Smart Button Project

Een project om een knop meer functionaliteit te koppelen

Auteur: J.A. Kok
Copyright: J.A. Kok (2017)
Website: www.hzns.nl

Inhoud

Inleiding

Algemeen

Plaatsgebrek bij het bouwen van een applicatie om meerdere knoppen (buttons) op te nemen op een formulier (form) leidde tot een nieuw project, de “Smart Button Project” (SBP). In dit project wordt het oppervlak van een knop verdeeld is in zones. Aan elk van deze zones kan functionaliteit worden gekoppeld.

Dit document is verdeeld in theoretisch gedeelte en een praktisch gedeelte. In het eerste deel wordt met een wiskundige blik gekeken naar het idee van de “smart button”. Deze informatie is in beginsel ontwikkel omgeving onafhankelijk. Het tweede deel is praktische implementatie in een Visual Studio Basic.Net omgeving (versie Visual Basic.Net 2015 Express). In bijlage 1 bij dit document treft u integraal de VB.Net code aan.

Dit document is geen kookboek (volg het recept en u komt er altijd uit), maar meer een ideeënboek met hints en misschien wat tricks.

De Theorie

Analyse

Een nadere beschouwing leert dat er twee aspecten uitgewerkt moeten worden, namelijk hoe verdeel ik het oppervlak van de knop in zones en hoe weet ik op welke zone geklikt is. Het coördinatensysteem, dat gebruikt wordt op locaties op het scherm aan te duiden, is het instrument voor zowel het creëren van zones als het vaststellen van plek van de “muisklik”.

Coördinatensysteem

Iedere ontwikkelomgeving maakt gebruik van een coördinatensysteem voor beschrijven van grafische elementen op een scherm (screen), waarbij de oorsprong meestal in de linker boven hoek is gelegen met een positieve X-as van links naar rechts en een positieve Y-as van boven naar onder. Hoeken worden gedefinieerd in graden, waarbij nul graden recht omhoog is en de telling met de klok mee gaat. Eén en ander wijkt af het cartesisch coördinatensysteem (zie ook Aandachtspunten bij het werken met sinus, cosinus en graden), dat we gewend zijn van uit de wiskunde.

Meestal worden punten in het systeem aangeduid met een X- en een Y-coördinaat. Het is echter ook mogelijk een punt aan te duiden met een richting en een afstand ten opzichte van een referentie punt. In dit geval spreken we van een poolcoördinaat.

Grafische elementen maken vaak van een eigen “intern” coördinatensysteem gebruik, waarbij meestal de linker bovenhoek van het element de oorsprong is. Houd er rekening mee dat niet altijd het geval is. Bij Visual Studio van Microsoft moet bij het werken met containers (zoals bijvoorbeeld een formulier) rekening worden gehouden met het kader en de titelbalk van die container.

Waar is de muis

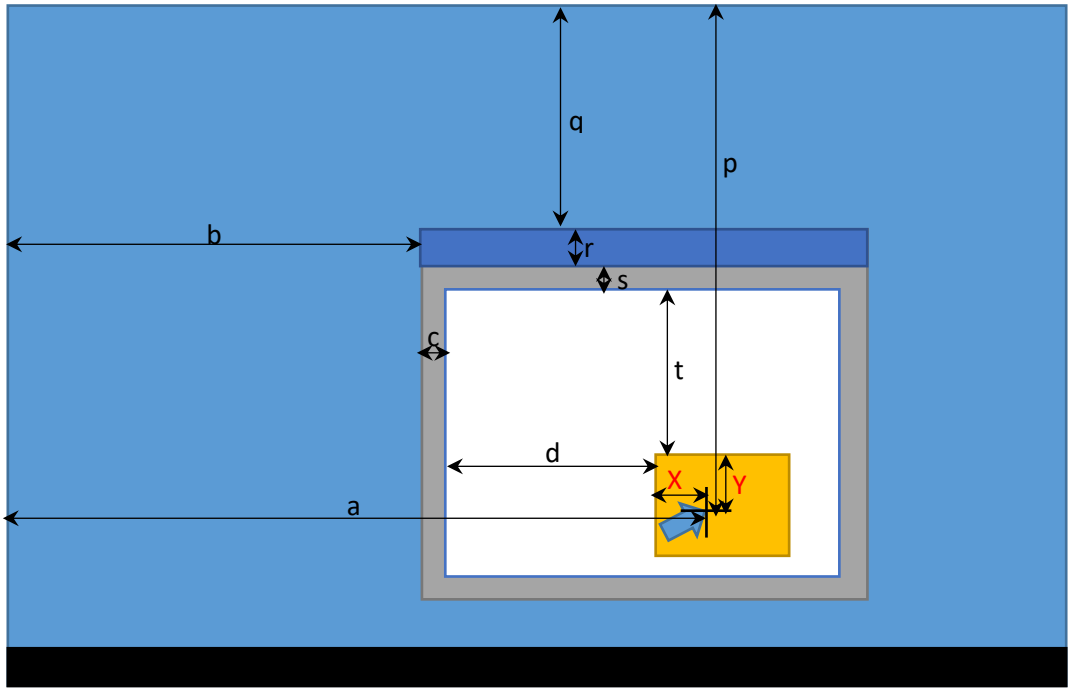
Algemeen

De kernvraag in deze paragraaf is: “Waar op de knop iser met de muis geklikt?” Hoewel het simpel lijkt om te stellen waar de muis was op het moment van klikken, blijken er nog al wat adertjes onder het gras te zitten, deels hier voor aan beschreven. De eerste betreft het feit dat muispositie altijd ten opzichte van he linkerbovenhoek (coördinaat (0,0)) van het scherm wordt gegeven. Als tweede blijkt dat ieder grafisch element op ons scherm intern een eigen coördinatensysteem heeft. Derde factor is dat sommige grafische elementen een “container” zijn voor andere grafische elementen, waarbij het

coördinaat (0,0) van het “container”- coördinatensysteem niet altijd precies in de linkerbovenhoek ligt. En een formulier is zo’n “container”.

Uitwerking

Voor de uitwerking maken we gebruik van de onderstaande schets. Het bestaat uit een scherm (licht blauw), een formulier (blauw (balk), grijs (kader), wit(werkruimte)), een knop (geel) en een assenkruis (plek waar met de muis geklikt is). Het gevraagde coördinaat is de (X,Y) van de muisklik t.o.v. oorsprong van de knop.



Voor de X ($MuisOpKnop_X$) kunnen we de volgende vergelijking opstellen:

$X = a - b - c - d.$

Hierbij komen a overeen met de x van de muisklik t.o.v. oorsprong van het scherm ($MuisOpScherm_X$), b overeen met x van het formulier t.o.v. oorsprong van het scherm ($Formulier_X$), c overeen met de breedte van het kader van het formulier ($Formulier_{KADER}$) en d overeen met x van de knop t.o.v. de werkruimte van het formulier ($Knop_X$). De waarden van $MuisOpScherm_X$, $Formulier_X$, en $Knop_X$ zijn opvraagbare variabelen. $Formulier_{KADER}$ is niet altijd opvraagbaar en zal dan berekend moeten worden. De berekening volgt verderop. Uitgewerkt ziet het geheel er als volgt uit:

$MuisOpKnop_X = MuisOpScherm_X - Formulier_X - Formulier_{KADER} - Knop_X.$

Voor de Y ($MuisOpKnop_Y$) kunnen we een vergelijkbare vergelijking opstellen:

$Y = p - q - r - s - t.$

Hierbij komen p overeen met de y van de muisklik t.o.v. oorsprong van het scherm ($MuisOpScherm_Y$), q overeen met y van het formulier t.o.v. oorsprong van het scherm ($Formulier_Y$), r overeen met de hoogte van het balk van het formulier ($Formulier_{BALK}$),s overeen met de breedte van het kader van het formulier ($Formulier_{KADER}$) en t overeen met y van de knop t.o.v. de werkruimte van het formulier ($Knop_Y$). De waarden van $MuisOpScherm_Y$, $Formulier_Y$, en $Knop_Y$ zijn wederom opvraagbare variabelen. $Formulier_{BALK}$ en $Formulier_{KADER}$ zijn niet altijd opvraagbaar en zal dan berekend moeten worden. De berekening volgt verderop. . Uitgewerkt ziet het geheel er als volgt uit:

$MuisOpKnop_Y = MuisOpScherm_Y - Formulier_Y - Formulier_{BALK} - Formulier_{KADER} - Knop_Y.$

De breedte van het kader ($Formulier_{KADER}$) kan worden berekend vanuit de breedte van het formulier ($Formulier_{BREEDTE}$) en de breedte van de werkruimte van het formulier ($Formulier_{WERKBREEDTE}$). Deze beide laatse variabelen zijn meestal opvraagbaar.

