TPI 2020 - LogoGo : Code Source

Walter JAUCH
I.DA-P4A
École d'informatique (CFPT-I)

9 Juin 2020

Table des matières

0.0.1 Sprite.cs

```
1   /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de ↩
                     calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : Sprite.cs
16
17 */
18 using System.Drawing;
19 using System.Windows.Forms;
20
21 namespace WF_LogoGo
22 {
23     public abstract class Sprite : PictureBox
24     {
25         #region Variables d'instance
26         private Color _couleur;
27         private float _epaisseurPen;
28         private bool _remplir;
29         private int _profondeur;
30         private int _numeroCalque;
31         private int _idType;
32         private PointF[] _trace;
33         private string _texte;
34         private string _nomPolice;
35         private int _taillePolice;
36
37         private Form _parent;
38
39
40
41         private bool _dragging;
42         private Point _posSouris;
43         #endregion
44
45         #region Constantes
46         const int HAUTEUR_INIT = 100;
47         const int LARGEUR_INIT = 100;
48         #endregion
49
50         #region Propriétés
51         /// <summary>
52         /// Couleur du pinceau / crayon du sprite
53         /// </summary>
54         public Color Couleur { get => _couleur; set => _couleur = ↩
               value; }
55
56         /// <summary>
57         /// Épaisseur du crayon
58         /// </summary>
59         public float EpaisseurPen { get => _epaisseurPen; set => ↩
               _epaisseurPen = value; }
60
61         /// <summary>
62         /// Bool permettant de savoir s'il faut remplir ou non un sprite
63         /// </summary>
64         public bool Remplir { get => _remplir; set => _remplir = value; }
65
66         /// <summary>
```

```
67          /// Profondeur qui définit l'ordre d'apparition des sprites dans un même calque
68          /// </summary>
69          public int Profondeur { get => _profondeur; set => _profondeur = value; }
70
71          /// <summary>
72          /// Calque sur lequel est se trouve le sprite
73          /// </summary>
74          public int NumeroCalque { get => _numeroCalque; set => _numeroCalque = value; }
75
76          /// <summary>
77          /// Identdifiant permettant de savoir quel type de sprite nous avons
78          /// </summary>
79          public int IdType { get => _idType; set => _idType = value; }
80
81          /// <summary>
82          /// Retourne la
83          /// </summary>
84          public int AlphaCouleur { get => Couleur.A; }
85
86          /// <summary>
87          /// Certains sprites ont besoin d'un tracé pour se dessiner
88          /// </summary>
89          public PointF[] Trace { get => _trace; set => _trace = value; }
90
91          /// <summary>
92          /// Certains sprites ont besoin d'un texte
93          /// </summary>
94          public string TexteAEcrire { get => _texte; set => _texte = value; }
95
96          /// <summary>
97          /// Certains sprites ont besoin d'un nom de police
98          /// </summary>
99          public string NomPolice { get => _nomPolice; set => _nomPolice = value; }
100
101         /// <summary>
102         /// Certains sprites ont besoin d'une taille de police
103         /// </summary>
104         public int TaillePolice { get => _taillePolice; set => _taillePolice = value; }
105
106
107         #endregion
108
109         #region Constructeurs
110         public Sprite(Form parent, int calque)
111         {
112             Couleur = Color.Black;
113             EpaisseurPen = 1;
114             Profondeur = 1;
115             NumeroCalque = calque;
116             Remplir = false;
117             _parent = parent;
118
119
120             Location = new Point(100, 100);
121             Size = new Size(LARGEUR_INIT, HAUTEUR_INIT);
122             BackColor = Color.Transparent;
123
124             base.MouseDown += SpriteMouseDown;
125             base.MouseUp += SpriteMouseUp;
126             base.MouseMove += SpriteMouseMove;
127         }
128
129         public Sprite(SpriteSerializable s, Form parent)
130         {
131             Couleur = Color.FromArgb(s.Couleur);
```

```
132             NumeroCalque = s.NumeroCalque;
133             EpaisseurPen = s.EpaisseurPen;
134             Profondeur = s.ProfondeurParCalque;
135             Remplir = s.Remplir;
136             if (this is Polygone)
137             {
138                 Trace = s.Trace;
139             }
140             if (this is Texte)
141             {
142                 TexteAEcrire = s.TexteAEcrire;
143                 TaillePolice = s.TaillePolice;
144                 NomPolice = s.NomPolice;
145             }
146             _parent = parent;
147 
148             Name = s.Nom;
149             Location = s.Location;
150             Size = s.Size;
151             BackColor = Color.Transparent;
152 
153             MouseDown += new MouseEventHandler(SpriteMouseDown);
154             MouseUp += new MouseEventHandler(SpriteMouseUp);
155             base.MouseMove += new MouseEventHandler(SpriteMouseMove);
156         }
157 
158         #endregion
159 
160         #region Méthodes
161 
162         /// <summary>
163         /// Applique une transparence à la couleur
164         /// </summary>
165         /// <param name="transparence"></param>
166         public void ChangerTransparence(int transparence)
167         {
168             Couleur = Color.FromArgb(transparence, Couleur);
169         }
170 
171         /// <summary>
172         /// Méthode qui dessine le sprite. La méthode est différente selon
173         /// le type du sprite (rond, carré, ...)
174         /// </summary>
175         /// <param name="sender"></param>
176         /// <param name="e"></param>
177         public void SpritePaint(object sender, PaintEventArgs e)
178         {
179             SpritePaintAvecGraphics(e.Graphics);
180         }
181 
182         /// <summary>
183         /// Dessine une forme (différent pour chaque classe fille).
184         /// </summary>
185         /// <param name="g">Objet Graphics avec lequel la forme sera ↩
                dessinée</param>
186         public abstract void SpritePaintAvecGraphics(Graphics g);
187 
188 
189         /// <summary>
190         /// Déplace la picturebox du sprite en fonction de la souris
191         /// </summary>
192         /// <param name="sender"></param>
193         /// <param name="e"></param>
194         private void SpriteMouseMove(object sender, MouseEventArgs e)
195         {
196             var c = sender as PictureBox;
197             if (!_dragging || null == c) return;
198             c.Top = e.Y + c.Top - _posSouris.Y;
199             c.Left = e.X + c.Left - _posSouris.X;
200             Parent.Invalidate();
201         }
202 
```

```
203         /// <summary>
204         /// Cesse le déplacement de la picturebox
205         /// </summary>
206         /// <param name="sender"></param>
207         /// <param name="e"></param>
208         private void SpriteMouseUp(object sender, MouseEventArgs e)
209         {
210             var pbx = sender as PictureBox;
211             if (null == pbx) return;
212             pbx.BackColor = Color.Transparent;
213             _dragging = false;
214         }
215 
216         /// <summary>
217         /// Commence le déplacement de la souris
218         /// </summary>
219         /// <param name="sender"></param>
220         /// <param name="e"></param>
221         private void SpriteMouseDown(object sender, MouseEventArgs e)
222         {
223             if (e.Button != MouseButtons.Left) return;
224             (sender as PictureBox).BackColor = Color.FromArgb(100, ↩
                    Color.Silver);
225             _dragging = true;
226             _posSouris = new Point(e.X, e.Y);
227         }
228 
229         /// <summary>
230         /// Retourne un sprite serializable
231         /// </summary>
232         /// <returns></returns>
233         public SpriteSerializable EnSpriteSerializable()
234         {
235             SpriteSerializable s = new SpriteSerializable();
236             s.AttribuerValeursProprietes(this);
237             return s;
238         }
239 
240         /// <summary>
241         /// Surcharge de la méthode ToString() de object
242         /// </summary>
243         /// <returns>Nom du sprite (" - Rond_1")</returns>
244         public override string ToString()
245         {
246             return "␣-␣" + Name;
247         }
248         #endregion
249     }
250 }
```

