

# 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 . . . . .	2
0.0.2	Carre.cs . . . . .	6
0.0.3	Rond.cs . . . . .	8
0.0.4	Triangle.cs . . . . .	10
0.0.5	Polygone.cs . . . . .	12
0.0.6	Texte.cs . . . . .	15
0.0.7	Sprites.cs . . . . .	17
0.0.8	SpriteSerializable.cs . . . . .	19
0.0.9	SpritesSerializables.cs . . . . .	22
0.0.10	Logo.cs . . . . .	24
0.0.11	frmLogoGo.cs . . . . .	27
0.0.12	frmExporterLogo.cs . . . . .	38
0.0.13	frmCreerPolygone.cs . . . . .	41

## 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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : Sprite.cs
17
18  */
19 using System.Drawing;
20 using System.Windows.Forms;
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 = ↵
55             value; }
56
57         /// <summary>
58         /// Épaisseur du crayon
59         /// </summary>
60         public float EpaisseurPen { get => _epaisseurPen; set => ↵
61             _epaisseurPen = value; }
62
63         /// <summary>
64         /// Bool permettant de savoir s'il faut remplir ou non un sprite
65         /// </summary>
66         public bool Remplir { get => _remplir; set => _remplir = value; }
67
68         /// <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      /// Identifiant 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 ↵
186     dessinée</param>
187 public abstract void SpritePaintAvecGraphics(Graphics g);
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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : Carre.cs
17
18  */
19 using System;
20 using System.Collections.Generic;
21 using System.Drawing;
22 using System.Linq;
23 using System.Text;
24 using System.Threading.Tasks;
25 using System.Windows.Forms;
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 ↵
54         /// dessinée</param>
55         public override void SpritePaintAvecGraphics(Graphics g)
56         {
57             g.SmoothingMode = ↵
58                 System.Drawing.Drawing2D.SmoothingMode.HighQuality;
59             if (Remplir)
60             {
61                 SolidBrush b = new SolidBrush(Couleur);
62                 g.FillRectangle(b, new Rectangle(Location.X, ↵
63                     Location.Y, Size.Width, Size.Height));
64             }
65             else
66             {
67                 Pen p = new Pen(Couleur, EpaisseurPen);
68             }
69         }
70     }
71 }
```

```
66         g.DrawRectangle(p, new Rectangle(Location.X, ↵
67             Location.Y, Size.Width, Size.Height));
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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : Rond.cs
17
18  */
19 using System.Drawing;
20 using System.Windows.Forms;
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 ↵
48         /// dessinée</param>
49         public override void SpritePaintAvecGraphics(Graphics g)
50         {
51             g.SmoothingMode = ↵
52             System.Drawing.Drawing2D.SmoothingMode.HighQuality;
53             if (Remplir)
54             {
55                 SolidBrush b = new SolidBrush(Couleur);
56                 g.FillEllipse(b, new Rectangle(Location.X, Location.Y, ↵
57                 Size.Width + 1, Size.Height + 1));
58             }
59             else
60             {
61                 Pen p = new Pen(Couleur, EpaisseurPen);
62                 g.DrawEllipse(p, new Rectangle(Location.X, Location.Y, ↵
63                 Size.Width + 1, Size.Height + 1));
64             }
65         }
66     }
67     #endregion
68 }
```

---

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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : Triangle.cs
17
18  */
19 using System.Drawing;
20 using System.Windows.Forms;
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, ↵
38             parent)
39         {
40             IdType = 3;
41         }
42         #endregion
43
44         #region Méthodes
45
46         /// <summary>
47         /// Dessine un Triangle selon la propriété Remplir.
48         /// </summary>
49         /// <param name="g">Objet Graphics avec lequel la forme sera ↵
50         /// dessinée</param>
51         public override void SpritePaintAvecGraphics(Graphics g)
52         {
53             g.SmoothingMode = ↵
54                 System.Drawing.Drawing2D.SmoothingMode.HighQuality;
55             Point[] Trace =
56             {
57                 new Point(Location.X, Location.Y + Height),
58                 new Point(Location.X + Width / 2, Location.Y),
59                 new Point(Location.X + Width, Location.Y + Height),
60                 new Point(Location.X, Location.Y + Height)
61             };
62             if (Remplir)
63             {
64                 SolidBrush b = new SolidBrush(Couleur);
65                 g.FillPolygon(b, Trace);
66             }
67             else
68             {
69                 g.DrawPolygon(Pen, Trace);
70             }
71         }
72     }
73 }
```

