```
1  /*
2   * Author  : Dubas Loïc
3   * Class   : I.FA-P3B
4   * School  : CFPT-I
5   * Date    : June 2018
6   * Descr.  : show user's hand and modele's hand
7   * Version : 1.0
8   * Ext. dll: LeapCSharp.NET4.5
9   */
10
11 using System;
12 using System.Collections.Generic;
13 using System.ComponentModel;
14 using System.Data;
15 using System.Drawing;
16 using System.Linq;
17 using System.Text;
18 using System.Threading.Tasks;
19 using System.Windows.Forms;
20 // References to add
21 using Leap;
22 using System.IO;
23
24 namespace fingers_cloner
25 {
26     public partial class frmMain : Form
27     {
28         #region Initialization
29         // Initialize Leap Motion
30         LeapController leapController;
31
32         // Initialize Paint class to draw
33         Paint paint;
34
35         // Initialize Hand class to store hands position info
36         MyHand userHand;
37         MyHand modeleHand;
38         List<Color> userFingersColor;
39
40         // Precision setted by trackbar
41         int precision;
42         List<Vector> handsDiff;
43         List<double> fingersDist;
44
45         // serialize/deserialize saved positions
46         Serialization savedPositions;
47         List<MyHand> allPositions;
48         #endregion
49
50         /// <summary>
51         /// default constructor
52         /// </summary>
53         public frmMain()
54         {
55             InitializeComponent();
56
```

```
57              // create the serial folder if not exist to store saved positions
58              Directory.CreateDirectory("serial");
59
60              DoubleBuffered = true;
61
62              // initialize the leap controller
63              leapController = new LeapController();
64
65              // initialize serialization class
66              savedPositions = new Serialization();
67              // initialize paint class
68              paint = new Paint();
69              // send panel dimensions to paint class
70              paint.GetPanelSize(pnlUserHand.Width, pnlUserHand.Height);
71
72              updateCombobox();
73              updateModele();
74
75              // get value of trackbar
76              precision = trackBar1.Value;
77          }
78
79          /// <summary>
80          /// Refresh panel on each tick
81          /// </summary>
82          /// <param name="sender"></param>
83          /// <param name="e"></param>
84          private void timer1_Tick(object sender, EventArgs e)
85          {
86              userHand = leapController.UserHand;
87              pnlUserHand.Invalidate();
88          }
89
90          /// <summary>
91          /// Draw the user's hand
92          /// </summary>
93          /// <param name="sender"></param>
94          /// <param name="e"></param>
95          private void pnlUserHand_Paint(object sender, PaintEventArgs e)
96          {
97              try
98              {
99                  // if combobox isn't empty, compare current modele with user's ↵
                       hand
100                 if (cbxModele.Items.Count > 0)
101                 {
102                     comparePosition();
103                     userFingersColor = colorIndicator();
104                     paint.paintHandColor(e, userHand, userFingersColor);
105
106                     ControlPaint.DrawBorder(e.Graphics,                    ↵
                        this.pnlUserHand.ClientRectangle, panelColor           ↵
                        (userFingersColor), ButtonBorderStyle.Solid);
107                 }
108                 else
109                 {
```

```
110                     paint.paintHand(e, userHand);
111                 }
112
113                 lblUserHand.Text = "Votre main :";
114                 btnNewModel.Enabled = true;
115             }
116             catch (Exception)
117             {
118                 lblUserHand.Text = "Pas de main détectée !";
119                 btnNewModel.Enabled = false;
120             }
121         }
122
123         /// <summary>
124         /// draw modele's hand
125         /// </summary>
126         /// <param name="sender"></param>
127         /// <param name="e"></param>
128         private void pnlModelHand_Paint(object sender, PaintEventArgs e)
129         {
130             // if combobox isn't empty, show selected modele's description and ↵
                position
131             if (cbxModele.Items.Count > 0)
132             {
133                 paint.paintHand(e, modeleHand);
134                 btnEdit.Enabled = true;
135                 btnDelete.Enabled = true;
136             }
137         }
138
139         private void cbxModele_SelectedIndexChanged(object sender, EventArgs   ↵
              e)
140         {
141             updateModele();
142         }
143
144         /// <summary>
145         /// Open a new form to create a new modele
146         /// </summary>
147         /// <param name="sender"></param>
148         /// <param name="e"></param>
149         private void btnNewModel_Click(object sender, EventArgs e)
150         {
151             frmNewModele newModele = new frmNewModele(userHand);
152
153             newModele.getAllPositions(allPositions);
154             newModele.ShowDialog();