Listing 1 – ./files/Sprite.cs

0.0.2 Carre.cs

```
1   /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : Carre.cs
16
17 */
18 using System;
19 using System.Collections.Generic;
20 using System.Drawing;
21 using System.Linq;
22 using System.Text;
23 using System.Threading.Tasks;
24 using System.Windows.Forms;
25
26 namespace WF_LogoGo
27 {
28     class Carre : Sprite
29     {
30         #region Variables d'instance
31         private static int _nombreCarres = 0;
32         #endregion
33
34         #region Constructeurs
35         public Carre(Form parent, int calque) : base(parent, calque)
36         {
37             _nombreCarres++;
38             Name = "Carré_" + _nombreCarres.ToString();
39             IdType = 1;
40         }
41
42         public Carre(SpriteSerializable s, Form parent) : base(s, parent)
43         {
44             IdType = 1;
45         }
46         #endregion
47
48         #region Méthodes
49
50         /// <summary>
51         /// Dessine un carré selon la propriété Remplir.
52         /// </summary>
53         /// <param name="g">Objet Graphics avec lequel la forme sera dessinée</param>
54         public override void SpritePaintAvecGraphics(Graphics g)
55         {
56
57             g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
58             if (Remplir)
59             {
60                 SolidBrush b = new SolidBrush(Couleur);
61                 g.FillRectangle(b, new Rectangle(Location.X, Location.Y, Size.Width, Size.Height));
62             }
63             else
64             {
65                 Pen p = new Pen(Couleur, EpaisseurPen);
```

```
66                  g.DrawRectangle(p, new Rectangle(Location.X, ↩
                        Location.Y, Size.Width, Size.Height));
67              }
68
69          }
70
71          #endregion
72      }
73  }
```

Listing 2 – ./files/Carre.cs

0.0.3 Rond.cs

```
1   /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : Rond.cs
16
17 */
18 using System.Drawing;
19 using System.Windows.Forms;
20
21 namespace WF_LogoGo
22 {
23     class Rond : Sprite
24     {
25         #region Variables d'instance
26         private static int _nombreCarres = 0;
27         #endregion
28
29         #region Constructeurs
30         public Rond(Form parent, int calque) : base(parent, calque)
31         {
32             _nombreCarres++;
33             Name = "Rond_" + _nombreCarres.ToString();
34             IdType = 2;
35         }
36         public Rond(SpriteSerializable s, Form parent) : base(s, parent)
37         {
38             IdType = 2;
39         }
40         #endregion
41
42         #region Méthodes
43
44         /// <summary>
45         /// Dessine un Rond selon la propriété Remplir.
46         /// </summary>
47         /// <param name="g">Objet Graphics avec lequel la forme sera dessinée</param>
48         public override void SpritePaintAvecGraphics(Graphics g)
49         {
50             g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
51             if (Remplir)
52             {
53                 SolidBrush b = new SolidBrush(Couleur);
54                 g.FillEllipse(b, new Rectangle(Location.X, Location.Y, Size.Width + 1, Size.Height + 1));
55             }
56             else
57             {
58                 Pen p = new Pen(Couleur, EpaisseurPen);
59                 g.DrawEllipse(p, new Rectangle(Location.X, Location.Y, Size.Width + 1, Size.Height + 1));
60             }
61         }
62
63         #endregion
64     }
```

```
65 }
```

Listing 3 – `./files/Rond.cs`

0.0.4 Triangle.cs

```
1   /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : Triangle.cs
16
17 */
18 using System.Drawing;
19 using System.Windows.Forms;
20
21 namespace WF_LogoGo
22 {
23     class Triangle : Sprite
24     {
25         #region Variables d'instance
26         private static int _nombreTriangles = 0;
27         #endregion
28
29         #region Constructeurs
30         public Triangle(Form parent, int calque) : base(parent, calque)
31         {
32             _nombreTriangles++;
33             Name = "Triangle_" + _nombreTriangles.ToString();
34             IdType = 3;
35         }
36
37         public Triangle(SpriteSerializable s, Form parent) : base(s, parent)
38         {
39             IdType = 3;
40         }
41         #endregion
42
43         #region Méthodes
44
45         /// <summary>
46         /// Dessine un Troangle selon la propriété Remplir.
47         /// </summary>
48         /// <param name="g">Objet Graphics avec lequel la forme sera dessinée</param>
49         public override void SpritePaintAvecGraphics(Graphics g)
50         {
51             g.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
52             Point[] Trace =
53             {
54                 new Point(Location.X, Location.Y + Height),
55                 new Point(Location.X + Width / 2, Location.Y),
56                 new Point(Location.X + Width, Location.Y + Height),
57                 new Point(Location.X, Location.Y + Height)
58             };
59             if (Remplir)
60             {
61                 SolidBrush b = new SolidBrush(Couleur);
62                 g.FillPolygon(b, Trace);
63             }
64             else
65             {
```

```
66                  Pen p = new Pen(Couleur, EpaisseurPen);
67                  g.DrawPolygon(p, Trace);
68              }
69          }
70          #endregion
71      }
72  }
```

Listing 4 – ./files/Triangle.cs

0.0.5 Polygone.cs

```
1    /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de ↩
                     calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : Polygone.cs
16
17 */
18 using System.Drawing;
19 using System.Windows.Forms;
20
21 namespace WF_LogoGo
22 {
23     public class Polygone : Sprite
24     {
25         #region Variables d'instance
26         private static int _nombrePolygones = 0;
27
28         /// <summary>
29         /// Coin de la pictureBox se situant en haut à gauche.
30         /// </summary>
31         private PointF CoinSuperieurGauche;
32
33         /// <summary>
34         /// Coin de la pictureBox se situant en haut à droite.
35         /// </summary>
36         private PointF CoinSuperieurDroit;
37
38         /// <summary>
39         /// Coin de la pictureBox se situant en bas à droite.
40         /// </summary>
41         private PointF CoinInferieurDroit;
42
43         /// <summary>
44         /// Coin de la pictureBox se situant en bas à gauche.
45         /// </summary>
46         private PointF CoinInferieurGauche;
47         #endregion
48
49         #region Constructeurs
50         public Polygone(Form parent, int calque) : base(parent, calque)
51         {
52             _nombrePolygones++;
53             Name = "Polygone_" + _nombrePolygones.ToString();
54             Width = 50;
55             Height = 50;
56             IdType = 5;
57         }
58
59         public Polygone(SpriteSerializable s, Form parent) : base(s, ↩
               parent)
60         {
61             IdType = 5;
62         }
63         #endregion
64
65         #region Méthodes
66
67         /// <summary>
```

```
68          /// Dessine un polygone selon la propriété Remplir.
69          /// </summary>
70          /// <param name="g">Objet Graphics avec lequel la forme sera ↩
                dessinée</param>
71          public override void SpritePaintAvecGraphics(Graphics g)
72          {
73              PointF[] nouveauTrace = new PointF[Trace.Length];
74              for (int i = 0; i < Trace.Length; i++)
75              {
76                  nouveauTrace[i] = new PointF(Location.X + Trace[i].X, ↩
                        Location.Y + Trace[i].Y);
77              }
78
79              RedimensionnerPictureBox(nouveauTrace);
80
81              g.SmoothingMode = ↩
                    System.Drawing.Drawing2D.SmoothingMode.HighQuality;
82              if (Remplir)
83              {
84                  SolidBrush b = new SolidBrush(Couleur);
85                  g.FillPolygon(b, nouveauTrace);
86              }
87              else
88              {
89                  Pen p = new Pen(Couleur, EpaisseurPen);
90                  g.DrawPolygon(p, nouveauTrace);
91              }
92          }
93
94
95          /// <summary>
96          /// Définit une nouvelle taille pour le polygone en fonction de ↩
                la position des différents points
97          /// </summary>
98          /// <param name="pointsPolygone">Tracé à utiliser pour calculer ↩
                la taille</param>
99          private void RedimensionnerPictureBox(PointF[] pointsPolygone)
100         {
101             CalculerCoins(pointsPolygone);
102             Size = new Size((int)(CoinSuperieurDroit.X - ↩
                    CoinSuperieurGauche.X), (int)(CoinInferieurDroit.Y - ↩
                    CoinSuperieurDroit.Y));
103         }
104
105         /// <summary>
106         /// Calcule les coins de la picturebox en utilisant les points ↩
                du tracé du polygone
107         /// </summary>
108         /// <param name="pointsPolygone">Tracé à utiliser pour calculer ↩
                les coins</param>
109         private void CalculerCoins(PointF[] pointsPolygone)
110         {
111             bool premierTour = true;
112             foreach (PointF unPoint in pointsPolygone)
113             {
114                 if (premierTour)
115                 {
116                     CoinSuperieurGauche = Location;
117                     CoinSuperieurDroit = new PointF(Location.X + Width, ↩
                            Location.Y);
118                     CoinInferieurDroit = new PointF(Location.X + Width, ↩
                            Location.Y + Height);
119                     CoinInferieurGauche = new PointF(Location.X, ↩
                            Location.Y + Height);
120
121                     premierTour = false;
122                 }
123
124                 //Vérification coin supérieur droit
125                 if (unPoint.X > CoinSuperieurDroit.X)
126                 {
```

```
127                     CoinSuperieurDroit = new PointF(unPoint.X, ↩
                            CoinSuperieurDroit.Y);
128                 }
129
130                 //vérification coin inférieur droit
131                 if (unPoint.X > CoinInferieurDroit.X)
132                 {
133                     CoinInferieurDroit = new PointF(unPoint.X, ↩
                            CoinInferieurDroit.Y);
134                 }
135                 if (unPoint.Y > CoinInferieurDroit.Y)
136                 {
137                     CoinInferieurDroit = new ↩
                            PointF(CoinInferieurDroit.X, unPoint.Y);
138                 }
139
140                 //vérification coin inférieur gauche
141                 if (unPoint.Y > CoinInferieurGauche.Y)
142                 {
143                     CoinInferieurGauche = new ↩
                            PointF(CoinInferieurGauche.X, unPoint.Y);
144                 }
145                 if (unPoint.Y > CoinInferieurDroit.Y)
146                 {
147                     CoinInferieurDroit = new ↩
                            PointF(CoinInferieurDroit.X, unPoint.Y);
148                 }
149             }
150         }
151
152         #endregion
153     }
154 }
```