---

```
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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : Polygone.cs
17
18  */
19 using System.Drawing;
20 using System.Windows.Forms;
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, ↵
60             parent)
61         {
62             IdType = 5;
63         }
64         #endregion
65
66         #region Méthodes
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, ←
128             CoinSuperieurDroit.Y);
129     }
130     //vérification coin inférieur droit
131     if (unPoint.X > CoinInferieurDroit.X)
132     {
133         CoinInferieurDroit = new PointF(unPoint.X, ←
134             CoinInferieurDroit.Y);
135     }
136     if (unPoint.Y > CoinInferieurDroit.Y)
137     {
138         CoinInferieurDroit = new ←
139             PointF(CoinInferieurDroit.X, unPoint.Y);
140     }
141     //vérification coin inférieur gauche
142     if (unPoint.Y > CoinInferieurGauche.Y)
143     {
144         CoinInferieurGauche = new ←
145             PointF(CoinInferieurGauche.X, unPoint.Y);
146     }
147     if (unPoint.Y > CoinInferieurDroit.Y)
148     {
149         CoinInferieurDroit = new ←
150             PointF(CoinInferieurDroit.X, unPoint.Y);
151     }
152     }
153     }
154     #endregion
155 }

```

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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier     : Texte.cs
17
18  */
19 using System.Drawing;
20 using System.Windows.Forms;
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 ↵
53         /// dessiné</param>
54         public override void SpritePaintAvecGraphics(Graphics g)
55         {
56             SolidBrush b = new SolidBrush(Couleur);
57             Font maPolice = new Font(NomPolice, TaillePolice, ↵
58                 FontStyle.Bold, GraphicsUnit.Pixel);
59             RedimensionnerPictureBox(maPolice, g);
60             g.DrawString(TexteAEcrire, maPolice, b, Location);
61         }
62
63         /// <summary>
64         /// Redimensionne la pictureBox en fonction de la police et du ↵
65         /// texte à écrire.
66         /// </summary>
67         /// <param name="uneFonte">Police utilisée</param>
68         /// <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 caractère à ←
76     /// afficher</param>
77     public void ChangerTexte(string nouveauTexte)
78     {
79         TexteAEcrire = nouveauTexte;
80     }
81
82     /// <summary>
83     /// Surcharge du ToString de Sprite.
84     /// </summary>
85     /// <returns>Retourne le texte</returns>
86     public override string ToString()
87     {
88         return TexteAEcrire;
89     }
90     #endregion
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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16 Fichier     : Sprites.cs
17
18 */
19 using System.Collections.Generic;
20 using System.Linq;
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; ↵
35             private set => _listeDeSprite = value; }
36         /// <summary>
37         /// Permet de savoir si la liste de sprites est vide.
38         /// </summary>
39         public bool EstVide { get => ListeDeSprite.Count == 0; }
40         #endregion
41
42         #region Constructeurs
43         public Sprites()
44         {
45             ListeDeSprite = new List<Sprite>();
46         }
47         #endregion
48
49         #region Méthodes
50
51         /// <summary>
52         /// Ajoute à la liste le sprite spécifié
53         /// </summary>
54         /// <param name="unSprite">le Sprite à ajouter</param>
55         public void Ajouter(Sprite unSprite)
56         {
57             ListeDeSprite.Add(unSprite);
58         }
59
60         /// <summary>
61         /// Supprime de la liste le sprite spécifié
62         /// </summary>
63         /// <param name="unSprite">le Sprite à supprimer</param>
64         public void Supprimer(Sprite unSprite)
65         {
66             ListeDeSprite.Remove(unSprite);
67         }
68     }
69 }
```