$Formulier_{KADER} = ½ \times (Formulier_{BREEDTE} - Formulier_{WERKBREEDTE})$.

De Hoogte van de balk ($Formulier_{BALK}$) is op een vergelijkbare manier te berekenen, zij het dat er rekening moet worden gehouden met de breedte van het kader.

$Formulier_{BALK} = Formulier_{HOOGTE} - Formulier_{WERKHOOGTE} - 2 \times Formulier_{KADER}$.

Bij deze berekeningen is er van uitgegaan dat het kader rondom even breed is. Ook is er geen rekening gehouden met andere eigenschappen van de randen van een formulier.

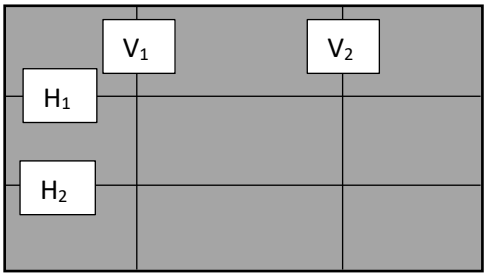
Zonering

Algemeen

De mogelijkheden tot het indelen in zones is oneindig, zolang de grenzen tussen de zones maar wiskundig te beschrijven zijn. In de praktijk zijn een raster-, een circulaire en/of een segment- (taartpunt) indeling goede werkbare opties.

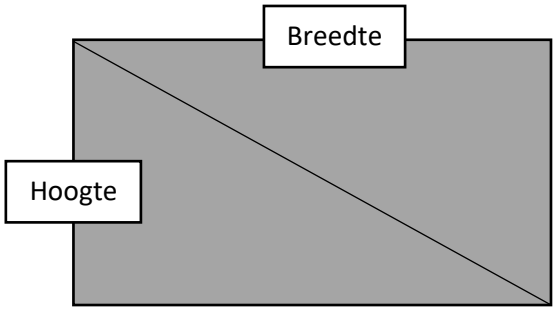
Rasterindeling

Horizontale en/of verticale grenzen. Dit is de eenvoudigste vorm van indeling. De wiskundige beschrijving van de grenzen zijn horizontale (Y heeft een vaste waarde) en verticale lijnen (X heeft een vaste waarde).



In het voorbeeld hiernaast is de voorwaarde voor een muisklik in het middelste vak: $V_1 <= X_{muis} < V_2$ *en* $H_1 <= Y_{muis} < H_2$.

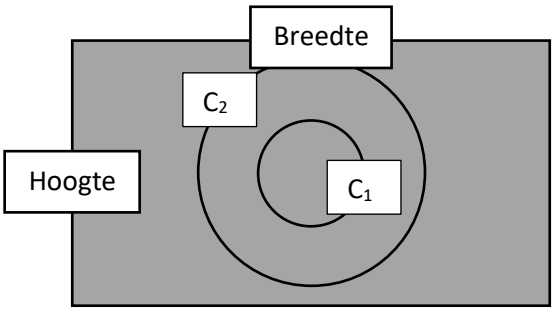
Scheve of gebogen grenzen. Voor een raster indeling met schuine en/of gebogen grenzen moet eerst de wiskundige formule van deze grens worden vastgesteld. Als voorbeeld: voor diagonaal van linksboven naar rechtsonder is de formule *Y = H(oogte) / (B(reedte) * X*. Deze formule kan worden omgeschreven naar *0 = (H / B) * X- Y* [1]. Het rechter gedeelte van deze formule kunnen we als een soort test gebruiken. Indien we een willekeurig punt (de locatie van de muisklik) invullen, kan de T(estwaarde) positief zijn (boven de grens), negatief (onder de grens) of 0 (op de grens). In het voorbeeld: als *T < 0* dan is er linksonder de grens geklikt, Als *T >= 0* dan is er rechtsboven of op de grens geklikt. Voor de wiskundige puristen: de breedte van het oppervlak mag nooit de waarde 0 hebben (omdat hij in de deler staat). Dit zal in de praktijk echter geen probleem zijn, een knop heeft immers altijd een breedte. De methode werkt ook voor gebogen grenzen, echter het het vinden van de juiste wiskundige formule is lastige.



Tenslotte: test vooraf waarde testwaarde positief en negatief is. Eventueel kunt u dan de logica van de uw implementatie hier op aanpassen.

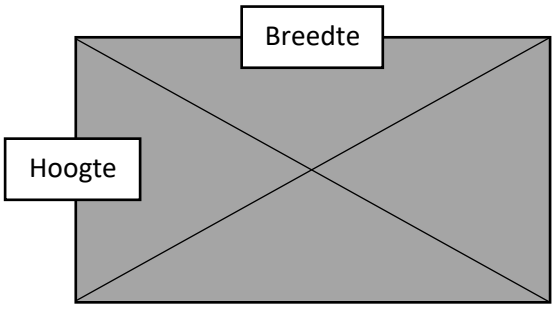
[1] De test-formule voort de andere diagonaal is *0 = - (H / B) * X- Y + H*

Circulaire indeling



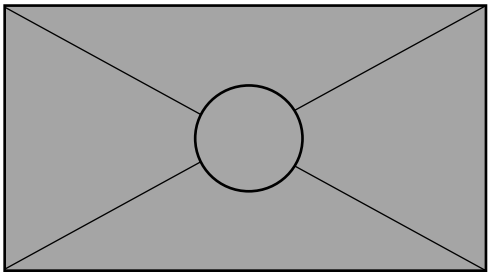
Een circulaire indeling is een zonering gebaseerd op de afstand van de muisklik en een specifiek punt, al of niet gelegen binnen het oppervlak. In het voorbeeld is als punt gekozen voor het centrum (X_{punt} = ½ * *Breedte*, Y_{punt} = ½ * *Hoogte*) van het oppervlak. De afstand is eenvoudig te berekenen met de stelling van Pythagoras: $A(fstand) = ((X_{punt} - X_{muis})^2 + (Y_{punt} - Y_{muis})^2)^{½}$. De voorwaarde voor het gebied tussen de cirkels C_1 en C_2 is: *Afstand* C_1 < *Afstand Muis* < *Afstand* C_2. De hier beschreven methode is ook bruikbaar voor een elliptische zonering, maar is wiskundig wat uitdagender!

Segmentindeling



Een segmentindeling is een zonering gebaseerd op hoek van de muisklik en een referentiepunt. In het voorbeeld is als uitgangspunt gekozen voor het centrum ($X_{referentiepunt}$ = ½ * *Breedte*, $Y_{referentiepunt}$ = ½ * *Hoogte*) van het oppervlak. De grenzen van de segmenten worden gevormd door de diagonalen van het oppervlak. Door de hoeken van deze grenzen (ten opzichte van het centrum) te bepalen met kunt u ze vergelijken met de hoek van het punt waar met de muis is geklikt. De voorwaarde voor het rechter segment wordt dan *Hoek* $_{rechtsboven}$ < *Hoek* $_{muis}$ < *Hoek* $_{rechtsonder}$. Hoeken kunnen niet alleen gedefinieerd in graden maar ook met hun Sinus en Cosinus. (eenvoudiger te bereken). Hiertoe worden eerst de sinus en cosinus berekend naar de linker bovenhoek. Voor het rechter segment geldt nu: *Cosinus* $_{muis}$ < - *Cosinus* $_{linksboven}$[2].