Listing 5 – `./files/Polygone.cs`

0.0.6 Texte.cs

```
1    /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : Texte.cs
16
17 */
18 using System.Drawing;
19 using System.Windows.Forms;
20
21 namespace WF_LogoGo
22 {
23     class Texte : Sprite
24     {
25         #region Variables d'instance
26         private static int _nombreCarres = 0;
27         #endregion
28
29
30         #region Constructeurs
31         public Texte(Form parent, int calque) : base(parent, calque)
32         {
33             _nombreCarres++;
34             Name = "Texte " + _nombreCarres.ToString();
35             TexteAEcrire = Name;
36             NomPolice = "Times New Roman";
37             TaillePolice = 24;
38             IdType = 4;
39         }
40
41         public Texte(SpriteSerializable s, Form parent) : base(s, parent)
42         {
43             IdType = 4;
44         }
45         #endregion
46
47         #region Méthodes
48
49         /// <summary>
50         /// Dessine le texte.
51         /// </summary>
52         /// <param name="g">Objet Graphics sur lequel le texte est dessiné</param>
53         public override void SpritePaintAvecGraphics(Graphics g)
54         {
55             SolidBrush b = new SolidBrush(Couleur);
56             Font maPolice = new Font(NomPolice, TaillePolice, FontStyle.Bold, GraphicsUnit.Pixel);
57             RedimensionnerPictureBox(maPolice, g);
58             g.DrawString(TexteAEcrire, maPolice, b, Location);
59         }
60
61         /// <summary>
62         /// Redimensionne la pictureBox en fonction de la police et du texte à écrire.
63         /// </summary>
64         /// <param name="uneFonte">Police utilisée</param>
65         /// <param name="g">Objet Graphics utilisé</param>
```

```
66          private void RedimensionnerPictureBox(Font uneFonte, Graphics g)
67          {
68              Width = (int)g.MeasureString(TexteAEcrire, uneFonte).Width;
69              Height = (int)g.MeasureString(TexteAEcrire, uneFonte).Height;
70          }
71
72          /// <summary>
73          /// Change le texte à afficher par le texte spécifié.
74          /// </summary>
75          /// <param name="nouveauTexte">nouvelle chaîne de caratère à ↩
                afficher</param>
76          public void ChangerTexte(string nouveauTexte)
77          {
78              TexteAEcrire = nouveauTexte;
79          }
80
81          /// <summary>
82          /// Surcharge du ToString de Sprite.
83          /// </summary>
84          /// <returns>Retourne le texte</returns>
85          public override string ToString()
86          {
87              return TexteAEcrire;
88          }
89          #endregion
90      }
91  }
```

Listing 6 – ./files/Texte.cs

0.0.7 Sprites.cs

```
1    /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : Sprites.cs
16
17 */
18 using System.Collections.Generic;
19 using System.Linq;
20
21 namespace WF_LogoGo
22 {
23     public class Sprites
24     {
25         #region Variables d'instance
26         private List<Sprite> _listeDeSprite;
27         #endregion
28
29         #region Propriétés
30
31         /// <summary>
32         /// Liste dans laquelle sont stockés tous les sprites existants
33         /// </summary>
34         public List<Sprite> ListeDeSprite { get => _listeDeSprite; private set => _listeDeSprite = value; }
35         /// <summary>
36         /// Permet de savoir si la liste de sprites est vide.
37         /// </summary>
38         public bool EstVide { get => ListeDeSprite.Count == 0; }
39         #endregion
40
41         #region Constructeurs
42         public Sprites()
43         {
44             ListeDeSprite = new List<Sprite>();
45         }
46         #endregion
47
48         #region Méthodes
49
50         /// <summary>
51         /// Ajoute à la liste le sprite spécifié
52         /// /// </summary>
53         /// <param name="unSprite">le Sprite à ajouter</param>
54         public void Ajouter(Sprite unSprite)
55         {
56             ListeDeSprite.Add(unSprite);
57         }
58
59         /// <summary>
60         /// Supprime de la liste le sprite spécifié
61         /// </summary>
62         /// <param name="unSprite">le Sprite à supprimer</param>
63         public void Supprimer(Sprite unSprite)
64         {
65             ListeDeSprite.Remove(unSprite);
66         }
67
```

```
68          /// <summary>
69          /// Trie les Sprites selon leur calque et leur profondeur
70          /// </summary>
71          public void Trier()
72          {
73              ListeDeSprite = ListeDeSprite.OrderBy(s => ↩
                    s.NumeroCalque).ThenBy(s => s.Profondeur).ToList<Sprite>();
74          }
75
76          /// <summary>
77          /// S'occupe de convertir la liste de Sprite pour quelle soit ↩
                serializable
78          /// </summary>
79          /// <returns>Une liste de SpriteSerializable</returns>
80          public SpritesSerializables EnListeSerializable()
81          {
82              SpritesSerializables listeSerializable = new ↩
                    SpritesSerializables();
83              foreach (Sprite unSprite in ListeDeSprite)
84              {
85                  listeSerializable.Ajouter(unSprite.EnSpriteSerializable());
86              }
87              return listeSerializable;
88          }
89          #endregion
90      }
91  }
```