```

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

```

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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : SpriteSerializable.cs
17
18  */
19 using System.Drawing;
20 using System.Windows.Forms;
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 _nomPolice;
39         private int _taillePolice;
40
41
42         private enum Types { Carre = 1, Rond = 2, Triangle = 3, Texte = ↵
43             4, Polygone = 5 };
44         #endregion
45
46         #region Propriétés
47
48         /// <summary>
49         /// Couleur du pinceau / crayon du sprite
50         /// </summary>
51         public int Couleur { get => _couleur; set => _couleur = value; }
52
53         /// <summary>
54         /// Position à laquelle le sprite doit être
55         /// </summary>
56         public Point Location { get => _location; set => _location = ↵
57             value; }
58
59         /// <summary>
60         /// Épaisseur du crayon
61         /// </summary>
62         public float EpaisseurPen { get => _epaisseurPen; set => ↵
63             _epaisseurPen = value; }
64
65         /// <summary>
66         /// Bool permettant de savoir s'il faut remplir ou non un sprite
67         /// </summary>
68         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     /// Identifiant 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 ont 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 appelée 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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : SpritesSerializables.cs
17
18  */
19 using System.Collections.Generic;
20 using System.Windows.Forms;
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 ↵
35             => _listeDeSpriteSerializable; private set => ↵
36             _listeDeSpriteSerializable = value; }
37         #endregion
38
39         #region Constructeurs
40         public SpritesSerializables()
41         {
42             ListeDeSpriteSerializable = new List<SpriteSerializable>();
43         }
44         #endregion
45
46         #region Méthodes
47
48         /// <summary>
49         /// Ajoute à la liste le SpriteSerializable spécifié
50         /// </summary>
51         /// <param name="s">le SpriteSerializable à ajouter</param>
52         public void Ajouter(SpriteSerializable s)
53         {
54             ListeDeSpriteSerializable.Add(s);
55         }
56
57         /// <summary>
58         /// Crée un objet Sprites à partir de ListeDeSpriteSerializable
59         /// </summary>
60         /// <param name="parent">Form servant de parent</param>
61         /// <returns>Un objet Sprites</returns>
62         public Sprites EnSprites(Form parent)
63         {
64             Sprites ListeDeSprite = new Sprites();
65             foreach (SpriteSerializable unSpriteSerializable in ↵
66                 ListeDeSpriteSerializable)
67             {
68                 ListeDeSprite.Ajouter(unSpriteSerializable.EnSprite(parent));
69             }
70         }
71     }
72 }
```