155
156             if (newModele.DialogResult == DialogResult.OK)
157             {
158                 updateCombobox();
159             }
160         }
161
162         /// <summary>
163         /// Choose the precision required to accept a position
```

```
164         /// </summary>
165         /// <param name="sender"></param>
166         /// <param name="e"></param>
167         private void trackBar1_ValueChanged(object sender, EventArgs e)
168         {
169             lblPercentage.Text = Convert.ToString(trackBar1.Value) + "%";
170             precision = trackBar1.Value;
171         }
172 
173         /// <summary>
174         /// send to paint class new dimensions of window and refresh panel of
              modele
175         /// </summary>
176         /// <param name="sender"></param>
177         /// <param name="e"></param>
178         private void frmMain_SizeChanged(object sender, EventArgs e)
179         {
180             paint.GetPanelSize(pnlUserHand.Width, pnlUserHand.Height);
181             pnlModelHand.Invalidate();
182         }
183 
184         #region edition
185         /// <summary>
186         /// open edit window
187         /// </summary>
188         /// <param name="sender"></param>
189         /// <param name="e"></param>
190         private void btnEdit_Click(object sender, EventArgs e)
191         {
192             frmEdit edit = new frmEdit(modeleHand);
193 
194             edit.ShowDialog();
195 
196             if (edit.DialogResult == DialogResult.OK)
197             {
198                 updateCombobox();
199             }
200         }
201 
202         /// <summary>
203         /// delete current modele
204         /// </summary>
205         /// <param name="sender"></param>
206         /// <param name="e"></param>
207         private void btnDelete_Click(object sender, EventArgs e)
208         {
209             DialogResult delete = MessageBox.Show("Êtes-vous sûr de vouloir
                supprimer la position " + modeleHand.Name + " ?", "Supprimer une
                 position", MessageBoxButtons.YesNo);
210 
211             if (delete == DialogResult.Yes)
212             {
213                 savedPositions.deletePosition(modeleHand.Name);
214                 updateCombobox();
215                 updateModele();
216                 pnlModelHand.Invalidate();
```

```
217             }
218         }
219         #endregion
220
221         #region functions
222         /// <summary>
223         /// Update combobox with the latest saved positions
224         /// </summary>
225         private void updateCombobox()
226         {
227             // get all saved position
228             allPositions = savedPositions.deserialize();
229
230             // add all position to combobox
231             cbxModele.DataSource = allPositions;
232             cbxModele.DisplayMember = "Name";
233
234             // if combobox isn't empty, select first of the list
235             if (cbxModele.Items.Count >= 1)
236             {
237                 cbxModele.SelectedIndex = 0;
238                 cbxModele.Enabled = true;
239                 updateModele();
240             }
241             else
242             {
243                 // if combobox is empty, disable combobox and edition buttons
244                 cbxModele.Enabled = false;
245                 btnEdit.Enabled = false;
246                 btnDelete.Enabled = false;
247             }
248         }
249
250         /// <summary>
251         /// Met à jour le modèle sélectionné, affiche son nom, sa description ↵
              et rafraîchit le panel
252         /// </summary>
253         private void updateModele()
254         {
255             // set modele's hand to the selected modele
256             modeleHand = (MyHand)cbxModele.SelectedItem;
257
258             // if there is modele hand saved
259             if (modeleHand != null)
260             {
261                 // show name, description and picture
262                 lblName.Text = modeleHand.Name;
263                 lblDescription.Text = modeleHand.Description;
264                 if (modeleHand.Image != null)
265                 {
266                     pbxModele.Image = stringToImage(modeleHand.Image);
267                 }
268                 else
269                 {
270                     pbxModele.Image = Properties.Resources.no_image_available;
271                 }
```

```
272             }
273             else
274             {
275                 lblName.Text = "Aucun modèle";
276                 lblDescription.Text = "Aucun modèle n'est chargé. Créez-en ou
                      sélectionnez-en un !";
277                 pbxModele.Image = Properties.Resources.no_image_available;
278             }
279 
280             lblName.Visible = true;
281             lblDescription.Visible = true;
282 
283             pnlModelHand.Invalidate();
284         }
285 
286         /// <summary>
287         /// Calculate distance between each fingers of user's and modele's
              hand
288         /// </summary>
289         /// <returns>A list of distances between each fingers</returns>
290         private List<double> comparePosition()