Listing 7 – ./files/Sprites.cs

0.0.8 SpriteSerializable.cs

```
1   /*
2
3  Auteur      : JAUCH Walter
4
5  Date        : 09.06.2020
6
7  Version     : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier     : SpriteSerializable.cs
16
17 */
18 using System.Drawing;
19 using System.Windows.Forms;
20
21 namespace WF_LogoGo
22 {
23     public class SpriteSerializable
24     {
25         #region Variables d'instance
26
27         private int _couleur;
28         private Point _location;
29         private Size _size;
30         private float _epaisseurPen;
31         private bool _remplir;
32         private int _numeroCalque;
33         private int _profondeurParCalque;
34         private string _nom;
35         private int _idType;
36         private PointF[] _trace;
37         private string _texte;
38         private string _nomPpolice;
39         private int _taillePolice;
40
41
42         private enum Types { Carre = 1, Rond = 2, Triangle = 3, Texte = 4, Polygone = 5 };
43         #endregion
44
45         #region Propriétés
46
47         /// <summary>
48         /// Couleur du pinceau / crayon du sprite
49         /// </summary>
50         public int Couleur { get => _couleur; set => _couleur = value; }
51
52         /// <summary>
53         /// Position à laquelle le sprite doit être
54         /// </summary>
55         public Point Location { get => _location; set => _location = value; }
56
57         /// <summary>
58         /// Épaisseur du crayon
59         /// </summary>
60         public float EpaisseurPen { get => _epaisseurPen; set => _epaisseurPen = value; }
61
62         /// <summary>
63         /// Bool permettant de savoir s'il faut remplir ou non un sprite
64         /// </summary>
65         public bool Remplir { get => _remplir; set => _remplir = value; }
```

```
66
67          /// <summary>
68          /// Calque sur lequel le sprite est dessiné
69          /// </summary>
70          public int NumeroCalque { get => _numeroCalque; set => ↩
                _numeroCalque = value; }
71
72          /// <summary>
73          /// Profondeur qui définit l'ordre d'apparition des sprites ↩
                dans un même calque
74          /// </summary>
75          public int ProfondeurParCalque { get => _profondeurParCalque; ↩
                set => _profondeurParCalque = value; }
76
77          /// <summary>
78          /// Nom du sprite
79          /// </summary>
80          public string Nom { get => _nom; set => _nom = value; }
81
82          /// <summary>
83          /// Identdifiant permettant de savoir quel type de sprite nous ↩
                avons
84          /// </summary>
85          public int IdType { get => _idType; set => _idType = value; }
86
87          /// <summary>
88          /// Certains sprites on besoin d'un tracé pour ce dessiner
89          /// </summary>
90          public PointF[] Trace { get => _trace; set => _trace = value; }
91
92
93          /// <summary>
94          /// Certains sprites ont besoin d'un texte
95          /// </summary>
96          public string TexteAEcrire { get => _texte; set => _texte = ↩
                value; }
97
98          /// <summary>
99          /// Certains sprites ont besoin d'un nom de police
100         /// </summary>
101         public string NomPolice { get => _nomPolice; set => _nomPolice ↩
                = value; }
102
103         /// <summary>
104         /// Certains sprites ont besoin d'une taille de police
105         /// </summary>
106         public int TaillePolice { get => _taillePolice; set => ↩
                _taillePolice = value; }
107
108         /// <summary>
109         /// Taille du sprite
110         /// </summary>
111         public Size Size { get => _size; set => _size = value; }
112         #endregion
113
114         #region Constructeurs
115
116         public SpriteSerializable()
117         {
118             //La copie se fait dans le méthode AttribuerValeursProprietes
119             //qui est appellée de l'extérieur.
120         }
121         #endregion
122
123         #region Méthodes
124         /// <summary>
125         /// Crée le bon type de Sprite en fonction de Type
126         /// </summary>
127         /// <param name="parent"></param>
128         /// <returns>Un objet Sprite</returns>
129         public Sprite EnSprite(Form parent)
130         {
```

```
131             switch (IdType)
132             {
133                 case (int)Types.Carre:
134                     return new Carre(this, parent);
135                 case (int)Types.Rond:
136                     return new Rond(this, parent);
137                 case (int)Types.Triangle:
138                     return new Triangle(this, parent);
139                 case (int)Types.Texte:
140                     return new Texte(this, parent);
141                 case (int)Types.Polygone:
142                     return new Polygone(this, parent);
143                 default:
144                     return new Carre(this, parent);
145             }
146         }
147 
148         /// <summary>
149         /// Attribue les valeur d'un objet Sprite au SpriteSerializable
150         /// </summary>
151         /// <param name="s">Sprite à "copier"</param>
152         public void AttribuerValeursProprietes(Sprite s)
153         {
154             Couleur = s.Couleur.ToArgb();
155             Location = s.Location;
156             Size = s.Size;
157             EpaisseurPen = s.EpaisseurPen;
158             Remplir = s.Remplir;
159             NumeroCalque = s.NumeroCalque;
160             ProfondeurParCalque = s.Profondeur;
161             Nom = s.Name;
162             IdType = s.IdType;
163             Trace = s.Trace;
164             TexteAEcrire = s.TexteAEcrire;
165             TaillePolice = s.TaillePolice;
166             NomPolice = s.NomPolice;
167         }
168         #endregion
169     }
170 }
```

Listing 8 – ./files/SpriteSerializable.cs

0.0.9 SpritesSerializables.cs

```
1   /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : SpritesSerializables.cs
16
17 */
18 using System.Collections.Generic;
19 using System.Windows.Forms;
20
21 namespace WF_LogoGo
22 {
23     public class SpritesSerializables
24     {
25         #region Variables d'instance
26
27         private List<SpriteSerializable> _listeDeSpriteSerializable;
28         #endregion
29
30         #region Propriétés
31         /// <summary>
32         /// Liste de tous les objets SpritesSerializables.
33         /// </summary>
34         public List<SpriteSerializable> ListeDeSpriteSerializable { get => _listeDeSpriteSerializable; private set => _listeDeSpriteSerializable = value; }
35         #endregion
36
37         #region Contructeurs
38         public SpritesSerializables()
39         {
40             ListeDeSpriteSerializable = new List<SpriteSerializable>();
41         }
42         #endregion
43
44         #region Méthodes
45
46         /// <summary>
47         /// Ajoute à la liste le SpriteSerializable spécifié
48         /// /// </summary>
49         /// <param name="s">le SpriteSerializable à ajouter</param>
50         public void Ajouter(SpriteSerializable s)
51         {
52             ListeDeSpriteSerializable.Add(s);
53         }
54
55         /// <summary>
56         /// Crée un objet Sprites à partir de ListeDeSpriteSerializable
57         /// </summary>
58         /// <param name="parent">Form servant de parent</param>
59         /// <returns>Un objet Sprites</returns>
60         public Sprites EnSprites(Form parent)
61         {
62             Sprites ListeDeSprite = new Sprites();
63             foreach (SpriteSerializable unSpriteSerializable in ListeDeSpriteSerializable)
64             {
65                 ListeDeSprite.Ajouter(unSpriteSerializable.EnSprite(parent));
```

```
66                 }
67                 return ListeDeSprite;
68             }
69             #endregion
70         }
71 }
```

Listing 9 – ./files/SpritesSerializables.cs

0.0.10 Logo.cs

```
1    /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : Logo.cs
16
17 */
18 using System;
19 using System.IO;
20 using System.Windows.Forms;
21 using System.Xml.Serialization;
22
23 namespace WF_LogoGo
24 {
25     [Serializable()]
26     public class Logo
27     {
28         #region Variables d'instance
29
30         private Sprites _sprites;
31         private SpritesSerializables _spritesSerializables;
32         private Sprite _spriteChoisi;
33         private string _nomFichier;
34         private Form _parent;
35         #endregion
36
37         #region Constantes
38
39         /// <summary>
40         /// Nom du logo s'il n'est pas changé lors de la sauvegarde
41         /// </summary>
42         const string NOM_PAR_DEFAUT = "Logo.xml";
43         #endregion
44
45         #region Propriétés
46
47
48         /// <summary>
49         /// Nom auquel le fichier ets enregistré
50         /// </summary>
51         public string NomFichier { get => _nomFichier; set => _nomFichier = value; }
52
53         /// <summary>
54         /// Sprite sur lequel l'utilisateur a cliqué pour la dernière fois
55         /// </summary>
56         [XmlIgnore]
57         public Sprite SpriteChoisi { get => _spriteChoisi; set => _spriteChoisi = value; }
58
59         /// <summary>
60         /// Liste de sprites serializables
61         /// </summary>
62         public SpritesSerializables SpritesSerializables { get => _spritesSerializables; set => _spritesSerializables = value; }
63
64         /// <summary>
65         /// Liste qui contient tous les sprites créés par l'utilisateur
```

```
66          /// </summary>
67          [XmlIgnore]
68          public Sprites Sprites { get => _sprites; set => _sprites = ↩
                value; }
69          #endregion
70
71          #region Constructeurs
72          public Logo(Form parent)
73          {
74              Sprites = new Sprites();
75              this._parent = parent;
76              NomFichier = NOM_PAR_DEFAUT;
77          }
78
79          public Logo()
80          {
81              SpritesSerializables = new SpritesSerializables();
82              Sprites = new Sprites();
83              NomFichier = NOM_PAR_DEFAUT;
84
85          }
86          #endregion
87
88          #region Méthodes
89
90
91          /// <summary>
92          /// Sauvegarde de Logo dans l'état
93          /// </summary>
94          public void Enregistrer(string nomFichier)
95          {
96              NomFichier = nomFichier;
97              XMLSerialize();
98          }
99
100
101         /// <summary>
102         /// Récupère un projet sauvegardé
103         /// </summary>
104         public void Charger(string nomFichier)
105         {
106             if (nomFichier != null)
107             {
108                 NomFichier = nomFichier;
109             }
110             Sprites = null;
111             XMLDeserialize();
112         }
113
114         /// <summary>
115         /// Serialise le Logo en un fichier XML
116         /// </summary>
117         private void XMLSerialize()
118         {
119             SpritesSerializables = Sprites.EnListeSerializable();
120             Stream stream = File.Open(NomFichier, FileMode.Create);
121             XmlSerializer formatter = new XmlSerializer(typeof(Logo));
122             formatter.Serialize(stream, this);
123             stream.Close();
124         }
125
126         /// <summary>
127         /// Deserialise un fichier XML
128         /// </summary>
129         public void XMLDeserialize()
130         {
131             Stream stream = File.Open(NomFichier, FileMode.Open);
132             XmlSerializer formatter = new XmlSerializer(typeof(Logo));
133             Logo obj = (Logo)formatter.Deserialize(stream);
134             stream.Close();
135
136             this.Sprites = new Sprites();
```

```
137             Sprites = obj._spritesSerializables.EnSprites(_parent);
138             NomFichier = obj.NomFichier;
139         }
140 
141 
142         /// <summary>
143         /// Ajoute le Sprite spécifié
144         /// </summary>
145         /// <param name="s">Sprite à ajouter</param>
146         public void AjouterSprite(Sprite s)
147         {
148             Sprites.Ajouter(s);
149         }
150 
151         /// <summary>
152         /// Supprime le Sprite spécifié
153         /// </summary>
154         /// <param name="s">Sprite à supprimer</param>
155         public void SupprimerSprite(Sprite s)
156         {
157             Sprites.Supprimer(s);
158         }
159 
160         /// <summary>
161         /// Trie les Sprites
162         /// </summary>
163         public void TrierSprites()
164         {
165             Sprites.Trier();
166         }
167         #endregion
168     }
169 }
```