---

```
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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : Logo.cs
17
18  */
19 using System;
20 using System.IO;
21 using System.Windows.Forms;
22 using System.Xml.Serialization;
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_DEFAULT = "Logo.xml";
43         #endregion
44
45         #region Propriétés
46
47         /// <summary>
48         /// Nom auquel le fichier ets enregistré
49         /// </summary>
50         public string NomFichier { get => _nomFichier; set => ↵
51             _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 => ↵
58             _spriteChoisi = value; }
59
60         /// <summary>
61         /// Liste de sprites serializables
62         /// </summary>
63         public SpritesSerializables SpritesSerializables { get => ↵
64             _spritesSerializables; set => _spritesSerializables = value; }
65
66         /// <summary>
67         /// 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_DEFAULT;
77     }
78
79     public Logo()
80     {
81         SpritesSerializables = new SpritesSerializables();
82         Sprites = new Sprites();
83         NomFichier = NOM_PAR_DEFAULT;
84     }
85     #endregion
86
87     #region Méthodes
88
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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : frmLogoGo.cs
17
18  */
19 using System;
20 using System.Drawing;
21 using System.Windows.Forms;
22 namespace WF_LogoGo
23 {
24     public partial class frmLogoGo : Form
25     {
26         Logo MonLogo;
27
28         private int CalqueChoisi { get => ↵
29             Convert.ToInt32(lsbCalques.SelectedItem.ToString().Substring(1, lsbCalques.SelectedIndex - 1)); }
30
31         #region Constructeur
32         public frmLogoGo()
33         {
34             InitializeComponent();
35             DoubleBuffered = true;
36             MonLogo = new Logo(this);
37         }
38         #endregion
39
40         #region Evenements
41
42         /// <summary>
43         /// Créé un nouveau objet de txpe Carre
44         /// </summary>
45         /// <param name="sender"></param>
46         /// <param name="e"></param>
47         private void btnCarre_Click(object sender, EventArgs e)
48         {
49             MonLogo.SpriteChoisi = new Carre(this, CalqueChoisi);
50             Dessine(MonLogo.SpriteChoisi);
51             Invalidate();
52         }
53
54         /// <summary>
55         /// Créé un nouveau objet de txpe Rond
56         /// </summary>
57         /// <param name="sender"></param>
58         /// <param name="e"></param>
59         private void btnRond_Click(object sender, EventArgs e)
60         {
61             MonLogo.SpriteChoisi = new Rond(this, CalqueChoisi);
62             Dessine(MonLogo.SpriteChoisi);
63             Invalidate();
64         }
65
66         /// <summary>
67         /// 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 propriétés 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 ←
139             Size(Convert.ToInt32(tbxLargeur.Text), ←
140                 Convert.ToInt32(tbxHauteur.Text));
141         MonLogo.SpriteChoisi.Size = nouvelleTaille;
142         Invalidate();
143     }
144
145     /// <summary>
146     /// Applique les modifications de taille lorsque l'on quitte
147     /// la textbox au sprite choisi
148     /// </summary>
149     /// <param name="sender"></param>
150     /// <param name="e"></param>
151     private void tbxLargeur_Leave(object sender, EventArgs e)
152     {
153         Size nouvelleTaille = new ←
154             Size(Convert.ToInt32(tbxLargeur.Text), ←
155                 Convert.ToInt32(tbxHauteur.Text));
156         MonLogo.SpriteChoisi.Size = nouvelleTaille;
157         Invalidate();
158     }
159
160     /// <summary>
161     /// Appelle Dessine() et change de calque le sprite en question
162     /// </summary>
163     /// <param name="sender"></param>
164     /// <param name="e"></param>
165     private void cmbCalque_SelectedIndexChanged(object sender, ←
166         EventArgs e)
167     {
168         ComboBox cmb = (sender as ComboBox);
169         MonLogo.SpriteChoisi.NumeroCalque = ←
170             Convert.ToInt32(cmb.SelectedItem.ToString().Substring(cmb.SelectedItem.
171                 - 1)); // Ici, utilisation d'un sender --> je prévois le ←
172             texte qui aura une autre cmb mais même event
173         Dessine();
174     }
175
176     /// <summary>
177     /// Change la profondeur par calque du sprite choisi en fonction
178     /// de la valeur du numericUpDown
179     /// </summary>
180     /// <param name="sender"></param>
181     /// <param name="e"></param>
182     private void nudProfondeur_ValueChanged(object sender, ←
183         EventArgs e)
184     {
185         NumericUpDown nudProf = (sender as NumericUpDown);
186         MonLogo.SpriteChoisi.Profondeur = (int)nudProf.Value;
187         MonLogo.TrierSprites();
188         Dessine();
189         Invalidate();
190     }
191
192     /// <summary>
193     /// Définit si le sprite sera rempli ou non.
194     /// Active/Désactive le champ nudEpaisseur selon le choix.
195     /// </summary>
196     /// <param name="sender"></param>
197     /// <param name="e"></param>
198     private void chkRemplir_CheckedChanged(object sender, EventArgs e)
199     {
200         if (MonLogo.SpriteChoisi != null)
201         {
202             if (chkRemplir.Checked)
203             {
204                 MonLogo.SpriteChoisi.Remplir = true;
205                 nudEpaisseur.Enabled = false;
206             }
207             else
208             {
209                 MonLogo.SpriteChoisi.Remplir = false;