291         {
292             handsDiff = new List<Vector>();
293             fingersDist = new List<double>();
294             List<Vector> modelePanelPos = paint.normToPalmPanelModelePos
                  (modeleHand);
295 
296             for (int i = 0; i < paint.FingersPanelPos.Count; i++)
297             {
298                 handsDiff.Add(paint.FingersPanelPos[i] - modelePanelPos[i]);
299                 fingersDist.Add(Math.Sqrt(
300                     (Math.Pow(handsDiff[i].x, 2)) + (Math.Pow(handsDiff[i].z,
                      2))
301                     ));
302             }
303 
304             return fingersDist;
305         }
306 
307         /// <summary>
308         /// List of color for each fingers to show how close user's hand is to
                modele
309         /// </summary>
310         /// <returns>List of the colors</returns>
311         private List<Color> colorIndicator()
312         {
313             List<Color> color = new List<Color>();
314             int tolerance = (pnlUserHand.Width / 4) - this.precision;
315 
316             for (int i = 0; i < fingersDist.Count; i++)
317             {
318                 if (fingersDist[i] < tolerance)
319                 {
320                     color.Add(Color.Green);
321                 }
322                 else if (fingersDist[i] < (tolerance + 10))
```

```
323                 {
324                     color.Add(Color.Orange);
325                 }
326                 else if (fingersDist[i] < (tolerance + 30))
327                 {
328                     color.Add(Color.Red);
329                 }
330                 else
331                 {
332                     color.Add(Color.Black);
333                 }
334             }
335 
336             return color;
337         }
338 
339         /// <summary>
340         /// set the panel border's color depending on average of user's fingers position
341         /// </summary>
342         /// <param name="fingersColor"></param>
343         /// <returns></returns>
344         private Color panelColor(List<Color> fingersColor)
345         {
346             Color panelColor = new Color();
347             int totalColorValue = 0;
348             int averageColorValue;
349 
350             for (int i = 0; i < fingersColor.Count; i++)
351             {
352                 if (fingersColor[i] == Color.Green)
353                 {
354                     totalColorValue += 3;
355                 }
356                 else if (fingersColor[i] == Color.Orange)
357                 {
358                     totalColorValue += 2;
359                 }
360                 else if (fingersColor[i] == Color.Red)
361                 {
362                     totalColorValue += 1;
363                 }
364                 else
365                 {
366                     totalColorValue += 0;
367                 }
368             }
369 
370             averageColorValue = totalColorValue / 5;
371 
372             if (averageColorValue == 3)
373             {
374                 panelColor = Color.Green;
375             }
376             else if (averageColorValue >= 2)
377             {
```

```
378                 panelColor = Color.Orange;
379             }
380             else if (averageColorValue >= 1)
381             {
382                 panelColor = Color.Red;
383             }
384             else if (averageColorValue == 0)
385             {
386                 panelColor = Color.Black;
387             }
388 
389             return panelColor;
390         }
391 
392         /// <summary>
393         /// transform a text as an image
394         /// </summary>
395         /// <param name="stringImage"></param>
396         /// <returns></returns>
397         private System.Drawing.Image stringToImage(string stringImage)
398         {
399             System.Drawing.Image image;
400 
401             Byte[] stringAsByte = Convert.FromBase64String(stringImage);
402             MemoryStream memstr = new MemoryStream(stringAsByte);
403 
404             image = System.Drawing.Image.FromStream(memstr);
405 
406             return image;
407         }
408         #endregion
409     }
410 }
411 
```

```
1  /*
2   * Author  : Dubas Loïc
3   * Class   : I.FA-P3B
4   * School  : CFPT-I
5   * Date    : June 2018
6   * Descr.  : Create a new modele, with a name, a description and a picture
7   * Version : 1.0
8   * Ext. dll: LeapCSharp.NET4.5
9   */
10
11 using System;
12 using System.Collections.Generic;
13 using System.ComponentModel;
14 using System.Data;
15 using System.Drawing;
16 using System.Linq;
17 using System.Text;
18 using System.Threading.Tasks;
19 using System.Windows.Forms;
20 // References to add
21 using Leap;
22 using System.Diagnostics;
23 using System.Xml;
24 using System.Xml.Serialization;
25 using System.IO;
26
27 namespace fingers_cloner
28 {
29     public partial class frmNewModele : Form
30     {
31         #region Initialization
32         // initialize Leap Motion
33         LeapController leapController;
34
35         // initialize Paint functions
36         Paint paint;
37
38         // current position
39         MyHand currentPosition;
40
