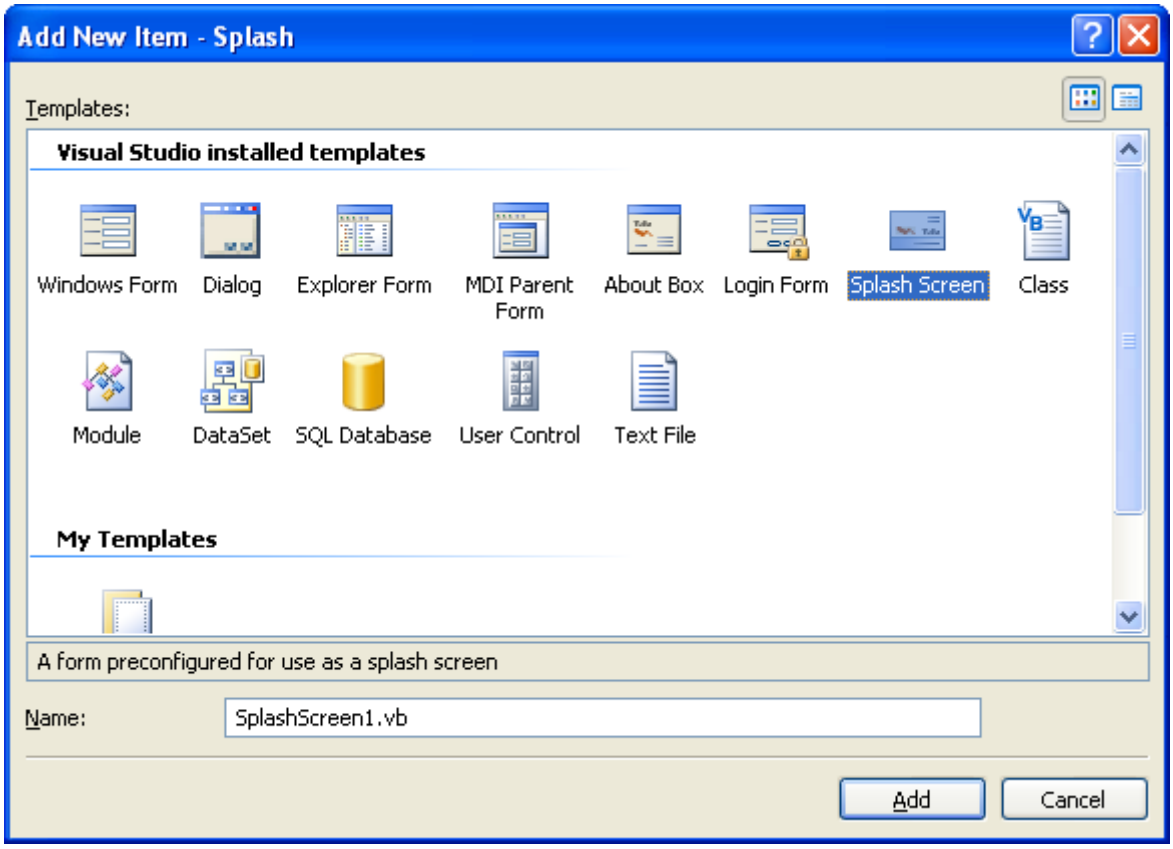
Chapter 3

The module *Bearing.vb* needs some small adjustments. The keyword 'Overloads' has to be removed from each of the functions. See the adjusted code in Appendix 1.