```

```

201         nudEpaisseur.Enabled = true;
202     }
203 }
204 Invalidate();
205 }
206
207 /// <summary>
208 /// Ouvre le colorDialog et applique la couleur choisie
209 /// </summary>
210 /// <param name="sender"></param>
211 /// <param name="e"></param>
212 private void btnCouleur_Click(object sender, EventArgs e)
213 {
214     colorDialog.ShowDialog();
215     btnCouleur.BackColor = colorDialog.Color;
216     MonLogo.SpriteChoisi.Couleur = colorDialog.Color;
217     Invalidate();
218 }
219
220 /// <summary>
221 /// Applique les modifications de position lorsque l'on quitte
222 /// la textbox au sprite choisi
223 /// </summary>
224 /// <param name="sender"></param>
225 /// <param name="e"></param>
226 private void tbxPos_Leave(object sender, EventArgs e)
227 {
228     Point nouvelleLocation = new ←
229         Point(Convert.ToInt32(tbxPosX.Text), ←
230             Convert.ToInt32(tbxPosY.Text));
231     MonLogo.SpriteChoisi.Location = nouvelleLocation;
232     Invalidate();
233 }
234
235 /// <summary>
236 /// Applique les modifications de position lorsque l'on quitte
237 /// la textbox au sprite choisi
238 /// </summary>
239 /// <param name="sender"></param>
240 /// <param name="e"></param>
241 private void tbxPosTexte_Leave(object sender, EventArgs e)
242 {
243     Point nouvelleLocation = new ←
244         Point(Convert.ToInt32(tbxPosXTexte.Text), ←
245             Convert.ToInt32(tbxPosYTexte.Text));
246     MonLogo.SpriteChoisi.Location = nouvelleLocation;
247     Invalidate();
248 }
249
250 /// <summary>
251 /// Applique le changement d'épaisseur au sprite choisi
252 /// </summary>
253 /// <param name="sender"></param>
254 /// <param name="e"></param>
255 private void nudEpaisseur_ValueChanged(object sender, EventArgs e)
256 {
257     MonLogo.SpriteChoisi.EpaisseurPen = ←
258         Convert.ToInt32(nudEpaisseur.Value);
259     Invalidate();
260 }
261
262 /// <summary>
263 /// Supprime le sprite actuellement sélectionné
264 /// </summary>
265 /// <param name="sender"></param>
266 /// <param name="e"></param>
267 private void btnSupprimerSprite_Click(object sender, EventArgs e)
268 {
269     //Suppression
270     Paint -= MonLogo.SpriteChoisi.SpritePaint;
271     Controls.Remove(MonLogo.SpriteChoisi);
272     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 transparence 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 ←
343                     NumericUpDown).Value);
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, ←
355         EventArgs e)
356     {
357         foreach (Sprite unSprite in MonLogo.Sprites.ListeDeSprite)
358         {
359             if (unSprite.NumeroCalque == CalqueChoisi)
360             {
361                 nudTransparenceCalque.Value = unSprite.AlphaCouleur;
362                 break;
363             }
364         }
365
366         /// <summary>
367         /// Ouvre une form montrant le logo qui sera exporté.
368         /// Lorsque l'utilisateur confirme sa volonté de sauve
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 ←
379                     créer votre logo avant d'exporter!");
380             }
381             else
382             {
383                 frmExporterLogo frmExporter = new ←
384                     frmExporterLogo(MonLogo);
385
386                 if (frmExporter.ShowDialog() == DialogResult.OK)
387                 {
388                     saveFileDialog = new SaveFileDialog();
389                     saveFileDialog.Filter = ←
390                         "BMP|*.bmp|GIF|*.gif|JPG|*.jpg;*.jpeg|PNG|*.png|TIFF|*.tif;*
391                             + "All Graphics|←
392                                 Types|*.bmp;*.jpg;*.jpeg;*.png;*.tif
393                     saveFileDialog.FileName = "Logo";
394
395                     if (saveFileDialog.ShowDialog() == DialogResult.OK)
396                     {
397                         frmExporter.LogoFinal.Save(saveFileDialog.FileName);
398                     }
399                 }
400             }
401         }
402
403         /// <summary>
404         /// Ouvre un openFileDialog pour que l'utilisateur choisisse
405         /// quel fichier il voudrait charger.
406         /// </summary>
407         /// <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 ←
418                     MonLogo.Sprites.ListeDeSprite)
419                 {
420                     unSprite.Click += this.ProprietesClick;
421                     Paint += unSprite.SpritePaint;
422                     this.Controls.Add(unSprite);
423                 }
424                 Invalidate();
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, ←
464         EventArgs e)
465     {
466         System.Diagnostics.Process.Start("https://github.com/walterjch/LogoGo/tr
467
468     /// <summary>
469     /// À la fermeture de frmLogo, affiche un messagebox. Ferme ←
470     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, ↵
                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             }
482             else
483             {
484                 e.Cancel = true;
485             }
486         }
487     }
488 }
489 #endregion
490
491 #region Méthodes
492     /// <summary>
493     /// Gère la création d'un nouveau sprite s'il est spécifié.
494     /// Efface et affiche à nouveau les sprites dans le bon ordre
495     /// </summary>
496     /// <param name="s"></param>
497     public void Dessine(Sprite s = null)
498     {
499         if (s != null)
500         {
501             MonLogo.AjouterSprite(s);
502             s.Click += this.ProprietesClick;
503         }
504
505         MonLogo.TrierSprites();
506
507         foreach (Sprite unSprite in MonLogo.Sprites.ListeDeSprite)
508         {
509             Paint -= unSprite.SpritePaint;
510             this.Controls.Remove(unSprite);
511             Paint += unSprite.SpritePaint;
512             this.Controls.Add(unSprite);
513         }
514     }
515 }
516
517
518     /// <summary>
519     /// Affiche les propriétés du sprite actuellement sélectionné
520     /// </summary>
521     public void AfficherProprietes()
522     {
523
524         if (MonLogo.SpriteChoisi is Texte)
525         {
526             pnlProprietesStandard.Visible = false;
527             pnlProprietesTexte.Visible = true;
528
529             //Affichage de la position du Sprite en question
530             tbxPosXTexte.Enabled = true;
531             tbxPosYTexte.Enabled = true;
532             tbxPosXTexte.Text = ↵
                MonLogo.SpriteChoisi.Location.X.ToString();