41         // initialize serialization functions
42         Serialization serialization;
43
44         // name, description and picture of the modele
45         string name;
46         string description;
47         Bitmap loadedPicture;
48         string imageAsString;
49
50         List<MyHand> allPositions;
51         #endregion
52
53         /// <summary>
54         /// create new modele form
55         /// </summary>
56         /// <param name="fingersNormPos">finger's normalized position</param>
```

```
57          /// <param name="palmNormPos">palm's normalized position</param>
58          public frmNewModele(MyHand handToSave)
59          {
60              InitializeComponent();
61              DoubleBuffered = true;
62
63              leapController = new LeapController();
64              paint = new Paint();
65              paint.GetPanelSize(pnlModele.Width, pnlModele.Height);
66              serialization = new Serialization();
67
68              this.currentPosition = handToSave;
69          }
70
71          /// <summary>
72          ///  draw hand if there is one
73          /// </summary>
74          /// <param name="sender"></param>
75          /// <param name="e"></param>
76          private void pnlModele_Paint(object sender, PaintEventArgs e)
77          {
78              try
79              {
80                  paint.paintHand(e, currentPosition);
81              }
82              catch (Exception)
83              {
84                  NoHandDetected();
85              }
86          }
87
88          /// <summary>
89          /// enable save button if there is a name to it and if the name is not ↵
                already taken
90          /// </summary>
91          /// <param name="sender"></param>
92          /// <param name="e"></param>
93          private void tbxModeleName_TextChanged(object sender, EventArgs e)
94          {
95              if (tbxModeleName.Text.Length <= 0)
96              {
97                  btnSave.Enabled = false;
98              }
99              else if (checkName())
100             {
101                 btnSave.Enabled = false;
102             }
103             else
104             {
105                 btnSave.Enabled = true;
106             }
107         }
108
109         /// <summary>
110         /// open file dialog to choose image
111         /// </summary>
```

```
112         /// <param name="sender"></param>
113         /// <param name="e"></param>
114         private void btnLoadImage_Click(object sender, EventArgs e)
115         {
116             OpenFileDialog ofd = new OpenFileDialog();
117 
118             ofd.InitialDirectory = "C:\\Users";
119             ofd.Filter = "Image files (*.png, *.jpg, *.jpeg, *.gif, *.bmp)|
                *.png;*.jpg;*.jpeg;*.gif;*.bmp";
120 
121             if (ofd.ShowDialog() == DialogResult.OK)
122             {
123                 loadedPicture = new Bitmap(ofd.FileName);
124                 lblFileName.Text = ofd.SafeFileName;
125                 lblFileName.Visible = true;
126                 TypeConverter converter = TypeDescriptor.GetConverter(typeof
                    (Bitmap));
127                 imageAsString = Convert.ToBase64String((Byte[])
                    converter.ConvertTo(loadedPicture, typeof(Byte[])));
128             }
129         }
130 
131         /// <summary>
132         /// modify position to save and serialize it
133         /// </summary>
134         /// <param name="sender"></param>
135         /// <param name="e"></param>
136         private void btnSave_Click(object sender, EventArgs e)
137         {
138             name = tbxModeleName.Text;
139 
140             // Open a new form to add a description
141             frmComment comment = new frmComment();
142             comment.ShowDialog();
143 
144             // when click on 'OK' on the comment form
145             if (comment.DialogResult == DialogResult.OK)
146             {
147                 // add description and name to position to save
148                 description = comment.Description;
149                 currentPosition.Description = description;
150                 currentPosition.Name = name;
151                 if (loadedPicture != null)
152                 {
153                     currentPosition.Image = imageAsString;
154                 }
155 
156                 // serialize the savedHand object
157                 serialization.serialize(currentPosition);
158 
159                 // Close comment and newModele form
160                 this.Close();
161             }
162         }
163 
164         /// <summary>
```

```
165         /// if there is no hand detected by the Leap, user is informed and
              send back to main form
166         /// </summary>
167         private void NoHandDetected()
168         {
169             MessageBox.Show("Aucune main détectée. Veuillez réessayer.");
170             this.Close();
171         }
172 
173         /// <summary>
174         /// get all saved positions
175         /// </summary>
176         /// <param name="allPositions">saved positions</param>
177         public void getAllPositions(List<MyHand> allPositions)
178         {
179             this.allPositions = allPositions;