Combinaties



Het is ook mogelijk combinaties van zone-indelingen te maken. Een voorbeeld hiernaast. Belangrijk is dan een goede volgorde van de voorwaarden voor de schillende zones. In he voorbeeld kan het beste eerst de cirkel worden afgehandeld op basis van de afstand tot het centrum en daarna de segmenten.

Hints voor de implementatie

<u>De grenzen</u>. Het is goed om van te voren na te denken tot welke zone de punten op een grens tussen zones behoren. Stel u heeft een knop met een breedte van 200 pixels en u heeft de knop verdeeld in even grote linker en rechter zone. De grens zal dan op 100 pixels liggen. U kiest voor de linker zone. De meest linker pixels hebben een X-coördinaat 0 en de meest rechter 100. Als we het aantal feitelijke pixels gaan tellen, komen we op 101 pixels en blijven er dus 99 pixels voor de rechter zone over. Een ongelijke verdeling dus. De grens meetellen met de rechter zone is dus beter. Deze betere oplossing heeft dan als voorwaarde voor de linker zone: $X_{muis} < X_{grens}$ en voor de rechter zone: $X_{muis} >= X_{grens}$. De oorzaak van dit “fenomeen” is dat het tellen van pixels begint met 0, zowel met de X (horizontaal) als Y (vertikaal). Een vuistregel is de grens aan de rechter- en/of onderzijde van de zone niet meetellen en de grens aan de linker- en/of bovenzijde wel.

[2] Voor het linker segment geldt: *Cosinus* $_{muis}$ < *Cosinus* $_{linksboven}$, voor het boven segment: *Sinus* $_{muis}$ < - *Sinus* $_{linksboven}$ en voor het onder segment: *Sinus* $_{muis}$ > *Sinus* $_{linksboven}$.

Zelf functies creëren. Bij het werken met segmenten en cirkels kan het handig zijn te werken met zelfgebouwde functies. Zelf gebruik ik de volgende functie:

Input: X_{punt}, Y_{punt}, $X_{referentiepunt}$, $Y_{referentiepunt}$

Output: $Hoek_{(punt, referentiepunt)}$, $Afstand_{(punt, referentiepunt)}$, $Cosinus_{(punt, referentiepunt)}$, $Sinus_{(punt, referentiepunt)}$,

Wiskundigen onder u zullen hier een knipoog naar de vectormeetkunde in zien. Een praktische uitwerking voor VB.Net vindt u in het praktijk deel van dit document.

De praktijk met Visual Basic .Net (2015 Express)

Algemeen

Alle codevoorbeelden zijn geschreven in Visual studio Basic.Net 2015 Express. Dit is een versie die Microsoft kosteloos beschikbaar stelt, echter met beperkingen voor onder andere commercieel gebruik. Het commentaar in de code is in het Engels geschreven zodat deze bruikbaar en leesbaar is voor een breder publiek. Deze code is doorspekt met commentaar regels. Dit zijn enerzijds geheugensteuntjes voor mij zelf (ter voorkoming van de vraag: “Wat/waarom heb ik dit gedaan?”) en anderzijds om de code toegankelijk te maken voor derden. De code voorbeelden in dit document zijn kopieerbaar naar VB.Net. De kleuren van de code komen overeen een met de ontwikkel omgeving van Visual Studio.

In het praktijkgedeelte worden eerste een aantal methodes uitgewerkt om zones te beschrijven (inbegrepen VB.Net code). Vervolgens worden een drietal voorbeelden uitgewerkt. Van deze voorbeelden is zowel de code als een uitvoerbaar bestand beschikbaar op mijn site www.hzns.nl.

Coördinaten

Voor het werken met pixel-coördinaten kan het handig zijn een nieuw type variabele te introduceren. Een tweetal kenmerken zijn van belang: de pixel-coördinaten zijn altijd gehele getallen (en dus integer) en zijn altijd kleiner dan 32000 (en dus 16 bits). De VB.NET oplossing ziet er dus als volgt uit:

```
'Variable structure for handling pixel-coordinates
Structure Coordinate
    Dim X As Short
    Dim Y As Short
End Structure
```

Oudere versies van VB hebben een andere syntax voor `Structure` en `Short`. Gebruik in plaats daar van `Type` en `Integer`.

Naast “gewone” coördinaten kan gebruik worden gemaakt van poolcoördinaten. Deze hebben de volgende structuur (hoek, straal(/afstand), cosinus en sinus):

```
'Variable structure for handling pixel-polar coordinates
Structure PolarCoordinate
    Dim Arc As Double
    Dim Radius As Double
    Dim Cosinus As Double
    Dim Sinus As Double
End Structure
```

In tegenstelling tot de gewone coördinaten zijn “gebroken” waarden mogelijk. Bij goniometrische functies wordt, bij van 90 graden of veelvouden daar van, gewerkt met hele kleine waarden voor de sinus of cosinus. Derhalve is het variabel type `Double` op zijn plaats.

Muispositie

Het uiteindelijke doel is het vinden van het coördinaat van de muisklik ten opzichte van de rechter bovenhoek van de knop. Hiervoor gebruiken we het schets zoals gegeven in de theorie. De Naam van de knop is `SmartButton`.

Muispositie t.o.v. scherm

De muispositie ten opzichte van de rechter bovenhoek van het scherm kan worden opgevraagd met de functie `MousePosition`. Dit zijn de waarden a ($MuisOpScherm_X$) en p ($MuisOpScherm_Y$)

```
'Mouse position on screen
 Dim ScreenMouse As Coordinate
 ScreenMouse.X = MousePosition.X
 ScreenMouse.Y = MousePosition.Y
```

Formulierpositie t.o.v. Scherm

De X en Y van het formulier (waarden b ($Formulier_X$) en q ($Formulier_Y$) uit schets) zijn een eigenschap van het formulier.

```
'Form topleftcorner on screen
 Dim FormTopLeftCorner As Coordinate
 FormTopLeftCorner.X = Me.Left
 FormTopLeftCorner.Y = Me.Top
```

Formulier kader

De volgende stap is het bepalen van de afmetingen van het kader van het formulier. Als eerste definiëren we een variabele daarvoor. Hierbij is `Border` de breedte van de rand en `Bar` de hoogte van de balk van het formulier.

```
'Variable structure for handling the bar and border size of a container
Structure FrameSize
    Dim Border As Short
    Dim Bar As Short
End Structure
```

Ik kan twee methoden aanbevelen. De eerste is een meer algemene en de tweede is er één specifiek voor het werken met formulieren. Beide worden beschreven in de vorm van een functie.

Voor de eerste methode moeten de een viertal parameter in de functie worden opgegeven. Dit betreft:
- de buitenbreedte van formulier `OuterWidth` (opvraagbaar als `<FormName>.Width`)
- de buitenhoogte van formulier `OuterHeight` (opvraagbaar als `<FormName>.Height`)
- de binnenbreedte van formulier `InnerWidth` (opvraagbaar als `<FormName>.ClientSize.Width`)
- de binnenhoogte van formulier `InnerHeight` (opvraagbaar als `<FormName>.ClientSize.Height`)

```
'Function calculates border- and barsize for rectangular objects
 Function ObjectFrame(OuterWidth As Integer, OuterHeight As Integer,
     InnerWidth As Integer, InnerHeight As Integer) As FrameSize
    'Internal function variable
     Dim Result As FrameSize

    'Calculating border and bar size (=Result)
     Result.Border = (OuterWidth - InnerWidth) / 2
     Result.Bar = OuterHeight - InnerHeight - 2 * Result.Border

    'Returning function value
     Return Result
 End Function
```

Voorbeeld voor gebruik

```
'Variable on module level
 Dim FormFrame As Framesize

'Forms border- and barsize are calculated by using the function ObjectFrame
 FormFrame = ObjectFrame(Me.Width, Me.Height, Me.ClientSize.width, _
             Me.ClientSize.Height)
```

Bij de tweede methode wordt het formulier als parameter opgegeven:

```
'Function calculates border- and barsize for forms
 Function Frame(Element As Form) As FrameSize
    'Internal function variable
     Dim Result As FrameSize

    'Calculating border and bar size (=Result)
     Result.Border = (Element.Width - Element.ClientSize.Width) / 2
     Result.Bar = Element.Height - Element.ClientSize.Height - 2 * Result.Border

    'Returning function value
     Return Result
 End Function
```

Voorbeeld voor gebruik

```
'Variable on module level
 Dim FormFrame As Framesize

'Forms border- and barsize are calculated by using the function Frame
 FormFrame = Frame(Me)
```

Beide functies leveren een zelfde resultaat op, namelijk `Frame.Border` voor de waarden c en r ($Formulier_{KADER}$) en `FormFrame.Bar` voor de waarde s ($Formulier_{BALK}$).