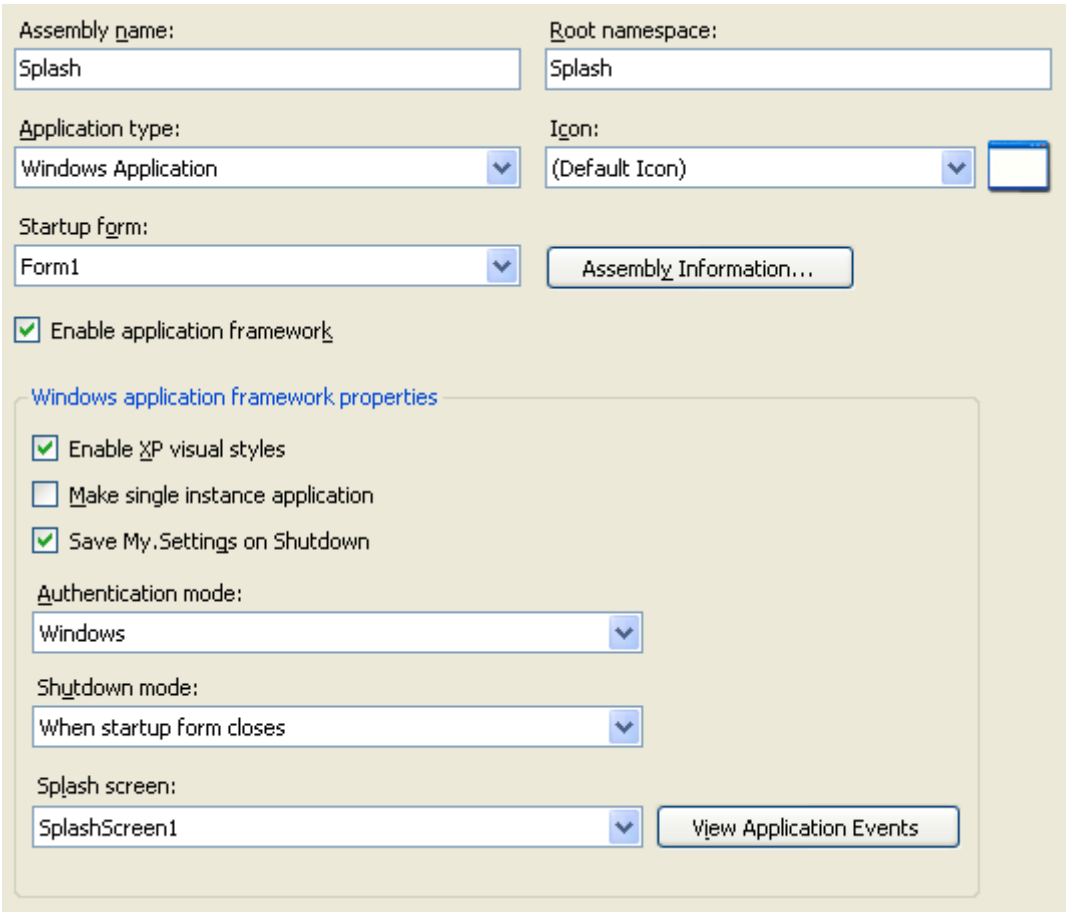
Chapter 4

In the program example *Splash* there is no need for a timer on the splash form, nor is there any need for the code module with the procedure S*ub Main*. Visual Basic Express provides a simple way of including a splash form.

From the Project menu select Add a Windows Form… Choose a Splash Screen as shown below:



Check the Project properties as shown on the following page, the splash form is then operational.