```

```

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

```

```

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

```

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

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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16  Fichier    : frmExporterLogo.cs
17
18  */
19 using System;
20 using System.Drawing;
21 using System.Windows.Forms;
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, ↵
44                               pbxResultatFinal.Height);
45         }
46
47         private void ExporterLogo_Load(object sender, EventArgs e)
48         {
49             bool premierSprite = true;
50             Graphics g = Graphics.FromImage(dessin);
51             foreach (Sprite unSprite in ↵
52                     LogoAExporter.Sprites.ListeDeSprite)
53             {
54                 //Repérage des extrémités du logo//
55                 //Pour le premier sprite, les coins du logo équivalent ↵
56                 //aux coins du sprite
57                 if (premierSprite)
58                 {
59                     CoinSuperieurGauche = unSprite.Location;
60                     CoinSuperieurDroit = new Point(unSprite.Location.X ↵
61                                                     + unSprite.Width, ↵
62                                                     +unSprite.Location.Y);
63                     CoinInferieurDroit = new Point(unSprite.Location.X ↵
64                                                     + unSprite.Width, ↵
65                                                     +unSprite.Location.Y + ↵
66                                                     unSprite.Height);
67                     CoinInferieurGauche = new ↵
68                                             Point(unSprite.Location.X, ↵
69                                                     +unSprite.Location.Y ↵
70                                                     + unSprite.Height);
71                     premierSprite = false;
72                 }
73             }
74         }
75     }
76 }
```