Listing 10 – ./files/Logo.cs

0.0.11 frmLogoGo.cs

```
1    /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : frmLogoGo.cs
16
17 */
18 using System;
19 using System.Drawing;
20 using System.Windows.Forms;
21
22 namespace WF_LogoGo
23 {
24     public partial class frmLogoGo : Form
25     {
26         Logo MonLogo;
27
28         private int CalqueChoisi { get => Convert.ToInt32(lsbCalques.SelectedItem.ToString().Substring(lsbCalques.S
               - 1)); }
29
30         #region Constructeur
31         public frmLogoGo()
32         {
33             InitializeComponent();
34             DoubleBuffered = true;
35             MonLogo = new Logo(this);
36         }
37         #endregion
38
39         #region Evenements
40
41         /// <summary>
42         /// Créé un nouveau objet de txpe Carre
43         /// </summary>
44         /// <param name="sender"></param>
45         /// <param name="e"></param>
46         private void btnCarre_Click(object sender, EventArgs e)
47         {
48             MonLogo.SpriteChoisi = new Carre(this, CalqueChoisi);
49             Dessine(MonLogo.SpriteChoisi);
50             Invalidate();
51         }
52
53         /// <summary>
54         /// Créé un nouveau objet de txpe Rond
55         /// </summary>
56         /// <param name="sender"></param>
57         /// <param name="e"></param>
58         private void btnRond_Click(object sender, EventArgs e)
59         {
60             MonLogo.SpriteChoisi = new Rond(this, CalqueChoisi);
61             Dessine(MonLogo.SpriteChoisi);
62             Invalidate();
63         }
64
65         /// <summary>
66         /// Créé un nouveau objet de txpe Triangle
```

```
67          /// </summary>
68          /// <param name="sender"></param>
69          /// <param name="e"></param>
70          private void btnTriangle_Click(object sender, EventArgs e)
71          {
72              MonLogo.SpriteChoisi = new Triangle(this, CalqueChoisi);
73              Dessine(MonLogo.SpriteChoisi);
74              Invalidate();
75          }
76
77          /// <summary>
78          /// Créé un nouveau objet de txpe Texte
79          /// </summary>
80          /// <param name="sender"></param>
81          /// <param name="e"></param>
82          private void btnTexte_Click(object sender, EventArgs e)
83          {
84              MonLogo.SpriteChoisi = new Texte(this, CalqueChoisi);
85              Dessine(MonLogo.SpriteChoisi);
86              Invalidate();
87          }
88
89          /// <summary>
90          /// Créé un nouveau polygone
91          /// </summary>
92          /// <param name="sender"></param>
93          /// <param name="e"></param>
94          private void btnPolygone_Click(object sender, EventArgs e)
95          {
96              Polygone p = new Polygone(this, 1);
97              frmCreerPolygone frmCreerPolygon = new frmCreerPolygone(p);
98
99              if (frmCreerPolygon.ShowDialog() == DialogResult.OK)
100             {
101                 p.Trace = frmCreerPolygon.Trace;
102                 MonLogo.SpriteChoisi = p;
103                 Dessine(MonLogo.SpriteChoisi);
104                 Invalidate();
105             }
106         }
107
108         /// <summary>
109         /// Affiche les proriétés du sprite sur lequel on a cliqué
110         /// </summary>
111         /// <param name="sender"></param>
112         /// <param name="e"></param>
113         private void ProprietesClick(object sender, EventArgs e)
114         {
115             MonLogo.SpriteChoisi = (sender as Sprite);
116             AfficherProprietes();
117         }
118
119         /// <summary>
120         /// Au chargement de la fiche, met à jour certains  éléments.
121         /// </summary>
122         /// <param name="sender"></param>
123         /// <param name="e"></param>
124         private void frmLogoGo_Load(object sender, EventArgs e)
125         {
126             //Par défaut, le premier élément est choisi
127             lsbCalques.SelectedIndex = 0;
128         }
129
130         /// <summary>
131         /// Applique les modifications de hauteur lorsque l'on quitte
132         /// la textbox au sprite choisi
133         /// </summary>
134         /// <param name="sender"></param>
135         /// <param name="e"></param>
136         private void tbxHauteur_Leave(object sender, EventArgs e)
137         {
```

```
138             Size nouvelleTaille = new ↩
                    Size(Convert.ToInt32(tbxLargeur.Text), ↩
                    Convert.ToInt32(tbxHauteur.Text));
139             MonLogo.SpriteChoisi.Size = nouvelleTaille;
140             Invalidate();
141         }
142
143         /// <summary>
144         /// Applique les modifications de taille lorsque l'on quitte
145         /// la textbox au sprite choisi
146         /// </summary>
147         /// <param name="sender"></param>
148         /// <param name="e"></param>
149         private void tbxLargeur_Leave(object sender, EventArgs e)
150         {
151             Size nouvelleTaille = new ↩
                    Size(Convert.ToInt32(tbxLargeur.Text), ↩
                    Convert.ToInt32(tbxHauteur.Text));
152             MonLogo.SpriteChoisi.Size = nouvelleTaille;
153             Invalidate();
154         }
155
156         /// <summary>
157         /// Appelle Dessine() et change de calque le sprite en question
158         /// </summary>
159         /// <param name="sender"></param>
160         /// <param name="e"></param>
161         private void cmbCalque_SelectedIndexChanged(object sender, ↩
                EventArgs e)
162         {
163             ComboBox cmb = (sender as ComboBox);
164             MonLogo.SpriteChoisi.NumeroCalque = ↩
                    Convert.ToInt32(cmb.SelectedItem.ToString().Substring(cmb.SelectedIte
                    - 1)); // Ici, utilisation d'un sender --> je prévois le ↩
                    texte qui aura une autre cmb mais même event
165             Dessine();
166         }
167
168         /// <summary>
169         /// Change la profondeur par calque du sprite choisi en fonction
170         /// de la valeur du numericUpDown
171         /// </summary>
172         /// <param name="sender"></param>
173         /// <param name="e"></param>
174         private void nudProfondeur_ValueChanged(object sender, ↩
                EventArgs e)
175         {
176             NumericUpDown nudProf = (sender as NumericUpDown);
177             MonLogo.SpriteChoisi.Profondeur = (int)nudProf.Value;
178             MonLogo.TrierSprites();
179             Dessine();
180             Invalidate();
181         }
182
183         /// <summary>
184         /// Définit si le sprite sera remplit ou non.
185         /// Active/Désactive le champ nudEpaisseur selon le choix.
186         /// </summary>
187         /// <param name="sender"></param>
188         /// <param name="e"></param>
189         private void chkRemplir_CheckedChanged(object sender, EventArgs e)
190         {
191             if (MonLogo.SpriteChoisi != null)
192             {
193                 if (chkRemplir.Checked)
194                 {
195                     MonLogo.SpriteChoisi.Remplir = true;
196                     nudEpaisseur.Enabled = false;
197                 }
198                 else
199                 {
200                     MonLogo.SpriteChoisi.Remplir = false;
```

```
                    nudEpaisseur.Enabled = true;
                }
            }
            Invalidate();
        }

        /// <summary>
        /// Ouvre le colorDialog et applique la couleur choisie
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnCouleur_Click(object sender, EventArgs e)
        {
            colorDialog.ShowDialog();
            btnCouleur.BackColor = colorDialog.Color;
            MonLogo.SpriteChoisi.Couleur = colorDialog.Color;
            Invalidate();
        }

        /// <summary>
        /// Applique les modifications de position lorsque l'on quitte
        /// la textbox au sprite choisi
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void tbxPos_Leave(object sender, EventArgs e)
        {
            Point nouvelleLocation = new ↩
               Point(Convert.ToInt32(tbxPosX.Text), ↩
               Convert.ToInt32(tbxPosY.Text));
            MonLogo.SpriteChoisi.Location = nouvelleLocation;
            Invalidate();
        }

        /// <summary>
        /// Applique les modifications de position lorsque l'on quitte
        /// la textbox au sprite choisi
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void tbxPosTexte_Leave(object sender, EventArgs e)
        {
            Point nouvelleLocation = new ↩
               Point(Convert.ToInt32(tbxPosXTexte.Text), ↩
               Convert.ToInt32(tbxPosYTexte.Text));