180         }
181 
182         /// <summary>
183         /// check if the name is already taken
184         /// </summary>
185         /// <returns></returns>
186         private bool checkName()
187         {
188             bool nameTaken = false;
189 
190             if (allPositions != null)
191             {
192                 for (int i = 0; i < allPositions.Count; i++)
193                 {
194                     if (allPositions[i].Name == tbxModeleName.Text)
195                     {
196                         nameTaken = true;
197                         break;
198                     }
199                 }
200             }
201 
202             return nameTaken;
203         }
204     }
205 }
206 
```

```
1  /*
2   * Author  : Dubas Loïc
3   * Class   : I.FA-P3B
4   * School  : CFPT-I
5   * Date    : June 2018
6   * Descr.  : Set the description of the position to save
7   * Version : 1.0
8   * Ext. dll: LeapCSharp.NET4.5
9   */
10
11 using System;
12 using System.Collections.Generic;
13 using System.ComponentModel;
14 using System.Data;
15 using System.Drawing;
16 using System.Linq;
17 using System.Text;
18 using System.Threading.Tasks;
19 using System.Windows.Forms;
20
21 namespace fingers_cloner
22 {
23     public partial class frmComment : Form
24     {
25         #region initialization
26         // description of the position to save
27         private string _description;
28         public string Description { get => _description; set => _description =
             value; }
29         #endregion
30
31         /// <summary>
32         /// default constructor
33         /// </summary>
34         public frmComment()
35         {
36             InitializeComponent();
37         }
38
39         /// <summary>
40         /// save description text
41         /// </summary>
42         /// <param name="sender"></param>
43         /// <param name="e"></param>
44         private void btnSave_Click(object sender, EventArgs e)
45         {
46             this.Description = tbxDescription.Text;
47         }
48
49         /// <summary>
50         /// Disable the save button if description is empty
51         /// </summary>
52         /// <param name="sender"></param>
53         /// <param name="e"></param>
54         private void tbxDescription_TextChanged(object sender, EventArgs e)
55         {
```

```
56              if (tbxDescription.Text.Length <= 0)
57              {
58                  btnSave.Enabled = false;
59              }
60              else
61              {
62                  btnSave.Enabled = true;
63              }
64          }
65      }
66  }
67
```

```
1  /*
2   * Author  : Dubas Loïc
3   * Class   : I.FA-P3B
4   * School  : CFPT-I
5   * Date    : June 2018
6   * Descr.  : edit the current loaded position
7   * Version : 1.0
8   * Ext. dll: LeapCSharp.NET4.5
9   */
10
11 using System;
12 using System.Collections.Generic;
13 using System.ComponentModel;
14 using System.Data;
15 using System.Drawing;
16 using System.Linq;
17 using System.Text;
18 using System.Threading.Tasks;
19 using System.Windows.Forms;
20
21 namespace fingers_cloner
22 {
23     public partial class frmEdit : Form
24     {
25         #region initialization
26         // name and picture of the modele
27         string nameHandToEdit;
28         string imageHandToEdit;
29         // the hand to edit and the updated picture
30         MyHand handToEdit;
31         Bitmap loadedPicture;
32         string imageAsString;
33
34         // Initialize serialization functions
35         Serialization serialization;
36         #endregion
37
38         /// <summary>
39         /// default constructor
40         /// </summary>
41         /// <param name="modelHand">the position to edit</param>
42         public frmEdit(MyHand modelHand)
43         {
44             InitializeComponent();
45
46             handToEdit = modelHand;
47             nameHandToEdit = modelHand.Name;
48             imageHandToEdit = modelHand.Image;
49
50             serialization = new Serialization();
51
52             tbxName.Text = modelHand.Name;
53             tbxDescription.Text = modelHand.Description;
54         }
55
56         /// <summary>
```

```
57          /// validation is possible only if the textbox of the name isn't empty
58          /// </summary>
59          /// <param name="sender"></param>
60          /// <param name="e"></param>
61          private void tbxName_TextChanged(object sender, EventArgs e)
62          {
63              if (tbxName.Text.Length <= 0)
64              {
65                  btnValidate.Enabled = false;
66              }
67              else
68              {
69                  btnValidate.Enabled = true;
70              }
71          }
72
73          /// <summary>
74          /// validation is possible only if the textbox of the description
              isn't empty
75          /// </summary>
76          /// <param name="sender"></param>
77          /// <param name="e"></param>
78          private void tbxDescription_TextChanged(object sender, EventArgs e)
79          {
80              if (tbxDescription.Text.Length <= 0)
81              {
82                  btnValidate.Enabled = false;