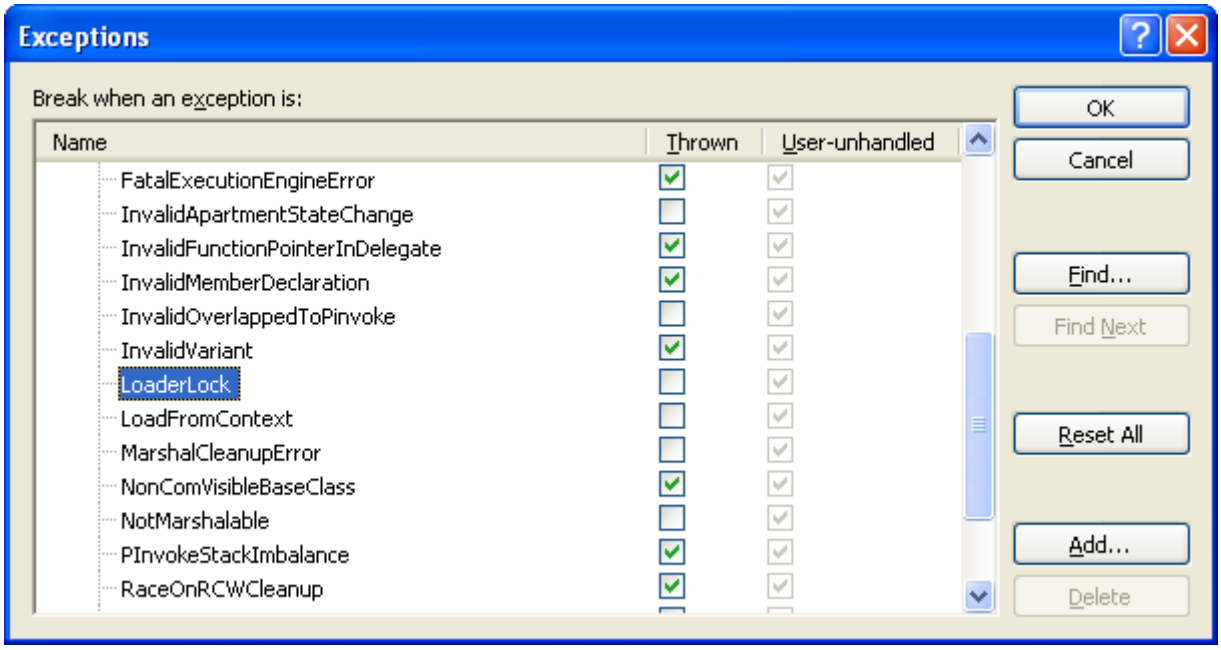


Chapter 6

Just one addition is required in the first line of code in the following procedure:

```
Private Sub btnOpen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
Handles btnOpen.Click
   If dlgOpen.ShowDialog() = Windows.Forms.DialogResult.OK Then
      SoundFile = dlgOpen.FileName
      btnPlay.Enabled = True
      If rdbDirectX.Checked Then  ' DirectX
         DirectXSound = New Audio(SoundFile)
         lblDuration.Text = CInt(DirectXSound.Duration) ' show total play time
      ElseIf rdbWmp.Checked Then  ' Windows media player
         WmpSound = New WindowsMediaPlayerClass
         WmpSound.autoStart = False ' waits for the Play method
         WmpSound.URL = SoundFile ' assign the file to play
         WmpSound.mediaCollection.add(SoundFile) ' required to access duration
         lblDuration.Text = CInt(WmpSound.currentItem.duration) ' show total play time
      End If
   End If
   btnStop.Enabled = False
End Sub
```

A setting needs to be made for DirectX 9. Go to the Debug menu and Choose Exceptions... Expand the Managed Debugging Assistants then scroll down till you find LoaderLock. Uncheck it as shown on the following page:



Chapter 7

For all applications using the module *Collision.vb* a small adjustment needs to be made to each of the functions. The keyword 'Overloads' has to be removed from each of the functions. In the module *Positioning.vb* the procedure *CentreOn* is now one procedure and not two overloads. See the adjusted code in Appendix 1.

Chapter 9

For all applications using a moving background a certain form property must be set to avoid a flicker each time the background changes. The DoubleBuffered property must be set to *True*.

Chapter 10

In the module *Positioning.vb* the procedure *CentreOn* is now one procedure and not two overloads. See the adjusted code in Appendix 1.