Knop Positie t.o.v. Werkbare ruimte van het formulier

De linkerbovenhoek positie van de knop (waarden d ($Knop_X$) en t ($Knop_Y$) uit schets) is als eigenschap van de knop opvraagbaar. Deze zijn gedefinieerd t.o.v. werkbare ruimte (ClientSize).

```
'Position topleftcorner of button based on clientsize form
 Dim ButtonTopLeftCorner As Coordinate
 ButtonTopLeftCorner.X = SmartButton.Left
 ButtonTopLeftCorner.Y = SmartButton.Top
```

Eindresultaat

Het uitwerken van de formule uit de theorie ziet er dan als volgt uit:

```
'Position mouse on SmartButton
 Dim ButtonMouse As Coordinate
'Calculate relative position mouse on the SmartButton
 ButtonMouse.X = ScreenMouse.X - FormTopLeftCorner.X – _
                 FormFrame.Border - ButtonTopLeftCorner.X
 ButtonMouse.Y = ScreenMouse.Y - FormTopLeftCorner.Y - _
                 FormFrame.Border - FormFrame.Bar - ButtonTopLeftCorner.Y
```

Zones door middel van een raster

In een raster worden de zones gescheiden door verticale en horizontale (grens)lijnen. Verticale zone grenzen hebben een “vast” X-coördinaat. In code kan dit er als volgt uitzien:

```
'Zone left side of the border
 ButtonMouse.X < VerticalZoneBorder
'zone right side of the border
 ButtonMouse.X >= VerticalZoneBorder
```

De grens (border) wordt meegenomen met rechter zone (zoals in de theorie).

Voor de horizontale grenzen geld min of meer het zelfde

```
'Zone above the border
 ButtonMouse.Y < HorizontalZoneBorder
'zone below the border
 ButtonMouse.Y >= HorizontalZoneBorder
```

In het tweede voorbeeld wordt een raster met zowel horizontale als verticale grenzen verder uitgewerkt.

Scheve lijnen of wiskundige formules voor grens

Indien een (grens)lijn scheefloopt (of wordt omschreven met een wiskundige formule) moeten we gaan werken met een testwaarde (`TestValue`). Als voorbeeld de simpele lijn met de wiskundige formule *Y=X*. Als deze wordt omgeschreven (zoals in de theorie) naar de testwaarde structuur ontstaat de formule: *Testwaarde = X - Y*. Onderstaand een code voorbeeld (beslis-logica) voor het testen boven, op of onder de grens lijn.

```
'Internal variable
 Dim TestValue As Short

'Calculating TestValue
 TestValue = MouseButton.X - MouseButton.Y

'Determinating above, on or below the border
 If TestValue > 0 then
     MsgBox ("Above border",,)
 ElseIf TestValue  = 0 then
     MsgBox ("On border",,)
 Else
     MsgBox ("Below border",,)
 EndIf
```

Zones door middel van cirkels

Een cirkel wordt in de wiskunde omschreven als een verzameling punten die een bepaalde afstand hebben tot een vast punt (*Referentiepunt*). De afstand (*Straal*) tussen twee punten (bijvoorbeeld het *Referentiepunt* en de plaats waar geklikt is met de muis) kan worden berekend door middel van de stelling van Pythagoras. Een vertaling in code kan er als volgt uitzien:

Een functie om de *straal* te bepalen:

```
Function CreateRadius(Point As Coordinate,  Referencepoint As Coordinate) As Double
   'Internal function variable
    Dim Result As Double

   'Calculating Radius (= Result)
    Result = ((Point.X - Referencepoint.X) ^ 2 + _
        (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5

   'Returning function value
    Return Result
End Function
```

De beslis-logica:

```
'Variable on module level
 Dim MouseClick As Coordinate
 Dim Center As Coordinate
 Dim RadiusBorder As Short
 Dim RadiusMouse As Double

'Calculating Radius
 RadiusMouse = CreateRadius(MouseClick, Center)

' Determinating in-, on or outside the circle
 If RadiusMouse > RadiusBorder Then
     MsgBox("Outside circle",,)
 ElseIf RadiusMouse = RadiusBorder Then
     MsgBox("On circle",,)
 Else
     MsgBox("Inside circle",,)
 End If
```

In het eerste voorbeeld verder op in dit document wordt een cirkelvormige zone gebruikt rondom het middelpunt van de knop.

Zones door middel van Segmenten

Segmenten kunnen worden beschreven als een taartpunt waarvan de grenzen worden bepaalt door de hoek tot een bepaald vast punt (*Referentiepunt*). Door deze grens-hoeken te vergelijken met de hoek naar de locatie van de muisklik tot het referentiepunt kunnen we bepalen of de muisklik in binnen het segment valt of niet.
Hoeken kunnen op verschillende manieren worden beschreven. In dit document worden de methodes met behulp van sinus/cosinus en met behulp van graden uitgewerkt.

Sinus en cosinus

De sinus en de cosinus worden in de wiskunde beschreven met behulp een rechthoekige driehoek. De sinus als “overstaande zijde” gedeeld door de “schuine zijde” en de cosinus als “aanliggende zijde” gedeeld door de “schuine zijde”. Bij een praktische implementatie kunnen de “schuine zijde” worden vertaald al de afstand tussen het centrum van de segmenten (*Referentiepunt*) en een specifiek punt (bijvoorbeeld de muisklik (*Muisklik*)), de “aanliggende zijde” het verschil in X-coördinaat tussen beide punten ($Muisklik_X$ - $Referentiepunt_X$) en de “overstaande zijde” het verschil in Y-coördinaat tussen beide punten ($Muisklik_Y$ - $Referentiepunt_Y$). In code zouden de volgende functies gebuikt kunnen worden:

```
Function CreateSinus(Point As Coordinate, Referencepoint As Coordinate) As Double
   'Internal function variable variables
    Dim Result As Double
    Dim Radius As Double

   'Calculating Radius and Sinus (= result)
    Radius = ((Point.X - Referencepoint.X) ^ 2 + _
        (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
    Result = (Point.Y - Referencepoint.Y) / Radius

   'Returning function value
    Return Result
End Function

Function CreateCosinus(Point As Coordinate, Referencepoint As Coordinate) As Double
   'Internal function variable variables
    Dim Result As Double
```

```
    Dim Radius As Double

   'Calculating Radius and Cosinus (= result)
    Radius = ((Point.X - Referencepoint.X) ^ 2 + _
        (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
    Result = (Point.X - Referencepoint.X) / Radius

   'Returning function value
    Return Result
End Function
```

De beslis-logica voor een cirkel met zes gelijke segmenten kan er als volgt uitzien:

```
   'Variable on module level
    Dim ButtonMouse As Coordinate   'Location of mouse click on button
    Dim Center As Coordinate        'Location of center of segments

   'Calculating Cosinus and Sinus
    Cosinus = CreateCosinus(ButtonMouse, Center)
    Sinus = CreateSinus(ButtonMouse, Center)

   'Determinating in which segment was clicked (six segments)
    Select Case Cosinus
        Case >= 0
            If Sinus < -0.5 Then
                MsgBox("top right",,)
            ElseIf Sinus >= -0.5 And Sinus < 0.5 Then
                MsgBox("middle right",,)
            ElseIf Sinus >= 0.5 Then
                MsgBox("bottom right",,)
            End If
        Case < 0
            If Sinus < -0.5 Then
                MsgBox("top left",,)
            ElseIf Sinus >= -0.5 And Sinus < 0.5 Then
                MsgBox("middle left",,)
            ElseIf Sinus >= 0.5 Then
                MsgBox("bottom left",,)
            End If
    End Select
```

In het eerste voorbeeld is werkt met sinus en cosinus bij het determineren van de juiste zone.