83              }
84              else
85              {
86                  btnValidate.Enabled = true;
87              }
88          }
89
90          /// <summary>
91          /// open file dialog choose a picture and transform it in string
92          /// </summary>
93          /// <param name="sender"></param>
94          /// <param name="e"></param>
95          private void btnImage_Click(object sender, EventArgs e)
96          {
97              OpenFileDialog ofd = new OpenFileDialog();
98
99              ofd.InitialDirectory = "C:\\Users";
100             ofd.Filter = "Image files (*.png, *.jpg, *.jpeg, *.gif, *.bmp)|
                  *.png;*.jpg;*.jpeg;*.gif;*.bmp";
101
102             if (ofd.ShowDialog() == DialogResult.OK)
103             {
104                 loadedPicture = new Bitmap(ofd.FileName);
105                 lblFileName.Text = ofd.SafeFileName;
106                 lblFileName.Visible = true;
107                 TypeConverter converter = TypeDescriptor.GetConverter(typeof
                      (Bitmap));
108                 imageAsString = Convert.ToBase64String((Byte[])
                      converter.ConvertTo(loadedPicture, typeof(Byte[])));
```

```
109             }
110         }
111 
112         /// <summary>
113         /// edit the hand
114         /// </summary>
115         /// <param name="sender"></param>
116         /// <param name="e"></param>
117         private void btnValidate_Click(object sender, EventArgs e)
118         {
119             handToEdit.Name = tbxName.Text;
120             handToEdit.Description = tbxDescription.Text;
121             if (loadedPicture == null)
122             {
123                 imageAsString = imageHandToEdit;
124             }
125             else
126             {
127                 handToEdit.Image = imageAsString;
128             }
129 
130             serialization.deletePosition(nameHandToEdit);
131             serialization.serialize(handToEdit);
132         }
133     }
134 }
135 
```

```
1  /*
2   * Author  : Dubas Loïc
3   * Class   : I.FA-P3B
4   * School  : CFPT-I
5   * Date    : June 2018
6   * Descr.  : Detect hand and calculate finger's position
7   * Version : 1.0
8   * Ext. dll: LeapCSharp.NET4.5
9   */
10
11 using System;
12 using System.Collections.Generic;
13 using System.Linq;
14 using System.Text;
15 using System.Threading.Tasks;
16 using System.Windows.Forms;
17 // References to add
18 using Leap;
19 using System.Xml;
20
21 namespace fingers_cloner
22 {
23     class LeapController : Controller
24     {
25         #region Initialization
26         // set
27         // List of detected hands and the first detected hand
28         private List<Hand> _hands;
29         private Hand _firstHand;
30
31         // Palm raw, normalized and stabilized location
32         private Vector _palmPos;
33         private Vector _palmNormPos;
34
35         // List of all the detected fingers
36         private List<Finger> _fingers;
37
38         // Fingers raw and normalized location
39         private List<Vector> _fingersPos;
40         private List<Vector> _fingersNormPos;
41
42         // User's hand
43         private MyHand _userHand;
44
45         // get
46         public List<Hand> Hands { get => _hands; set => _hands = value; }
47         public Hand FirstHand { get => _firstHand; set => _firstHand = value; }
48         public List<Finger> Fingers { get => _fingers; set => _fingers =
             value; }
49         public List<Vector> FingersStabPos { get => _fingersPos; set =>
             _fingersPos = value; }
50         public List<Vector> FingersNormPos { get => _fingersNormPos; set =>
             _fingersNormPos = value; }
51         public Vector PalmPos { get => _palmPos; set => _palmPos = value; }
52         public Vector PalmNormPos { get => _palmNormPos; set => _palmNormPos =
             value; }
```

```
53          public MyHand UserHand { get => _userHand; set => _userHand = value; }
54          #endregion
55
56          /// <summary>
57          /// Leap Motion's default constructor
58          /// </summary>
59          public LeapController()
60          {
61              EventContext = WindowsFormsSynchronizationContext.Current;
62              FrameReady += newFrameHandler;
63          }
64
65          /// <summary>
66          /// Refresh the fingers info on every frame of the Leap Motion
67          /// </summary>
68          /// <param name="sender"></param>
69          /// <param name="eventArgs"></param>
70          public void newFrameHandler(object sender, FrameEventArgs eventArgs)
71          {
72              Frame frame = eventArgs.frame;
73              InteractionBox iBox = frame.InteractionBox;
74
75              if (frame.Hands.Count > 0)
76              {
77                  Hands = frame.Hands;
78                  FirstHand = Hands[0];
79
80                  PalmPos = FirstHand.PalmPosition;
81                  PalmNormPos = iBox.NormalizePoint(PalmPos);
82
83                  Fingers = FirstHand.Fingers;
84                  FingersStabPos = new List<Vector>();
85                  FingersNormPos = new List<Vector>();
86
87                  for (int i = 0; i < Fingers.Count; i++)