```

61     }
62     else
63     {
64         //Vérifie si le coin supérieur gauche doit changer
65         if (unSprite.Location.X < CoinSuperieurGauche.X)
66         {
67             CoinSuperieurGauche = new ←
68                 Point(unSprite.Location.X, ←
69                     CoinSuperieurGauche.Y);
70         }
71         if (unSprite.Location.Y < CoinSuperieurGauche.Y)
72         {
73             CoinSuperieurGauche = new ←
74                 Point(CoinSuperieurGauche.X, ←
75                     unSprite.Location.Y);
76         }
77         //Vérifie si le coin supérieur droit doit changer
78         if (unSprite.Location.X + unSprite.Width > ←
79             CoinSuperieurDroit.X)
80         {
81             CoinSuperieurDroit = new ←
82                 Point(unSprite.Location.X + unSprite.Width, ←
83                     CoinSuperieurDroit.Y);
84         }
85         if (unSprite.Location.Y < CoinSuperieurDroit.Y)
86         {
87             CoinSuperieurDroit = new ←
88                 Point(CoinSuperieurDroit.X, ←
89                     unSprite.Location.Y - (unSprite.Location.Y - ←
90                         CoinSuperieurGauche.Y));
91         }
92         //Vérifie si le coin inférieur droit doit changer
93         if (unSprite.Location.X + unSprite.Width > ←
94             CoinInferieurDroit.X)
95         {
96             CoinInferieurDroit = new ←
97                 Point(unSprite.Location.X + unSprite.Width, ←
98                     CoinInferieurDroit.Y);
99         }
100         if (unSprite.Location.Y + unSprite.Height > ←
101             CoinInferieurDroit.Y)
102         {
103             CoinInferieurDroit = new ←
104                 Point(CoinInferieurDroit.X, ←
105                     unSprite.Location.Y + unSprite.Height);
106         }
107         //Vérifie si le coin inférieur gauche doit changer
108         if (unSprite.Location.X < CoinInferieurGauche.X)
109         {
110             CoinInferieurGauche = new ←
111                 Point(unSprite.Location.X, ←
112                     CoinInferieurGauche.Y);
113         }
114         if (unSprite.Location.Y + unSprite.Height > ←
115             CoinInferieurGauche.Y)
116         {
117             CoinInferieurGauche = new ←
118                 Point(CoinInferieurGauche.X, ←
119                     unSprite.Location.Y + unSprite.Height);
120         }
121     }
122     // On dessine le sprite
123     unSprite.SpritePaintAvecGraphics(g);
124 }
125 //On resize la picturebox pour qu'elle fasse la même ←
126 dimension que le logo
127
128 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 : ←
116     https://stackoverflow.com/questions/13217156/cropping-picture-getting-re
117     /// <summary>
118     /// Rogne une image donnée en une zone qu'on doit spécifier.
119     /// </summary>
120     /// <param name="original_image">Image que l'on veut ←
121         rogner</param>
122     /// <param name="x">Position X de là où on veut rogner ←
123         l'image</param>
124     /// <param name="y">Position Y de là où on veut rogner ←
125         l'image</param>
126     /// <param name="width">Largeur voulue</param>
127     /// <param name="height">Hauteur voulue</param>
128     /// <returns>Retourne l'image finale</returns>
129     public Bitmap RedimensionnerLogo(Bitmap original_image, int x, ←
        int y, int width, int height)
130     {
131         Rectangle crop = new Rectangle(x - BORDURE_OFFSET, y - ←
            BORDURE_OFFSET, width + 2 * BORDURE_OFFSET, height + 2 * ←
            BORDURE_OFFSET);
132
133         var bmp = new Bitmap(crop.Width, crop.Height);
134         using (var gr = Graphics.FromImage(bmp))
135         {
136             gr.DrawImage(original_image, new Rectangle(0, 0, ←
                bmp.Width, bmp.Height), crop, GraphicsUnit.Pixel);
137         }
138         return bmp;
139     }
140 }

```

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 modifier ces formes et il dispose de ↵
12               calques.
13
14               Il est possible d'exporter, enregistrer, et ouvrir un logo.
15
16 Fichier     : frmCreerPolygone.cs
17
18 */
19 using System;
20 using System.Collections.Generic;
21 using System.Drawing;
22 using System.Windows.Forms;
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 = ↵
32             value; }
33
34         public frmCreerPolygone(Polygone p)
35         {
36             InitializeComponent();
37             _pointsATracer = new List<PointF>();
38             NouveauPolygone = p;
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, ↵
46                 MouseEventArgs e)
47             {
48                 btnOK.Enabled = true;
49                 _pointsATracer.Add(new PointF(e.X, e.Y));
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 - ↵
63                             _pointsATracer[0].X, unPointF.Y - _pointsATracer[0].Y);
64                     }
65                     this.DialogResult = System.Windows.Forms.DialogResult.OK;
66                     Close();
67                 }
68             }
69         }
70     }
71 }
```

---

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

Listing 13 – ./files/frmCreerPolygone.cs