Graden

Met behulp van een gegeven sinus en cosinus van een hoek kunnen we de hoek in graden bepalen. De VB.Net functies voor sinus en cosinus leveren een hoek in radialen. Door deze waarde te delen door 2 Pi en te vermenigvuldigen met 360 wordt deze hoek omgezet in graden (2Pi/360 = Pi/180).

```
Function CreateDegree(Point As Coordinate, Referencepoint As Coordinate) As Double
   'Internal function variable variables
    Dim Result As Double
    Dim Radius As Double
    Dim Sinus As Double
    Dim Cosinus As Double
    Dim TempArc As Double

   'Calculating Radius, Sinus and Cosinus
    Radius = ((Point.X - Referencepoint.X) ^ 2 + _
        (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
    Sinus = (Point.Y - Referencepoint.Y) / Radius
    Cosinus = (Point.X - Referencepoint.X) / Radius
```

```
    'Calculating TempArc in degrees
     TempArc = Math.Acos (Cosinus) * (180 / Math.PI)

    'Determinating the real arc (=Result) based on Sinus and Cosinus in degree
     If Cosinus >= 0 And Sinus < 0 Then
         Result = 90 - TempArc
     ElseIf Cosinus >= 0 And Sinus >= 0 Then
         Result = 90 + TempArc
     ElseIf Cosinus < 0 And Sinus >= 0 Then
         Result = 90 + TempArc
     ElseIf Cosinus < 0 And Sinus < 0 Then
         Result = 450 - TempArc
     End If

    'Returning function value
     Return Result

End Function
```

De beslis-logica voor een cirkel met zes gelijke segmenten zou er als volgt uit kunnen zien:

```
    'Variable on module level
     Dim ButtonMouse As Coordinate   'Location of mouse click on button
     Dim Center As Coordinate        'Location of center of segments

    'Calculating Cosinus and Sinus
     Degree = CreateDegree(ButtonMouse, Center)

    'Determinating in which segment was clicked (six segments)
     If Degree >= 0 And Degree < 60 Then
         MsgBox("top right",,)
     ElseIf Degree >= 60 And Degree < 120 Then
         MsgBox("middle right",,)
     ElseIf Degree >= 120 And Degree < 180 Then
         MsgBox("bottom right",,)
     ElseIf Degree >= 180 And Degree < 240 Then
         MsgBox("bottom left",,)
     ElseIf Degree >= 240 And Degree < 300 Then
         MsgBox("middle left",,)
     ElseIf Degree >= 300 And Degree < 360 Then
         MsgBox("top left",,)
     End If
```

In het derde voorbeeld is werkt met sinus en cosinus bij het determineren van de juiste zone.

Aandachtspunten bij het werken met sinus, cosinus en graden

Wees er bij het gebruiken van deze sinus en cosinus bedacht op, dat ze gebaseerd zijn op een cartesisch coördinaten systeem: X loopt op van links naar rechts, Y loopt op van **onder naar boven** en hoeken **tegen de klok in**. Het coördinaten systeem waar we mee werken op ons scherm wijkt af: X loopt wel op van links naar rechts, maar Y loopt op van **boven naar onder** en hoeken **met de klok mee**. Bij het opzetten van beslis-logica moeten we hier rekening mee houden.

Een tweede punt van aandacht is volgorde van de termen bij het bepalen van de verschillen. De eerste term is het specifieke punt en de tweede het referentiepunt. Op deze wijze komen de tekens van de het verschil in X en het verschil in Y overeen met de tekens van het coördinatensysteem. Het omdraaien van de volgorde heeft invloed op de beslis-logica.

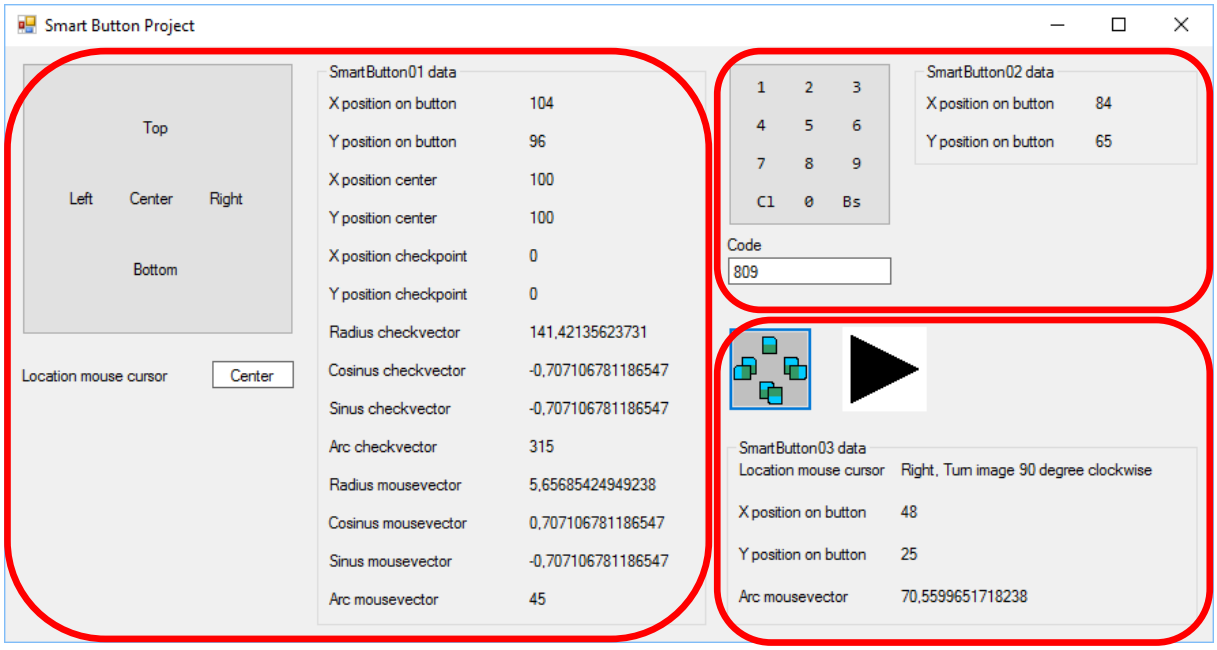
Poolcoördinaat

In de theorie is al aangegeven dat een punt ook kan worden beschreven met behulp van een poolcoördinaat, het beschrijven van een punt door het geven van de afstand en richting ten opzichte van een ander punt. In de functie `CreatePolarCoord` wordt een variabele gegenereerd met alle eigenschappen van een poolcoördinaat. eigenlijk is het een samenvoeging van de voorgaande vier functies (`CreateRadius, CreateSinus, CreateCosinus, CreateDegree`).

```
Function CreatePolarCoord(Point As Coordinate, Referencepoint As Coordinate) _
    As PolarCoordinate
   'Internal function variable variables
    Dim TempArc As Double
    Dim Result As PolarCoordinate

   'Calculating Radius, Cosinus, Sinus and Arc
    Result.Radius = ((Point.X - Referencepoint.X) ^ 2 + _
        (Point.Y - Centerpoint.Y) ^ 2) ^ 0.5
    Result.Cosinus = (Point.X - Referencepoint.X) / Result.Radius
    Result.Sinus = (Point.Y - Referencepoint.Y) / Result.Radius

   'Calculating TempArc in degrees
    TempArc = Math.Acos(Result.Cosinus) * (180 / Math.PI)

    'Determining the real arc (=Result.Arc) based on Sinus and Cosinus in degree
    If Result.Cosinus >= 0 And Result.Sinus < 0 Then
        Result.Arc = 90 - TempArc
    ElseIf Result.Cosinus >= 0 And Result.Sinus >= 0 Then
        Result.Arc = 90 + TempArc
    ElseIf Result.Cosinus < 0 And Result.Sinus >= 0 Then
        Result.Arc = 90 + TempArc
    ElseIf Result.Cosinus < 0 And Result.Sinus < 0 Then
        Result.Arc = 450 - TempArc
    End If

   'Returning function value
    Return Result

End Function
```