88                  {
89                      FingersStabPos.Add(Fingers[i].StabilizedTipPosition);
90                      FingersNormPos.Add(iBox.NormalizePoint(FingersStabPos[i]));
91                  }
92
93                  UserHand = new MyHand(PalmNormPos, FingersNormPos);
94              }
95          }
96      }
97  }
98
```

```
1  /*
2   * Author  : Dubas Loïc
3   * Class   : I.FA-P3B
4   * School  : CFPT-I
5   * Date    : June 2018
6   * Descr.  : Store hand data
7   * Version : 1.0
8   * Ext. dll: LeapCSharp.NET4.5
9   */
10
11 using System;
12 using System.Collections.Generic;
13 using System.Linq;
14 using System.Text;
15 using System.Threading.Tasks;
16 // References to add
17 using Leap;
18
19 namespace fingers_cloner
20 {
21     public class MyHand
22     {
23         #region Initialization
24         // get
25         private string _name;
26         private string _description;
27         private Vector _palmNormPos;
28         private List<Vector> _fingersNormPos;
29         private string _image;
30         // set
31         // name
32         public string Name { get => _name; set => _name = value; }
33         // description
34         public string Description { get => _description; set => _description =
             value; }
35         // normalized position of the palm
36         public Vector PalmNormPos { get => _palmNormPos; set => _palmNormPos =
             value; }
37         // normalized positions of the fingers
38         public List<Vector> FingersNormPos { get => _fingersNormPos; set =>
             _fingersNormPos = value; }
39         // image of the position as a string
40         public string Image { get => _image; set => _image = value; }
41         #endregion
42
43         /// <summary>
44         /// default constructor
45         /// </summary>
46         public MyHand() { }
47
48         /// <summary>
49         /// MyHand constructor
50         /// </summary>
51         /// <param name="palmPosNorm">Normalized position of the palm</param>
52         /// <param name="fingersPosNorm">Normalized positions of the fingers</
             param>
```

```
53          public MyHand(Vector palmPosNorm, List<Vector> fingersPosNorm)
54          {
55              this.PalmNormPos = palmPosNorm;
56              this.FingersNormPos = fingersPosNorm;
57          }
58
59          /// <summary>
60          /// MyHand constructor
61          /// </summary>
62          /// <param name="name">Name of the position</param>
63          /// <param name="description">Description of the position</param>
64          /// <param name="palmPosNorm">Normalized position of the palm</param>
65          /// <param name="fingersPosNorm">Normalized positions of the fingers</
              param>
66          public MyHand(string name, string description, Vector palmPosNorm,
              List<Vector> fingersPosNorm)
67          {
68              this.Name = name;
69              this.Description = description;
70              this.PalmNormPos = palmPosNorm;
71              this.FingersNormPos = fingersPosNorm;
72          }
73
74          /// <summary>
75          /// MyHand constructor
76          /// </summary>
77          /// <param name="name">Name of the position</param>
78          /// <param name="description">Description of the position</param>
79          /// <param name="palmPosNorm">Normalized position of the palm</param>
80          /// <param name="fingersPosNorm">Normalized positions of the fingers</
              param>
81          /// <param name="image">Image of the position as a string</param>
82          public MyHand(string name, string description, Vector palmPosNorm,
              List<Vector> fingersPosNorm, string image)
83          {
84              this.Name = name;
85              this.Description = description;
86              this.PalmNormPos = palmPosNorm;
87              this.FingersNormPos = fingersPosNorm;
88              this.Image = image;
89          }
90      }
91  }
92
```

```
1  /*
2   * Author  : Dubas Loïc
3   * Class   : I.FA-P3B
4   * School  : CFPT-I
5   * Date    : June 2018
6   * Descr.  : Drawing hand, circle and line functions
7   * Version : 1.0
8   * Ext. dll: LeapCSharp.NET4.5
9   */
10
11 using System;
12 using System.Collections.Generic;
13 using System.Linq;
14 using System.Text;
15 using System.Threading.Tasks;
16 // References to add
17 using System.Drawing;
18 using System.Windows.Forms;
19 using Leap;
20
21 namespace fingers_cloner
22 {
23     class Paint
24     {
25         #region Initialization
26         // Fixed circle size
27         const int CIRCLESIZE = 50;
28
29         // Palm fixed location in the panel
30         Vector palmPanelPos;
31
32         // Dimensions of the panel
33         private int _panelWidth;
34         private int _panelHeight;
35         public int PanelWidth { get => _panelWidth; set => _panelWidth =
             value; }
36         public int PanelHeight { get => _panelHeight; set => _panelHeight =
             value; }
37
38         // hand to draw
39         private MyHand _hand;
40         private List<Vector> _fingersPanelPos;
41         private List<Vector> _modelePanelPos;