            MonLogo.SpriteChoisi.Location = nouvelleLocation;
            Invalidate();
        }

        /// <summary>
        /// Applique le changement d'épaisseur au sprite choisi
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void nudEpaisseur_ValueChanged(object sender, EventArgs e)
        {
            MonLogo.SpriteChoisi.EpaisseurPen = ↩
               Convert.ToInt32(nudEpaisseur.Value);
            Invalidate();
        }

        /// <summary>
        /// Suprrime le sprite actuellement selectionné
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void btnSupprimerSprite_Click(object sender, EventArgs e)
        {
            //Suppression
            Paint -= MonLogo.SpriteChoisi.SpritePaint;
            Controls.Remove(MonLogo.SpriteChoisi);
            MonLogo.SupprimerSprite(MonLogo.SpriteChoisi);
```

```
268             MonLogo.SpriteChoisi = null;
269             //Mise à jour des infos affichées
270             ReinitialiserProprietes();
271             Invalidate();
272         }
273 
274 
275         /// <summary>
276         /// Change le texte de l'objet Texte
277         /// </summary>
278         /// <param name="sender"></param>
279         /// <param name="e"></param>
280         private void tbxTexte_Leave(object sender, EventArgs e)
281         {
282             Texte texteAChanger = (Texte)MonLogo.SpriteChoisi;
283             if (tbxTexte.Text != string.Empty)
284             {
285                 texteAChanger.ChangerTexte(tbxTexte.Text);
286                 Invalidate();
287             }
288         }
289 
290         /// <summary>
291         /// Change la taille de la police du texte
292         /// </summary>
293         /// <param name="sender"></param>
294         /// <param name="e"></param>
295         private void nudFontSize_ValueChanged(object sender, EventArgs e)
296         {
297             Texte t = (Texte)MonLogo.SpriteChoisi;
298             t.TaillePolice = (int)nudFontSize.Value;
299             Invalidate();
300         }
301 
302         /// <summary>
303         /// Valide les modifications des textbox si la touche enter
304         /// est appuyée
305         /// </summary>
306         /// <param name="sender"></param>
307         /// <param name="e"></param>
308         private void Proprietes_KeyDown(object sender, KeyEventArgs e)
309         {
310             TextBox t = (sender as TextBox);
311             if (e.KeyCode == Keys.Enter)
312             {
313                 ProcessTabKey(true);
314             }
315         }
316 
317         /// <summary>
318         /// Interdit les lettres dans les textbox en n'autorisant que
319         /// les chiffres et les contrôles
320         /// </summary>
321         /// <param name="sender"></param>
322         /// <param name="e"></param>
323         private void Proprietes_KeyPress(object sender, ↩
              KeyPressEventArgs e)
324         {
325             if (!char.IsNumber(e.KeyChar) && (!char.IsControl(e.KeyChar)))
326             {
327                 e.Handled = true;
328             }
329         }
330 
331         /// <summary>
332         /// Change la transaprence de tous les sprites d'un calque
333         /// </summary>
334         /// <param name="sender"></param>
335         /// <param name="e"></param>
336         private void nudTransparenceCalque_ValueChanged(object sender, ↩
              EventArgs e)
337         {
```

```
338             foreach (Sprite unSprite in MonLogo.Sprites.ListeDeSprite)
339             {
340                 if (unSprite.NumeroCalque == CalqueChoisi)
341                 {
342                     unSprite.ChangerTransparence((int)(sender as ↩
                        NumericUpDown).Value);
343                 }
344             }
345 
346             Invalidate();
347         }
348 
349         /// <summary>
350         /// Mets à jour la transparence
351         /// </summary>
352         /// <param name="sender"></param>
353         /// <param name="e"></param>
354         private void lsbCalques_SelectedIndexChanged(object sender, ↩
            EventArgs e)
355         {
356             foreach (Sprite unSprite in MonLogo.Sprites.ListeDeSprite)
357             {
358                 if (unSprite.NumeroCalque == CalqueChoisi)
359                 {
360                     nudTransparenceCalque.Value = unSprite.AlphaCouleur;
361                     break;
362                 }
363             }
364         }
365 
366         /// <summary>
367         /// Ouvre une form montrant le logo qui sera exporté.
368         /// Lorsque l'utilisateur confirme sa volonter de sauver
369         /// le logo, un saveFileDialog s'affiche pour qui choisise
370         /// où le fichier sera enregistré.
371         /// </summary>
372         /// <param name="sender"></param>
373         /// <param name="e"></param>
374         private void msExporter_Click(object sender, EventArgs e)
375         {
376             if (MonLogo.Sprites.EstVide)
377             {
378                 MessageBox.Show("Votre␣projet␣est␣vide.␣Essayer␣de␣↩
                    créer␣votre␣logo␣avant␣d'exporter␣!");
379             }
380             else
381             {
382                 frmExporterLogo frmExporter = new ↩
                    frmExporterLogo(MonLogo);
383 
384                 if (frmExporter.ShowDialog() == DialogResult.OK)
385                 {
386                     saveFileDialog = new SaveFileDialog();
387                     saveFileDialog.Filter = ↩
                        "BMP|*.bmp|GIF|*.gif|JPG|*.jpg;*.jpeg|PNG|*.png|TIFF|*.tif;*
388                                             + "All␣Graphics␣↩
                                                Types|*.bmp;*.jpg;*.jpeg;*.png;*.tif
389                     saveFileDialog.FileName = "Logo";
390 
391                     if (saveFileDialog.ShowDialog() == DialogResult.OK)
392                     {
393                         frmExporter.LogoFinal.Save(saveFileDialog.FileName);
394                     }
395                 }
396             }
397         }
398 
399         /// <summary>
400         /// Ouvre un openFileDialog pour que l'utilisateur choisisse
401         /// quel fichier il voudrait charger.
402         /// </summary>
403         /// <param name="sender"></param>
```

```
404         /// <param name="e"></param>
405         private void msOuvrir_Click(object sender, EventArgs e)
406         {
407             try
408             {
409                 openFileDialog = new OpenFileDialog();
410                 openFileDialog.Filter = "XML-File | *.xml";
411 
412                 if (openFileDialog.ShowDialog() == DialogResult.OK)
413                 {
414                     MonLogo.Charger(openFileDialog.FileName);
415 
416                     MonLogo.TrierSprites();
417                     foreach (Sprite unSprite in MonLogo.Sprites.ListeDeSprite)
418                     {
419                         unSprite.Click += this.ProprietesClick;
420                         Paint += unSprite.SpritePaint;
421                         this.Controls.Add(unSprite);
422                     }
423                     Invalidate();
424                 }
425             }
426             catch (Exception)
427             {
428                 MessageBox.Show("Il semblerait que vous tentiez d'ouvrir un fichier invalide. Le fichier contient peut-être une erreur.");
429                 Application.Restart();
430             }
431         }
432 
433         /// <summary>
434         /// Ouvre un saveFileDialog pour que l'utilisateur choisisse
435         /// où le fichier sera enregistré.
436         /// </summary>
437         /// <param name="sender"></param>
438         /// <param name="e"></param>
439         private void msEnregistrer_Click(object sender, EventArgs e)
440         {
441             if (MonLogo.Sprites.EstVide)
442             {
443                 MessageBox.Show("Votre projet est vide. Essayer de créer votre logo avant d'enregistrer !");
444             }
445             else
446             {
447                 saveFileDialog = new SaveFileDialog();
448                 saveFileDialog.Filter = "XML-File | *.xml";
449                 saveFileDialog.FileName = "Logo.xml";
450 
451                 if (saveFileDialog.ShowDialog() == DialogResult.OK)
452                 {
453                     MonLogo.Enregistrer(saveFileDialog.FileName);
454                 }
455             }
456         }
457 
458         /// <summary>
459         /// Ouvre une page web sur la documentation du projet
460         /// </summary>
461         /// <param name="sender"></param>
462         /// <param name="e"></param>
463         private void aideToolStripMenuItem_Click(object sender, EventArgs e)