Voorbeelden

Bij dit document behoren een drietal voorbeelden. Deze zijn opgenomen in de voorbeeld applicatie “Smart Button Project” (SBP.exe). De meeste van de in dit document besproken functionaliteit komt in deze voorbeelden naar voren. `Smartbutton01` (links) is het eerste voorbeeld. Het werd gebruikt voor het ontwikkelen en beschrijven van de functionaliteit. `Smartbutton02` (rechtsboven) is een voorbeeld van een raster verdeling. `Smartbutton03` (rechtsonder) laat zien dat u op een knop van 62 x 62 pixels vier verschillende (samenhangende) functionaliteiten kunt concentreren.



Het project bestaat uit één formulier (SPBmain.vb (VB-code, bijlage 1). Deze VB.Net code bestaat uit 6 gedeelten, namelijk “Structuren voor variabelen en variabelen op module niveau”, “Functies”, “Activiteiten bij het laden van het formulier” en de functionaliteiten voor `SmartButton01` tot en met `SmartButton03`. U ziet in bovenstaande afbeelding dat allerlei gegevens worden weergegeven. Deze hebben betrekking op de variabelen die gebruikt worden binnen de verschillende subroutines.

Deel 1: Structuren voor variabelen en variabelen

In deel 1 zijn de structuren voor coördinaten, poolcoördinaten en de balk en kader van een formulier. Daarnaast zij er nog twee variabelen op module niveau opgenomen ten behoeve van `SmartButton02` en `SmartButton03`.

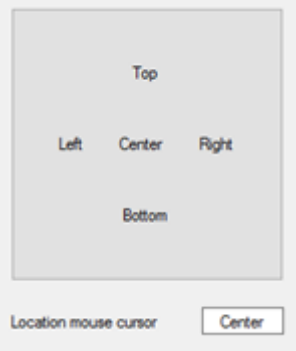
Deel 2: Functies

Hier vindt u alle in dit document beschreven functies.

Deel 3: Activiteiten bij het laden van het formulier

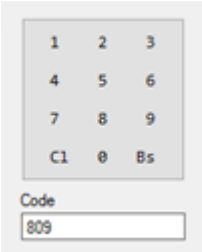
Bij het laden van het formulier wordt teksten voor `SmartButton01` en `SmartButton02` geladen. Tevens wordt een kopie gemaakt van de “pijl” afbeelding van `SmartButton03`.

Deel 4: Voorbeeld Smartbutton01



Dit eerste voorbeeld was de eerste vingeroefening van dit project. Het doel was een knop te realiseren met 5 zones, namelijk een circulaire zone in het midden met daar omheen 4 zones die gescheiden worden door de diagonalen. (zie theorie combinaties).De knop is voor testdoeleinden over geproportioneerd. De eerste stappen van de subroutine `SmartButton01_Click` zijn er opgericht de plaats van de muisklik op de knop vast te stellen. Vervolgens worden de sinus, de cosinus en de afstand naar de linker bovenhoek (`CheckPoint/CheckVector`) en de muisklik bepaald met behulp van de functie `CreatePolarCoord`. Deze beide hoeken worden vervolgens gebruikt voor het bepalen van de juiste zone door ze onderling te vergelijken. De feitelijke actie die aan de muisklik wordt gekoppeld is het tonen van de zone waar gelikt is.

Deel 5: Voorbeeld Smartbutton02



Het tweede voorbeeld is een “simulatie” van een telefoon toetsenbord. De knop is daar bij verdeelt in twaalf zones (een raster indeling met drie zones in de breedte, vier zones in de hoogte). Ook in dit voorbeeld zijn de eerste stappen er opgericht de plaats van de muisklik op de knop vast te stellen. Vervolgens worden de X en Y van de muisklik vergeleken met de grenzen tussen de zones. Daarna wordt het aangeklikte cijfer toegevoegd aan de codetekst onder de knop. De zones CL en Bs geven de mogelijkheid om respectievelijk alle tekens (CL = Clear) of het laatste teken (Bs = Backspace) te wissen

Deel 6: Voorbeeld Smartbutton03

Het derde voorbeeld betreft reëel prototype voor een knop die ik gebruik in een ander applicatie. De vierkante knop bestaat uit 4 zones, gescheiden door de diagonalen. Aan de zones zijn (samenhangende) functionaliteiten gekoppeld om een afbeelding te kunnen roteren (van boven rechtsom: Afbeelding herstellen, 90 graden naar met de klok mee roteren, 180 graden roteren en 90 graden tegen de klok in roteren). In dit prototype zult u de pijl rechts zien roteren als u op de knop klikt. Wederom zijn de eerste van deze subroutine gericht op de plaatsbepaling van de muisklik. Vervolgens wordt met de functie `CreatePolarCoord` de hoek bepaald ten opzichte van de verticale as. Deze hoek is dan bepalen in welke zonde geklikt is.
Ter informatie: de afbeeldingen Button.bmp en Arrow.bmp zijn in de “My Project” map geplaatst, vervolgens in het project geïmporteerd in “Project Resource file” en van daaruit gekoppeld aan de juiste afbeelding.

Ervaringen

- Een zone van 30 bij 30 levert een werkbaar situatie op, zowel voor het gebruik van een muis als voor het gebruik van een stick op een touchscreen.
- Test de functionaliteit van alle zones, zodra dit mogelijk is. Desnoods met de meldingen-functionaliteit van VB (`MsgBox`). Op deze manier kunt u zich overtuigen van de juistheid van de beslis-logica.
- Houdt, bij het bouwen van beslis-logica rekening met een afwijkend coördinatensysteem ten opzicht van het cartesisch stelsel (zie aandachtspunten)

