Corsi di Programmazione di Interfacce, Interazione uomo macchina

Nome: Eleonora Di Gregorio

Matricola: 520655

Piattaforma utilizzata: il codice è stato scritto in F# 4.0 utilizzando Visual Studio Code 1.17.2 su Windows 10 Pro.

## Esercizio 1

Per dare la percezione a chi estende la classe *LWControl* di avere un proprio spazio di coordinate è necessario modificare il member *OnPaint* della classe *LWContainer*. Prima di chiamare *l’OnPaint* di ciascun controllo l’origine del mondo del *LWContainer* deve coincidere con la Position del *LWControl*, la variabile position infatti rappresenta il punto in cui viene fissata l’origine di questo sistema di coordinate ed è espressa in coordinate vista o mondo del *LWContainer* a seconda del valore che assume la variabile *coordinates*. Per tutelarsi da eventuali modifiche, il *LWContainer* può limitare l’area di disegno del *LWControl* alla regione individuata dall’origine dello spazio di coordinate e dalla *size* del *LWControl*. Di seguito sono riportati:

* Il codice della nuova *OnPaint*

override this.OnPaint e =

let g = e.Graphics

let t = g.Transform

//viene effettuata una traslazione in modo tale che l'origine del contesto grafico corrente sia fissata a position

//viene delimitata la regione di disegno del LWControl

//viene creato un nuovo evento paint che interessa solo questa regione.

let TranslateClip (c:LWControl) =

let s = e.Graphics.Save()

e.Graphics.TranslateTransform(c.Position.X, c.Position.Y)

e.Graphics.Clip <- new Region(RectangleF(0.f, 0.f, c.Size.Width+1.f, c.Size.Height+1.f))

let r = e.Graphics.ClipBounds

let evt = new PaintEventArgs(e.Graphics, new Rectangle(int(r.Left), int(r.Top), int(r.Width), int(r.Height)))

c.OnPaint evt

e.Graphics.Restore(s)

g.Transform <- transform.W2V

for idx in (controls.Count - 1) .. -1 .. 0 do

let c = controls.[idx]

if c.CoordinateType = World then

TranslateClip(c)

g.Transform <- t

for idx in (controls.Count - 1) .. -1 .. 0 do

let c = controls.[idx]

if c.CoordinateType = View then

TranslateClip(c)

* Esempio di modifica dell’*OnMouseDown*

override this.OnMouseDown e =

let p = PointF(single e.X, single e.Y)

let controlsView = controls |> Seq.filter (fun c -> c.CoordinateType = View)

match (controlsView |> Seq.tryFind (fun c -> c.HitTest (PointF(p.X - c.Position.X,p.Y - c.Position.Y)) )) with

| Some c -> c.OnMouseDown(e)

| None ->

let pw = transformPoint transform.V2W p

let controlsWorld = controls |> Seq.filter (fun c -> c.CoordinateType = World)

match (controlsWorld |> Seq.tryFind(fun c -> c.HitTest (PointF(pw.X - c.Position.X,pw.Y - c.Position.Y)))) with

| Some c -> c.OnMouseDown(e)

| None -> ()

* Esempio estensione *LWContainer*

type PIButton() =

inherit LWControl()

let mutable text = ""

member this.Text

with get() = text

and set(v) = text <- v

override this.OnPaint e =

let parent = this.Parent

let g = e.Graphics

let r = RectangleF(PointF(0.f,0.f), this.Size) |> RectF2Rect

g.DrawRectangle(Pens.Red, r)

let ssz = g.MeasureString(text, parent.Font)

let sz = this.Size

let sx, sy = (sz.Width - ssz.Width) / 2.f, (sz.Height - ssz.Height) / 2.f

g.DrawString(text, parent.Font, Brushes.Red, PointF(sx, sy))

Nella cartella Esercizio1 si trova il codice della libreria estesa nel file LWC1.fsx. Nel file LWC1test.fsx è presente una modifica del test realizzato a lezione per la libreria.

## Esercizio 2

La forma generalizzata di una curva di Bezier di grado n è la seguente:

Di conseguenza quella di una curva di Bezier cubica è:

All’aumentare delle dimensioni della curva aumenta il costo per calcolarla ed inoltre diventa sempre più difficile controllarle a causa dei troppi punti di controllo. È per questo che si preferisce utilizzare delle curve di Bezier cubiche, inoltre queste ultime hanno proprietà rilevanti di continuità che, oltre a renderle esteticamente piacevoli, favoriscono la loro combinazione. Sequenze di curve di Bezier, chiamate a volte path, possono creare curve di grande complessità. Per combinare due curve di Bezier cubiche B1 e B2, la prima avente punti di controllo principali e secondari (analogo per la seconda), si fa coincidere il punto di controllo con e se si vuole che la curva sia liscia si deve imporre che i due punti siano colineari, ovvero valga . Un esempio è mostrato in figura:



Si possono modificare però i punti intermedi per creare diversi effetti, ad esempio in modo da creare delle cuspidi.

Applicando iterativamente questo procedimento si possono combinare un numero arbitrario di curve.