Chapter 14

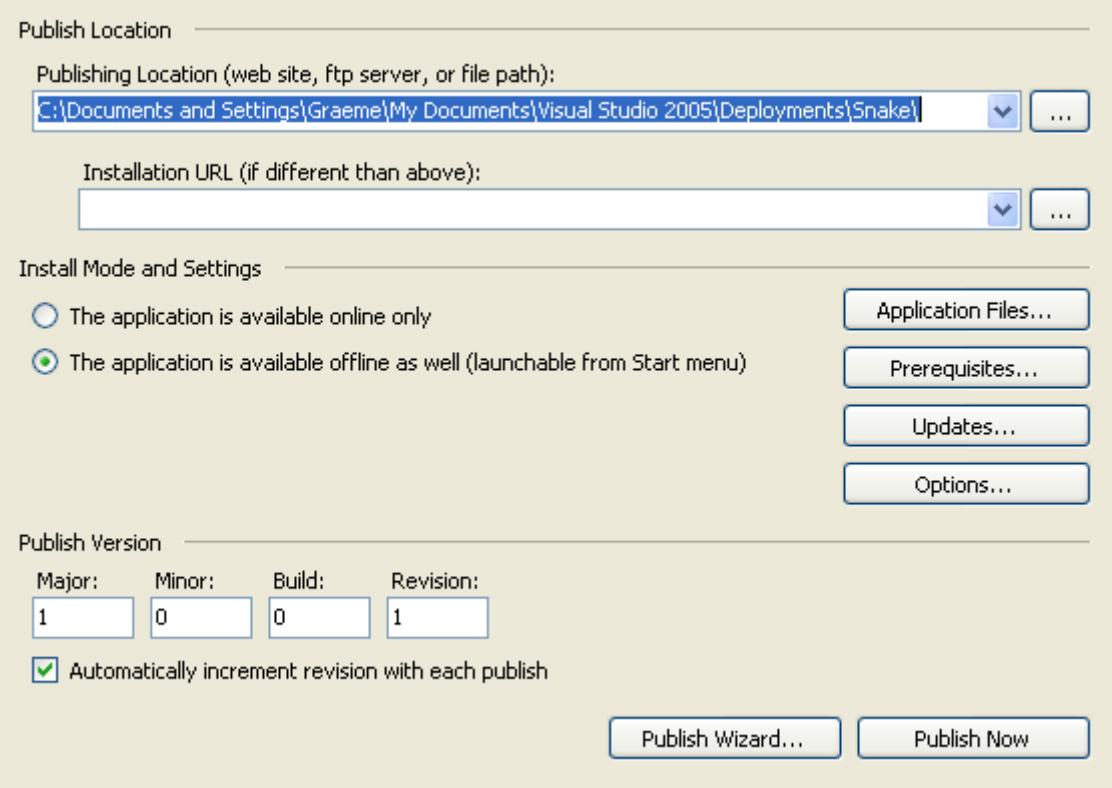
For all applications using the module *Collision.vb* a small adjustment needs to be made to each of the functions. The keyword 'Overloads' has to be removed from each of the functions. In the module *Positioning.vb* the procedure *CentreOn* is now one procedure and not two overloads. See the adjusted code in Appendix 1.

Chapter 15

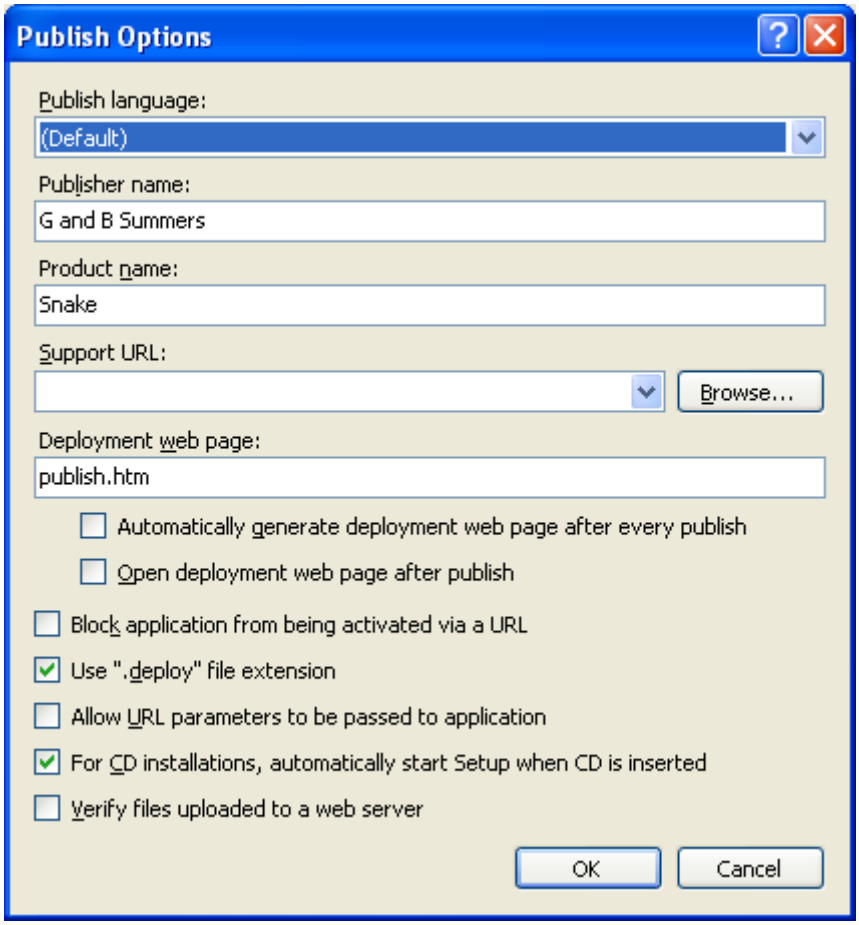
Visual Basic Express does not have a 'Setup and Deployment Project' as the project type, nor does it have a Setup Wizard. However there are simple means to allow the deployment of applications.