464         {
465             System.Diagnostics.Process.Start("https://github.com/walterjch/LogoGo/tr
466         }
467 
468         /// <summary>
469         /// À la fermeture de frmLogo, affiche un messagebox. Ferme l'application ou non en fonction du dialogResilt
```

```
470         /// </summary>
471         /// <param name="sender"></param>
472         /// <param name="e"></param>
473         private void frmLogoGo_FormClosing(object sender, ↩
                FormClosingEventArgs e)
474         {
475             if (MonLogo.Sprites != null && ↩
                    MonLogo.Sprites.ListeDeSprite.Count != 0)
476             {
477                 DialogResult resultat = MessageBox.Show("Attention,␣la␣↩
                        fermeture␣de␣LogoGo␣aura␣pour␣résultat␣la␣perte␣de␣↩
                        votre␣projet␣si␣vous␣ne␣l'avez␣pas␣enregistré.␣Si␣↩
                        vous␣êtes␣sûr␣de␣l'avoir␣sauvegardé,␣cliquez␣sur␣↩
                        OK.", "Fermeture␣de␣l'application", ↩
                        MessageBoxButtons.OKCancel);
478                 if (resultat == DialogResult.OK)
479                 {
480                     e.Cancel = false;
481                     Application.Exit();
482                 }
483                 else
484                 {
485                     e.Cancel = true;
486                 }
487             }
488 
489         }
490         #endregion
491 
492         #region Méthodes
493         /// <summary>
494         /// Gère la création d'un nouveau sprite s'il est spécifié.
495         /// Efface et affiche à nouveau les sprites dans le bon ordre
496         /// </summary>
497         /// <param name="s"></param>
498         public void Dessine(Sprite s = null)
499         {
500             if (s != null)
501             {
502                 MonLogo.AjouterSprite(s);
503                 s.Click += this.ProprietesClick;
504             }
505 
506             MonLogo.TrierSprites();
507 
508             foreach (Sprite unSprite in MonLogo.Sprites.ListeDeSprite)
509             {
510                 Paint -= unSprite.SpritePaint;
511                 this.Controls.Remove(unSprite);
512                 Paint += unSprite.SpritePaint;
513                 this.Controls.Add(unSprite);
514             }
515 
516         }
517 
518 
519         /// <summary>
520         /// Affiche les propriétés du sprite actuellement sélectionné
521         /// </summary>
522         public void AfficherProprietes()
523         {
524 
525             if (MonLogo.SpriteChoisi is Texte)
526             {
527                 pnlProprietesStandard.Visible = false;
528                 pnlProprietesTexte.Visible = true;
529 
530                 //Affichage de la position du Sprite en question
531                 tbxPosXTexte.Enabled = true;
532                 tbxPosYTexte.Enabled = true;
533                 tbxPosXTexte.Text = ↩
                        MonLogo.SpriteChoisi.Location.X.ToString();
```

```
534                 tbxPosYTexte.Text = ↩
                        MonLogo.SpriteChoisi.Location.Y.ToString();
535
536                 //Affichage du texte du Sprite en question
537                 tbxTexte.Enabled = true;
538                 tbxTexte.Text = MonLogo.SpriteChoisi.ToString();
539
540                 //Affichage de sa couleur
541                 btnCouleurTexte.BackColor = MonLogo.SpriteChoisi.Couleur;
542                 btnCouleurTexte.Enabled = true;
543
544
545                 //Affichage de la taille de police
546                 Texte t = (Texte)MonLogo.SpriteChoisi;
547                 nudFontSize.Enabled = true;
548                 nudFontSize.Value = t.TaillePolice;
549
550                 //Affichage de la profondeur
551                 nudProfondeurTexte.Enabled = true;
552                 nudProfondeurTexte.Value = ↩
                        MonLogo.SpriteChoisi.Profondeur;
553
554
555                 //Affichage du calques dans la combobox
556                 cmbCalqueTexte.Enabled = true;
557                 foreach (string calque in cmbCalque.Items)
558                 {
559                     if (calque == "Calque␣" + ↩
                            MonLogo.SpriteChoisi.NumeroCalque)
560                     {
561                         cmbCalqueTexte.SelectedIndex = ↩
                                cmbCalqueTexte.FindStringExact(calque);
562                     }
563                 }
564
565                 btnSupprimerTexte.Enabled = true;
566
567             }
568             else
569             {
570                 pnlProprietesTexte.Visible = false;
571                 pnlProprietesStandard.Visible = true;
572                 //Affichage de la position du Sprite en question
573                 tbxPosX.Enabled = true;
574                 tbxPosY.Enabled = true;
575                 tbxPosX.Text = MonLogo.SpriteChoisi.Location.X.ToString();
576                 tbxPosY.Text = MonLogo.SpriteChoisi.Location.Y.ToString();
577
578
579                 //Affichage de la taille
580                 if(!(MonLogo.SpriteChoisi is Polygone))
581                 {
582                     tbxHauteur.Enabled = true;
583                     tbxLargeur.Enabled = true;
584                 }
585                 else
586                 {
587                     tbxHauteur.Enabled = false;
588                     tbxLargeur.Enabled = false;
589                 }
590                 tbxHauteur.Text = MonLogo.SpriteChoisi.Height.ToString();
591                 tbxLargeur.Text = MonLogo.SpriteChoisi.Width.ToString();
592
593                 //Affichage de sa couleur
594                 btnCouleur.BackColor = MonLogo.SpriteChoisi.Couleur;
595                 btnCouleur.Enabled = true;
596
597                 //Affichage de la profondeur
598                 nudProfondeur.Enabled = true;
599                 nudProfondeur.Value = MonLogo.SpriteChoisi.Profondeur;
600
601
```

```
602                 //Affichage de l'épaisseur
603                 if (!MonLogo.SpriteChoisi.Remplir)
604                 {
605                     nudEpaisseur.Enabled = true;
606                 }
607                 nudEpaisseur.Value = ↩
                        Convert.ToInt32(MonLogo.SpriteChoisi.EpaisseurPen);
608 
609                 //Affichage du remplissement
610                 chkRemplir.Enabled = true;
611                 chkRemplir.Checked = MonLogo.SpriteChoisi.Remplir;
612 
613                 //Affichage du calques dans la combobox
614                 cmbCalque.Enabled = true;
615                 foreach (string calque in cmbCalque.Items)
616                 {
617                     if (calque == "Calque␣" + ↩
                            MonLogo.SpriteChoisi.NumeroCalque)
618                     {
619                         cmbCalque.SelectedIndex = ↩
                                cmbCalque.FindStringExact(calque);
620                     }
621                 }
622 
623                 btnSupprimerSprite.Enabled = true;
624 
625             }
626         }
627 
628         /// <summary>
629         /// Réinitialise les propriétés affichées
630         /// </summary>
631         public void ReinitialiserProprietes()
632         {
633 
634             // Partie Sprite
635             tbxPosX.Enabled = false;
636             tbxPosY.Enabled = false;
637             tbxPosX.Text = string.Empty;
638             tbxPosY.Text = string.Empty;
639 
640             tbxHauteur.Enabled = false;
641             tbxLargeur.Enabled = false;
642             tbxHauteur.Text = string.Empty;
643             tbxLargeur.Text = string.Empty;
644 
645             btnCouleur.BackColor = default(Color);
646             btnCouleur.Enabled = true;
647 
648             chkRemplir.Enabled = false;
649             chkRemplir.Checked = false;
650 
651             cmbCalque.Enabled = false;
652 
653             btnSupprimerSprite.Enabled = false;
654 
655             nudProfondeur.Enabled = false;
656 
657             nudEpaisseur.Enabled = false;
658 
659             //Partie Texte
660             tbxTexte.Enabled = false;
661             tbxTexte.Text = string.Empty;
662 
663             tbxPosXTexte.Enabled = false;
664             tbxPosYTexte.Enabled = false;
665             tbxPosXTexte.Text = string.Empty;
666             tbxPosYTexte.Text = string.Empty;
667 
668             btnCouleurTexte.BackColor = default(Color);
669             btnCouleurTexte.Enabled = true;
670 
```

```
671                 cmbCalqueTexte.Enabled = false;
672
673                 btnSupprimerTexte.Enabled = false;
674
675                 nudProfondeurTexte.Enabled = false;
676             }
677             #endregion
678         }
679 }
```