Bijlage 1: Code van SBPmain.vb

```
Public Class SBPmain

    'Part 1: Variable tructures and variables on module level

    'Variable structure for handling pixel-coordinates
    Structure Coordinate
        Dim X As Short
        Dim Y As Short
    End Structure

    'Variable structure for handling pixel-polar coordinates
    Structure PolarCoordinate
        Dim Arc As Double
        Dim Radius As Double
        Dim Cosinus As Double
        Dim Sinus As Double
    End Structure

    'Variable structure for handling the bar and border size of a container
    Structure FrameSize
        Dim Border As Short
        Dim Bar As Short
    End Structure

    'Variable used bij SmartButton02
    Dim SmartText As String

    'Variable (object) used bij SmartButton03
    Dim BaseImage As Image

    'Part 2: Functions

    'Function calculates border- and barsize for rectangular objects
    Function ObjectFrame(OuterWidth As Integer, OuterHeight As Integer,
        InnerWidth As Integer, InnerHeight As Integer) As FrameSize
        'Internal function variable
        Dim Result As FrameSize

        'Calculating border and Bar size (=Result)
        Result.Border = (OuterWidth - InnerWidth) / 2
        Result.Bar = OuterHeight - InnerHeight - 2 * Result.Border

        'Returning function value
        Return Result

    End Function

    'Function calculates border- and barsize for forms
    Function Frame(Element As Form) As FrameSize
        'Internal function variable
        Dim Result As FrameSize

        'Calculating border and Bar size (=Result)
        Result.Border = (Element.Width - Element.ClientSize.Width) / 2
        Result.Bar = Element.Height - Element.ClientSize.Height - 2 * Result.Border

        'Returning function value
        Return Result

    End Function

    'Function calculates the distance (Radius) between two points
    Function CreateRadius(Point As Coordinate, Referencepoint As Coordinate) As Double
        'Internal function variable variables
        Dim Result As Double

        'Calculating Radius (= Result)
        Result = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5

        'Returning function value
```

```
        Return Result

    End Function

    'Function calculates the sinus between two points
    Function CreateSinus(Point As Coordinate, Referencepoint As Coordinate) As Double
        'Internal function variables
        Dim Result As Double
        Dim Radius As Double

        'Calculating Radius and Sinus (= Result)
        Radius = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
        Result = (Point.Y - Referencepoint.Y) / Radius

        'Returning function value
        Return Result

    End Function

    'Function calculates the cosinus between two points
    Function CreateCosinus(Point As Coordinate, Referencepoint As Coordinate) As Double
        'Internal function variables
        Dim Result As Double
        Dim Radius As Double

        'Calculating Radius and Cosinus (= Result)
        Radius = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
        Result = (Point.X - Referencepoint.X) / Radius

        'Returning function value
        Return Result

    End Function

    'Function calculates the arc in degrees between two points
    Function CreateDegree(Point As Coordinate, Referencepoint As Coordinate) As Double
        'Internal function variables
        Dim Result As Double
        Dim Radius As Double
        Dim Sinus As Double
        Dim Cosinus As Double
        Dim TempArc As Double

        'Calculating Radius, Sinus and Cosinus
        Radius = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
        Sinus = (Point.Y - Referencepoint.Y) / Radius
        Cosinus = (Point.X - Referencepoint.X) / Radius

        'Calculating TempArc in degree
        TempArc = Math.Acos(Cosinus) * (180 / Math.PI)

        'Determining the real arc (=Result) based on Sinus and Cosinus in degree
        If Cosinus >= 0 And Sinus < 0 Then
            Result = 90 - TempArc
        ElseIf Cosinus >= 0 And Sinus >= 0 Then
            Result = 90 + TempArc
        ElseIf Cosinus < 0 And Sinus >= 0 Then
            Result = 90 + TempArc
        ElseIf Cosinus < 0 And Sinus < 0 Then
            Result = 450 - TempArc
        End If

        'Returning function value
        Return Result

    End Function

    'Function calculates polar coordinate between two points
    Function CreatePolarCoord(Point As Coordinate, Referencepoint As Coordinate) As PolarCoordinate
        'Internal function variable variables
        Dim TempArc As Double
        Dim Result As PolarCoordinate

        'Calculating Radius, Cosinus, Sinus and Arc
```

```
        Result.Radius = ((Point.X - Referencepoint.X) ^ 2 + (Point.Y - Referencepoint.Y) ^ 2) ^ 0.5
        Result.Cosinus = (Point.X - Referencepoint.X) / Result.Radius
        Result.Sinus = (Point.Y - Referencepoint.Y) / Result.Radius

        'Calculating TempArc in degrees
        TempArc = Math.Acos(Result.Cosinus) * (180 / Math.PI)

        'Determining the real arc (=Result.Arc) based on Sinus and Cosinus in degree
        If Result.Cosinus >= 0 And Result.Sinus < 0 Then
            Result.Arc = 90 - TempArc
        ElseIf Result.Cosinus >= 0 And Result.Sinus >= 0 Then
            Result.Arc = 90 + TempArc
        ElseIf Result.Cosinus < 0 And Result.Sinus >= 0 Then
            Result.Arc = 90 + TempArc
        ElseIf Result.Cosinus < 0 And Result.Sinus < 0 Then
            Result.Arc = 450 - TempArc
        End If

        'Returning function value
        Return Result

    End Function


    'Part 3: Form loading activities

    Private Sub SBPmain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'On load the text or image for the buttons is created (Text is horizontal an vertical centered
)
        'Subroutine internal variables
        Dim ButtonText As String
        Dim EndOfLine As String = Chr(13) & Chr(10)

        'Creating the text for SmartButton01
        ButtonText = "Top" & EndOfLine & EndOfLine & EndOfLine & EndOfLine &
            "Left         Center         Right" &
             EndOfLine & EndOfLine & EndOfLine & EndOfLine & "Bottom"
        SmartButton01.Text = ButtonText

        'Creating the text for SmartButton02
        ButtonText = "1    2    3" & EndOfLine & EndOfLine &
            "4    5    6" & EndOfLine & EndOfLine &
            "7    8    9" & EndOfLine & EndOfLine &
            "Cl   0   Bs"
        SmartButton02.Text = ButtonText

        'Image used for SmartButton03
        'Saving a copy of the original picture
        BaseImage = Button03PictureBox.Image.Clone

    End Sub

    'Part 4: Handling clicks on SmartButton01
    Private Sub SmartButton01_Click(sender As Object, e As EventArgs) Handles SmartButton01.Click

        'Determinating positioon of mouse click

        'Mouse position on screen
        Dim ScreenMouse As Coordinate
        ScreenMouse.X = MousePosition.X
        ScreenMouse.Y = MousePosition.Y

        'Form topleftcorner on screen
        Dim FormTopLeftCorner As Coordinate
        FormTopLeftCorner.X = Me.Left
        FormTopLeftCorner.Y = Me.Top

        'Forms border- and barsize are calculated by using the function Frame
        Dim FormFrame As FrameSize
        FormFrame = Frame(Me)

        'Position topleftcorner of button based on clientsize form
        Dim ButtonTopLeftCorner As Coordinate
        ButtonTopLeftCorner.X = SmartButton01.Left
```

```
        ButtonTopLeftCorner.Y = SmartButton01.Top

        'Position mouse on SmartButton
        Dim ButtonMouse As Coordinate
        'Calculate relative position mouse on the SmartButton
        ButtonMouse.X = ScreenMouse.X - FormTopLeftCorner.X - FormFrame.Border - ButtonTopLeftCorner.X
        ButtonMouse.Y = ScreenMouse.Y - FormTopLeftCorner.Y - FormFrame.Border - FormFrame.Bar -
ButtonTopLeftCorner.Y

        'Data to labels on form
        XposMouse01Value.Text = ButtonMouse.X
        YposMouse01Value.Text = ButtonMouse.Y

        'Calculating center of SmartButton01
        Dim Center As Coordinate
        Center.X = SmartButton01.Width / 2
        Center.Y = SmartButton01.Height / 2

        'Data to labels on form
        XposCenterValue.Text = Center.X
        YposCenterValue.Text = Center.Y

        'To discreminate Top/Bottom/Left/Right of the mouseposition the mouse sinus and cosinus
        'will be compared with the sinus and cosinus of the topleftcormer of the button
(CheckPoint/CheckVector)

        'Setting CheckPoint data
        Dim CheckPoint As Coordinate
        Dim CheckVector As PolarCoordinate