The Snake application will be used as an example as it is in 'Game Programming…':

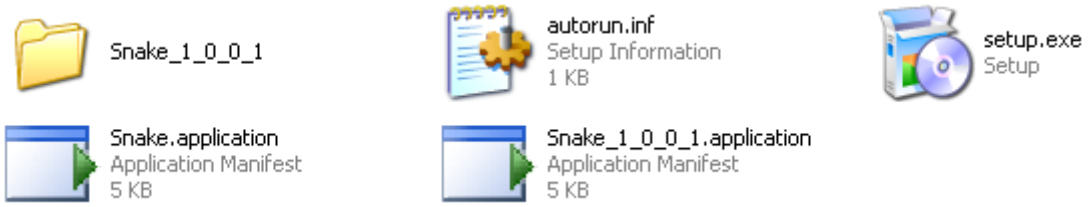
Firstly create a folder to be used for publishing applications. In the Snake project set this path in the Publish tab of the project properties:



Choose Options... and make settings similar to those below:



Click on the Publish Wizard… button and, in a few short steps, a deployment package will be created in the location specified.



Appendix 1

Collision.vb

```
Module Collision

   Public Function Collision(ByVal Obj1 As Object, ByVal Obj2 As Object,_
   Optional ByVal HorizTolerance1 As Integer = 0, Optional ByVal VertTolerance1 As_
   Integer = 0, Optional ByVal HorizTolerance2 As Integer = 0, Optional ByVal _
   VertTolerance2 As Integer = 0) As Boolean
      If Obj1 Is Nothing Or Obj2 Is Nothing Then
         Return False
      Else
         Return (Obj1.Top + Obj1.Height - VertTolerance1 >= Obj2.Top + _
         VertTolerance2) And (Obj2.Top + Obj2.Height - VertTolerance2 >= Obj1.Top _
         + VertTolerance1) And (Obj1.Left + Obj1.Width - HorizTolerance1 >= _
         Obj2.Left + HorizTolerance2) And (Obj2.Left + Obj2.Width - _
         HorizTolerance2 >= Obj1.Left + HorizTolerance1) And Obj1.Visible _
         And Obj2.Visible
      End If
   End Function

   Public Function Collision(ByVal Rect As Rectangle, ByVal Obj As _
   Object, Optional ByVal HorizToleranceRect As Integer = 0, Optional ByVal _
   VertToleranceRect As Integer = 0, Optional ByVal HorizToleranceObj As Integer _
   = 0, Optional ByVal VertToleranceObj As Integer = 0) As Boolean
      If Obj Is Nothing Then
         Return False
      Else
         Return (Rect.Top + Rect.Height - VertToleranceRect >= Obj.Top + _
         VertToleranceObj) And (Obj.Top + Obj.Height - VertToleranceObj >= _
         Rect.Top + VertToleranceRect) And (Rect.Left + Rect.Width - _
         HorizToleranceRect >= Obj.Left + HorizToleranceObj) And (Obj.Left + _
         Obj.Width - HorizToleranceObj >= Rect.Left + HorizToleranceRect) _
         And Obj.Visible
      End If
   End Function

   Public Function Collision(ByVal Rect1 As Rectangle, ByVal Rect2 As _
   Rectangle, Optional ByVal HorizTolerance1 As Integer = 0, Optional ByVal _
   VertTolerance1 As Integer = 0, Optional ByVal HorizTolerance2 As Integer = 0, _
   Optional ByVal VertTolerance2 As Integer = 0) As Boolean
      Return (Rect1.Top + Rect1.Height - VertTolerance1 >= Rect2.Top + _
      VertTolerance2) And (Rect2.Top + Rect2.Height - VertTolerance2 >= Rect1.Top _
      + VertTolerance1) And (Rect1.Left + Rect1.Width - HorizTolerance1 >= _
      Rect2.Left + HorizTolerance2) And (Rect2.Left + Rect2.Width - _
      HorizTolerance2 >= Rect1.Left + HorizTolerance1)
   End Function
```

```
   Public Function Collision(ByVal Loc As Point, ByVal Obj As Object, _
   Optional ByVal HorizTolerance As Integer = 0, Optional ByVal VertTolerance As _
   Integer = 0) As Boolean
      If Obj Is Nothing Then
         Return False
      Else
         Return Loc.X >= Obj.Left + HorizTolerance And Loc.X <= Obj.Left + _
         Obj.Width - HorizTolerance And Loc.Y >= Obj.Top + VertTolerance And _
         Loc.Y <= Obj.Top + Obj.Height - VertTolerance And Obj.Visible
      End If
   End Function

Public Function Collision(ByVal Loc As Point, ByVal Rect As _
   Rectangle, Optional ByVal HorizTolerance As Integer = 0, Optional ByVal _
   VertTolerance As Integer = 0) As Boolean
      Return Loc.X >= Rect.Left + HorizTolerance And Loc.X <= Rect.Left + _
      Rect.Width - HorizTolerance And Loc.Y >= Rect.Top + VertTolerance And Loc.Y _
      <= Rect.Top + Rect.Height - VertTolerance
   End Function

End Module
```