Listing 11 – ./files/frmLogoGo.cs

0.0.12 frmExporterLogo.cs

```
1    /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : frmExporterLogo.cs
16
17 */
18 using System;
19 using System.Drawing;
20 using System.Windows.Forms;
21
22 namespace WF_LogoGo
23 {
24     public partial class frmExporterLogo : Form
25     {
26         private Logo LogoAExporter;
27         public Image LogoFinal;
28         private Bitmap dessin;
29
30         const int BORDURE_OFFSET = 25;
31
32         //Cela sert à identifier les quatre coins de la picturebox afin
33         //de pouvoir la recadrer à la bonne taille du logo en entier.
34         private Point CoinSuperieurGauche;
35         private Point CoinSuperieurDroit;
36         private Point CoinInferieurDroit;
37         private Point CoinInferieurGauche;
38
39         public frmExporterLogo(Logo logoAExporer)
40         {
41             InitializeComponent();
42             LogoAExporter = logoAExporer;
43             dessin = new Bitmap(pbxResultatFinal.Width, pbxResultatFinal.Height);
44         }
45
46         private void ExporterLogo_Load(object sender, EventArgs e)
47         {
48             bool premierSprite = true;
49             Graphics g = Graphics.FromImage(dessin);
50             foreach (Sprite unSprite in LogoAExporter.Sprites.ListeDeSprite)
51             {
52                 //Repérage des extrémités du logo//
53                 //Pour le premier sprite, les coins du logo équivalent aux coins du sprite
54                 if (premierSprite)
55                 {
56                     CoinSuperieurGauche = unSprite.Location;
57                     CoinSuperieurDroit = new Point(unSprite.Location.X + unSprite.Width, +unSprite.Location.Y);
58                     CoinInferieurDroit = new Point(unSprite.Location.X + unSprite.Width, +unSprite.Location.Y + unSprite.Height);
59                     CoinInferieurGauche = new Point(unSprite.Location.X, +unSprite.Location.Y + unSprite.Height);
60                     premierSprite = false;
```

```
61                  }
62                  else
63                  {
64                      //Vérifie si le coin supérieur gauche doit changer
65                      if (unSprite.Location.X < CoinSuperieurGauche.X)
66                      {
67                          CoinSuperieurGauche = new ↩
                               Point(unSprite.Location.X, ↩
                               CoinSuperieurGauche.Y);
68                      }
69                      if (unSprite.Location.Y < CoinSuperieurGauche.Y)
70                      {
71                          CoinSuperieurGauche = new ↩
                               Point(CoinSuperieurGauche.X, ↩
                               unSprite.Location.Y);
72                      }
73
74                      //Vérifie si le coin supérieur droit doit changer
75                      if (unSprite.Location.X + unSprite.Width > ↩
                           CoinSuperieurDroit.X)
76                      {
77                          CoinSuperieurDroit = new ↩
                               Point(unSprite.Location.X + unSprite.Width, ↩
                               CoinSuperieurDroit.Y);
78                      }
79                      if (unSprite.Location.Y < CoinSuperieurDroit.Y)
80                      {
81                          CoinSuperieurDroit = new ↩
                               Point(CoinSuperieurDroit.X, ↩
                               unSprite.Location.Y - (unSprite.Location.Y - ↩
                               CoinSuperieurGauche.Y));
82                      }
83
84                      //Vérifie si le coin inférieur droit doit changer
85                      if (unSprite.Location.X + unSprite.Width > ↩
                           CoinInferieurDroit.X)
86                      {
87                          CoinInferieurDroit = new ↩
                               Point(unSprite.Location.X + unSprite.Width, ↩
                               CoinInferieurDroit.Y);
88                      }
89                      if (unSprite.Location.Y + unSprite.Height > ↩
                           CoinInferieurDroit.Y)
90                      {
91                          CoinInferieurDroit = new ↩
                               Point(CoinInferieurDroit.X, ↩
                               unSprite.Location.Y + unSprite.Height);
92                      }
93
94                      //Vérifie si le coin inférieur gauche doit changer
95                      if (unSprite.Location.X < CoinInferieurGauche.X)
96                      {
97                          CoinInferieurGauche = new ↩
                               Point(unSprite.Location.X, ↩
                               CoinInferieurGauche.Y);
98                      }
99                      if (unSprite.Location.Y + unSprite.Height > ↩
                           CoinInferieurGauche.Y)
100                     {
101                         CoinInferieurGauche = new ↩
                               Point(CoinInferieurGauche.X, ↩
                               unSprite.Location.Y + unSprite.Height);
102                     }
103                 }
104
105                 // On dessine le sprite
106                 unSprite.SpritePaintAvecGraphics(g);
107             }
108             //On resize la picturebox pour qu'elle fasse la même ↩
                    dimension que le logo
109
110             g.Dispose();
```

```
111             pbxResultatFinal.Image = dessin;
112             LogoFinal = RedimensionnerLogo(dessin, ↩
                    CoinSuperieurGauche.X, CoinSuperieurGauche.Y, ↩
                    CoinSuperieurDroit.X - CoinSuperieurGauche.X, ↩
                    CoinInferieurDroit.Y - CoinSuperieurDroit.Y);
113         }
114
115         //Trouvé sur : ↩
                https://stackoverflow.com/questions/13217156/cropping-picture-getting-rec
116         /// <summary>
117         /// Rogne une image donnée en une zone qu'on doit spécifier.
118         /// </summary>
119         /// <param name="original_image">Image que l'on veut ↩
                rogner</param>
120         /// <param name="x">Position X de là où on veut rogner ↩
                l'image</param>
121         /// <param name="y">Position Y de là où on veut rogner ↩
                l'image</param>
122         /// <param name="width">Largeur voulue</param>
123         /// <param name="height">Hauteur voulue</param>
124         /// <returns>Retourne l'image finale</returns>
125         public Bitmap RedimensionnerLogo(Bitmap original_image, int x, ↩
                int y, int width, int height)
126         {
127             Rectangle crop = new Rectangle(x - BORDURE_OFFSET, y - ↩
                    BORDURE_OFFSET, width + 2 * BORDURE_OFFSET, height + 2 * ↩
                    BORDURE_OFFSET);
128
129             var bmp = new Bitmap(crop.Width, crop.Height);
130             using (var gr = Graphics.FromImage(bmp))
131             {
132                 gr.DrawImage(original_image, new Rectangle(0, 0, ↩
                        bmp.Width, bmp.Height), crop, GraphicsUnit.Pixel);
133             }
134             return bmp;
135         }
136     }
137 }
```

Listing 12 – ./files/frmExporterLogo.cs

0.0.13 frmCreerPolygone.cs

```
1   /*
2
3  Auteur       : JAUCH Walter
4
5  Date         : 09.06.2020
6
7  Version      : 1.0
8
9  Description : LogoGo est une application permettant de créer des logos
10               à partir de certaines formes (carré, rond, texte, etc.).
11               L'utilisateur peut modfifier ces formes et il dipsose de calques.
12
13               Il est possible d'exporter, enregistrer, et ouvir un logo.
14
15 Fichier      : frmCreerPolygone.cs
16
17 */
18 using System;
19 using System.Collections.Generic;
20 using System.Drawing;
21 using System.Windows.Forms;
22
23 namespace WF_LogoGo
24 {
25     public partial class frmCreerPolygone : Form
26     {
27         private PointF[] _trace;
28         private List<PointF> _pointsATracer;
29         private Polygone NouveauPolygone;
30
31         public PointF[] Trace { get => _trace; private set => _trace = value; }
32
33         public frmCreerPolygone(Polygone p)
34         {
35             InitializeComponent();
36             _pointsATracer = new List<PointF>();
37             NouveauPolygone = p;
38         }
39
40         /// <summary>
41         /// Appelé lorsque l'utilisateur "clique" sur la PictureBox
42         /// </summary>
43         /// <param name="sender"></param>
44         /// <param name="e"></param>
45         private void pbxPolygone_MouseDown(object sender, MouseEventArgs e)
46         {
47             btnOK.Enabled = true;
48             _pointsATracer.Add(new PointF(e.X, e.Y));
49         }
50
51         /// <summary>
52         /// À l'appui du bouton OK, ajoute les points au tableau Trace.
53         /// </summary>
54         /// <param name="sender"></param>
55         /// <param name="e"></param>
56         private void btnOK_Click(object sender, EventArgs e)
57         {
58             Trace = new PointF[_pointsATracer.Count];
59             int compteur = 0;
60             foreach (PointF unPointF in _pointsATracer)
61             {
62                 Trace[compteur++] = new PointF(unPointF.X - _pointsATracer[0].X, unPointF.Y - _pointsATracer[0].Y);
63             }
64             this.DialogResult = System.Windows.Forms.DialogResult.OK;
65             Close();
```

```
66            }
67
68
69        }
70    }
```

Listing 13 – `./files/frmCreerPolygone.cs`