        CheckPoint.X = 0
        CheckPoint.Y = 0
        'Calculating data for CheckVector
        CheckVector = CreatePolarCoord(CheckPoint, Center)

        'Data to labels on form
        XposCheckValue.Text = CheckPoint.X
        YposCheckValue.Text = CheckPoint.Y
        RadiusCheckVectorValue.Text = CheckVector.Radius
        CosinusCheckVectorValue.Text = CheckVector.Cosinus
        SinusCheckVectorValue.Text = CheckVector.Sinus
        ArcCheckVectorValue.Text = CheckVector.Arc

        'Calculate PolarCoordinate MouseVector
        Dim MouseVector As PolarCoordinate
        MouseVector = CreatePolarCoord(ButtonMouse, Center)

        'Data to the labels on the form
        RadiusMouseVector01Value.Text = MouseVector.Radius
        CosinusMouseVector01Value.Text = MouseVector.Cosinus
        SinusMouseVector01Value.Text = MouseVector.Sinus
        ArcMouseVector01Value.Text = MouseVector.Arc

        'Creating 5 zones and adding functionality to those zones

        'If Mouse position within a radius of 20% of the CheckVector > Center
        If MouseVector.Radius < CheckVector.Radius / 5 Then
            Button01ResultValue.Text = "Center"

            'Comparing sinus and cosinus values for determinating Top/Bottom/Left/Right

        ElseIf MouseVector.Sinus < CheckVector.Sinus Then
            Button01ResultValue.Text = "Top"

        ElseIf MouseVector.Sinus > -CheckVector.Sinus Then
            Button01ResultValue.Text = "Bottom"

        ElseIf MouseVector.Cosinus < -CheckVector.Cosinus Then
            Button01ResultValue.Text = "Left"

        ElseIf MouseVector.Cosinus > CheckVector.Cosinus Then
            Button01ResultValue.Text = "Right"

        Else
            'For the most increadible cases (if cases above don't fit)
```

```
            Button01ResultValue.Text = "Nothing"
        End If

    End Sub

    'Part 5: Handling clicks on SmartButton02
    Private Sub SmartButton02_Click(sender As Object, e As EventArgs) Handles SmartButton02.Click

        'Determinating positioon of mouse click

        'Mouse position on screen
        Dim ScreenMouse As Coordinate
        ScreenMouse.X = MousePosition.X
        ScreenMouse.Y = MousePosition.Y

        'Form topleftcorner on screen
        Dim FormTopLeftCorner As Coordinate
        FormTopLeftCorner.X = Me.Left
        FormTopLeftCorner.Y = Me.Top

        'Forms border- and barsize are calculated by using the function Frame
        Dim FormFrame As FrameSize
        FormFrame = Frame(Me)

        'Position topleftcorner of button based on clientsize form
        Dim ButtonTopLeftCorner As Coordinate
        ButtonTopLeftCorner.X = SmartButton02.Left
        ButtonTopLeftCorner.Y = SmartButton02.Top

        'Position mouse on SmartButton
        Dim ButtonMouse As Coordinate
        'Calculate relative position mouse on the SmartButton
        ButtonMouse.X = ScreenMouse.X - FormTopLeftCorner.X - FormFrame.Border - ButtonTopLeftCorner.X
        ButtonMouse.Y = ScreenMouse.Y - FormTopLeftCorner.Y - FormFrame.Border - FormFrame.Bar -
ButtonTopLeftCorner.Y

        'Data to labels on form
        XposMouse02Value.Text = ButtonMouse.X
        YposMouse02Value.Text = ButtonMouse.Y

        'Adding functionality to zones on SmartButton02 (adding characters to SmartText)

        'Vertical borders at (X=) 40 and 80
        'Horizontal borders a (Y=) 30, 60 and 90
        Select Case ButtonMouse.X
            Case 0 To 39
                Select Case ButtonMouse.Y
                    Case 0 To 29
                        SmartText = SmartText + "1"
                    Case 30 To 59
                        SmartText = SmartText + "4"
                    Case 60 To 89
                        SmartText = SmartText + "7"
                    Case 90 To 119
                        'Deleting all text
                        SmartText = ""
                End Select
            Case 40 To 79
                Select Case ButtonMouse.Y
                    Case 0 To 29
                        SmartText = SmartText + "2"
                    Case 30 To 59
                        SmartText = SmartText + "5"
                    Case 60 To 89
                        SmartText = SmartText + "8"
                    Case 90 To 119
                        SmartText = SmartText + "0"
                End Select
            Case 80 To 119
                Select Case ButtonMouse.Y
                    Case 0 To 29
                        SmartText = SmartText + "3"
                    Case 30 To 59
                        SmartText = SmartText + "6"
```

```
                    Case 60 To 89
                        SmartText = SmartText + "9"
                    Case 90 To 119
                        'Deleting last characters (only if Smart <> "")
                        If SmartText <> "" Then
                            SmartText = Mid(SmartText, 1, Len(SmartText) - 1)
                        End If
                End Select
        End Select

        'Smarttect to label on form
        Button02ResultValue.Text = SmartText
    End Sub

    'Part 6: Handling clicks on SmartButton03

    Private Sub SmartButton03_Click(sender As Object, e As EventArgs) Handles SmartButton03.Click
        'Subroutine internal variable
        Dim TempImage As Bitmap

        'Determinating positioon of mouse click

        'Mouse position on screen
        Dim ScreenMouse As Coordinate
        ScreenMouse.X = MousePosition.X
        ScreenMouse.Y = MousePosition.Y

        'Form topleftcorner on screen
        Dim FormTopLeftCorner As Coordinate
        FormTopLeftCorner.X = Me.Left
        FormTopLeftCorner.Y = Me.Top

        'Forms border- and barsize are calculated by using the function Frame
        Dim FormFrame As FrameSize
        FormFrame = Frame(Me)

        'Position topleftcorner of button based on clientsize form
        Dim ButtonTopLeftCorner As Coordinate
        ButtonTopLeftCorner.X = SmartButton03.Left
        ButtonTopLeftCorner.Y = SmartButton03.Top

        'Position mouse on SmartButton
        Dim ButtonMouse As Coordinate
        'Calculate relative position mouse on the SmartButton
        ButtonMouse.X = ScreenMouse.X - FormTopLeftCorner.X - FormFrame.Border - ButtonTopLeftCorner.X
        ButtonMouse.Y = ScreenMouse.Y - FormTopLeftCorner.Y - FormFrame.Border - FormFrame.Bar -
ButtonTopLeftCorner.Y

        'Data to labels on form
        XposMouse03Value.Text = ButtonMouse.X
        YposMouse03Value.Text = ButtonMouse.Y

        'Calculating center of SmartButton03
        Dim Center As Coordinate
        Center.X = SmartButton03.Width / 2
        Center.Y = SmartButton03.Height / 2

        'To discreminate Top/Bottom/Left/Right of the mouseposition the mouse Arc will be used

        'Calculate PolarCoordinate MouseVector
        Dim MouseVector As PolarCoordinate
        MouseVector = CreatePolarCoord(ButtonMouse, Center)

        'Data to the label on the form
        ArcMouseVector03Value.Text = MouseVector.Arc

        'Discriminating 4 zones of 90 degree
        If MouseVector.Arc >= 315 Or MouseVector.Arc < 45 Then          'Top zone
            'Data to label on form
            Button03ResultValue.Text = "Top, Reset image"
            'Loading the original picture
            Button03PictureBox.Image = BaseImage.Clone

        ElseIf MouseVector.Arc >= 45 And MouseVector.Arc < 135 Then     'Right zone
```

```
            'Data to label on form
            Button03ResultValue.Text = "Right, Turn image 90 degree clockwise"
            'Turning image 90 degree
            TempImage = CType(Button03PictureBox.Image, Bitmap)
            TempImage.RotateFlip(RotateFlipType.Rotate90FlipNone)
            Button03PictureBox.Image = TempImage

        ElseIf MouseVector.Arc >= 135 And MouseVector.Arc < 225 Then    'Bottom zone
            'Data to label on form
            Button03ResultValue.Text = "Bottom, Turn image 180 degree"
            'Turning image 180 degree
            TempImage = CType(Button03PictureBox.Image, Bitmap)
            TempImage.RotateFlip(RotateFlipType.Rotate180FlipNone)
            Button03PictureBox.Image = TempImage

        ElseIf MouseVector.Arc >= 225 And MouseVector.Arc < 315 Then    'Left zone
            'Data to label on form
            Button03ResultValue.Text = "Left, Turn image 90 degree anti-clockwise"
            'Turning image 270 degree (= -90 degree)
            TempImage = CType(Button03PictureBox.Image, Bitmap)
            TempImage.RotateFlip(RotateFlipType.Rotate270FlipNone)
            Button03PictureBox.Image = TempImage

        End If

    End Sub

End Class
```