Bearing.vb

```
Module Bearing

   ' The positive Left direction is an angle of 0
   ' The positive Top direction is an angle of 90 degrees or Pi/2 radians
   '***********************
   '*  OUTPUT IN RADIANS  *
   '***********************
   Public Function DirnRad(ByVal FromObj As Object, ByVal ToObj As Object) As Double
      ' Calculated in radians
      Dim DirnCalc As Double
        Dim FromCentre As New Point(CInt(FromObj.Left + FromObj.Width / 2), _
         CInt(FromObj.Top + FromObj.Height / 2))
        Dim ToCentre As New Point(CInt(ToObj.Left + ToObj.Width / 2), _
         CInt(ToObj.Top + ToObj.Height / 2))
        Dim HorizDist As Integer = ToCentre.X - FromCentre.X
        Dim VertDist As Integer = ToCentre.Y - FromCentre.Y
        DirnCalc = Math.Atan2(CDbl(VertDist), CDbl(HorizDist))
        If DirnCalc < 0 Then
            DirnCalc += Math.PI * 2
        End If
        Return DirnCalc
    End Function

   Public Overloads Function DirnRad(ByVal FromPoint As Point, ByVal ToPoint As Point) _
   As Double
      ' Calculated in radians
      Dim DirnCalc As Double
      Dim HorizDist As Integer = ToPoint.X - FromPoint.X
      Dim VertDist As Integer = ToPoint.Y - FromPoint.Y
      DirnCalc = Math.Atan2(CDbl(VertDist), CDbl(HorizDist))
      If DirnCalc < 0 Then
         DirnCalc += Math.PI * 2
      End If
      Return DirnCalc
   End Function
```

```
Public Function DirnRad(ByVal FromPoint As Point, ByVal ToObj As Object) As Double
   ' Calculated in radians
   Dim DirnCalc As Double
   Dim ToCentre As New Point(CInt(ToObj.Left + ToObj.Width / 2), _
   CInt(ToObj.Top + ToObj.Height / 2))
   Dim HorizDist As Integer = ToCentre.X - FromPoint.X
   Dim VertDist As Integer = ToCentre.Y - FromPoint.Y
   DirnCalc = Math.Atan2(CDbl(VertDist), CDbl(HorizDist))
   If DirnCalc < 0 Then
      DirnCalc += Math.PI * 2
   End If
   Return DirnCalc
End Function

Public Function DirnRad(ByVal FromObj As Object, ByVal ToPoint As Point) As Double
   ' Calculated in radians
   Dim DirnCalc As Double
   Dim FromCentre As New Point(CInt(FromObj.Left + FromObj.Width / 2), _
   CInt(FromObj.Top + FromObj.Height / 2))
   Dim HorizDist As Integer = ToPoint.X - FromCentre.X
   Dim VertDist As Integer = ToPoint.Y - FromCentre.Y
   DirnCalc = Math.Atan2(CDbl(VertDist), CDbl(HorizDist))
   If DirnCalc < 0 Then
      DirnCalc += Math.PI * 2
   End If
   Return DirnCalc
End Function

'***********************
'*  OUTPUT IN DEGREES  *
'***********************
Public Function DirnDeg(ByVal FromObj As Object, ByVal ToObj As Object) As Double
   ' Calculated in degrees
   Dim DirnCalc As Double
   Dim FromCentre As New Point(CInt(FromObj.Left + FromObj.Width / 2), _
   CInt(FromObj.Top + FromObj.Height / 2))
   Dim ToCentre As New Point(CInt(ToObj.Left + ToObj.Width / 2), _
   CInt(ToObj.Top + ToObj.Height / 2))
   Dim HorizDist As Integer = ToCentre.X - FromCentre.X
   Dim VertDist As Integer = ToCentre.Y - FromCentre.Y
   DirnCalc = Math.Atan2(CDbl(VertDist), CDbl(HorizDist)) * 180 / Math.PI
   If DirnCalc < 0 Then
      DirnCalc += 360
   End If
   Return DirnCalc
End Function

Public Function DirnDeg(ByVal FromPoint As Point, ByVal ToPoint As Point) As Double
   ' Calculated in degrees
   Dim DirnCalc As Double
   Dim HorizDist As Integer = ToPoint.X - FromPoint.X
   Dim VertDist As Integer = ToPoint.Y - FromPoint.Y
   DirnCalc = Math.Atan2(CDbl(VertDist), CDbl(HorizDist)) * 180 / Math.PI
   If DirnCalc < 0 Then
      DirnCalc += 360
   End If
   Return DirnCalc
End Function
```

```
   Public Function DirnDeg(ByVal FromPoint As Point, ByVal ToObj As Object) As Double
      ' Calculated in degrees
      Dim DirnCalc As Double
      Dim ToCentre As New Point(CInt(ToObj.Left + ToObj.Width / 2), _
      CInt(ToObj.Top + ToObj.Height / 2))
      Dim HorizDist As Integer = ToCentre.X - FromPoint.X
      Dim VertDist As Integer = ToCentre.Y - FromPoint.Y
      DirnCalc = Math.Atan2(CDbl(VertDist), CDbl(HorizDist)) * 180 / Math.PI
      If DirnCalc < 0 Then
         DirnCalc += 360
      End If
      Return DirnCalc
   End Function

   Public Function DirnDeg(ByVal FromObj As Object, ByVal ToPoint As Point) As Double
      ' Calculated in degrees
      Dim DirnCalc As Double
      Dim FromCentre As New Point(CInt(FromObj.Left + FromObj.Width / 2), _
      CInt(FromObj.Top + FromObj.Height / 2))
      Dim HorizDist As Integer = ToPoint.X - FromCentre.X
      Dim VertDist As Integer = ToPoint.Y - FromCentre.Y
      DirnCalc = Math.Atan2(CDbl(VertDist), CDbl(HorizDist)) * 180 / Math.PI
      If DirnCalc < 0 Then
         DirnCalc += 360
      End If
      Return DirnCalc
   End Function

End Module
```

Positioning.vb

```
Module Positioning

Public Sub CentreOn(ByVal ObjOnWhichToCentre As Object, ByRef ObjToCentre As Object)
   ' Centre on a point
   If TypeOf ObjOnWhichToCentre Is Point Then
      ObjToCentre.Location = New Point(ObjOnWhichToCentre.X - ObjToCentre.Width \ 2, _
      ObjOnWhichToCentre.Y - ObjToCentre.Height \ 2)
   Else ' Centre on an object
      Dim WidthDiff As Integer = ObjOnWhichToCentre.Width - ObjToCentre.Width
      Dim HeightDiff As Integer = ObjOnWhichToCentre.Height - ObjToCentre.Height
      ObjToCentre.Location = New Point(ObjOnWhichToCentre.Left + WidthDiff \ 2, _
      ObjOnWhichToCentre.Top + HeightDiff \ 2)
   End If
End Sub

Public Function InSightsVert(ByVal Attacker As Object, ByVal Target As Object) _
   As Boolean
      ' in a vertical line above or below
      Return (Target.Left >= Attacker.Left And Target.Left < (Attacker.Left + _
      Attacker.Width) Or Attacker.Left >= Target.Left And Attacker.Left < _
      (Target.Left + Target.Width)) And Attacker.Visible And Target.Visible
   End Function

   Public Function InSightsHoriz(ByVal Attacker As Object, _
   ByVal Target As Object) As Boolean
      ' in a horizontal line to the left or right
      Return (Target.Top >= Attacker.Top And Target.Top < (Attacker.Top + _
      Attacker.Height) Or Attacker.Top >= Target.Top And Attacker.Top < _
      (Target.Top + Target.Height)) And Attacker.Visible And Target.Visible
   End Function

End Module
```