42         public MyHand Hand { get => _hand; set => _hand = value; }
43         public List<Vector> FingersPanelPos { get => _fingersPanelPos; set =>
             _fingersPanelPos = value; }
44         public List<Vector> ModelePanelPos { get => _modelePanelPos; set =>
             _modelePanelPos = value; }
45         #endregion
46
47         /// <summary>
48         /// Paint constructor
49         /// </summary>
50         /// <param name="panelWidth">Panel width</param>
51         /// <param name="panelHeight">Panel height</param>
52         public Paint() { }
```

```
53
54          /// <summary>
55          /// get the panel size
56          /// </summary>
57          /// <param name="panelWidth">Panel width</param>
58          /// <param name="panelHeight">Panel height</param>
59          public void GetPanelSize(int panelWidth, int panelHeight)
60          {
61              this.PanelWidth = panelWidth;
62              this.PanelHeight = panelHeight;
63
64              palmPanelPos = new Vector((PanelWidth / 2), 0, (PanelHeight -
                  CIRCLESIZE));
65          }
66
67          #region drawing black
68          /// <summary>
69          /// Draw a hand
70          /// </summary>
71          /// <param name="e">paint event</param>
72          /// <param name="hand">hand to paint</param>
73          public void paintHand(PaintEventArgs e, MyHand hand)
74          {
75              this.Hand = hand;
76              FingersPanelPos = normToPalmPanelPos();
77
78              this.DrawEllipseRectangle(e, Convert.ToInt32(palmPanelPos.x),
                  Convert.ToInt32(palmPanelPos.z));
79              for (int i = 0; i < FingersPanelPos.Count; i++)
80              {
81                  this.DrawEllipseRectangle(e, Convert.ToInt32(FingersPanelPos
                      [i].x), Convert.ToInt32(FingersPanelPos[i].z));
82                  this.DrawLinePoint(e, Convert.ToInt32(FingersPanelPos[i].x),
                      Convert.ToInt32(FingersPanelPos[i].z));
83              }
84          }
85
86          /// <summary>
87          /// Draw a circle at a certain location
88          /// </summary>
89          /// <param name="e">Paint event</param>
90          /// <param name="x">Horizonzal coordinate of finger/palm</param>
91          /// <param name="z">Vertical coordinate of finger/palm</param>
92          private void DrawEllipseRectangle(PaintEventArgs e, int x, int z)
93          {
94              // Create pen.
95              Pen Pen = new Pen(Color.Black, 3);
96
97              // Create rectangle for ellipse.
98              Rectangle rect = new Rectangle(x - (CIRCLESIZE / 2), z -
                  (CIRCLESIZE / 2), CIRCLESIZE, CIRCLESIZE);
99
100             // Draw ellipse to screen.
101             e.Graphics.DrawEllipse(Pen, rect);
102         }
103
```

```
104         /// <summary>
105         /// Draw a line beteween two points (center of palm to finger)
106         /// </summary>
107         /// <param name="e">Paint event</param>
108         /// <param name="x">Horizontal coordinate of finger</param>
109         /// <param name="z">Vertical coordinate of finger</param>
110         private void DrawLinePoint(PaintEventArgs e, int x, int z)
111         {
112             // Create pen.
113             Pen Pen = new Pen(Color.Black, 3);
114 
115             // Create points that define line.
116             Point point1 = new Point(Convert.ToInt32(palmPanelPos.x),
                  Convert.ToInt32(palmPanelPos.z));
117             Point point2 = new Point(x, z);
118 
119             // Draw line to screen.
120             e.Graphics.DrawLine(Pen, point1, point2);
121         }
122         #endregion
123 
124         #region drawing colors
125         /// <summary>
126         /// draw user's hand in color
127         /// </summary>
128         /// <param name="e">paint event</param>
129         /// <param name="hand">hand of user</param>
130         /// <param name="colors">list of colors of user's finger</param>
131         public void paintHandColor(PaintEventArgs e, MyHand hand, List<Color>
              colors)
132         {
133             this.Hand = hand;
134             FingersPanelPos = normToPalmPanelPos();
135 
136             this.DrawEllipseRectangle(e, Convert.ToInt32(palmPanelPos.x),
                  Convert.ToInt32(palmPanelPos.z));
137             for (int i = 0; i < FingersPanelPos.Count; i++)
138             {
139                 this.DrawEllipseRectangleColor(e, Convert.ToInt32
                      (FingersPanelPos[i].x), Convert.ToInt32(FingersPanelPos
                      [i].z), colors[i]);
140                 this.DrawLinePointColor(e, Convert.ToInt32(FingersPanelPos
                      [i].x), Convert.ToInt32(FingersPanelPos[i].z), colors[i]);
141             }
142         }
143 
144         /// <summary>
145         /// Draw a circle at a certain
146         /// </summary>
147         /// <param name="e">Paint event</param>
148         /// <param name="x">Horizonzal coordinate of finger/palm</param>
149         /// <param name="z">Vertical coordinate of finger/palm</param>
150         /// <param name="penColor">color of the finger</param>
151         private void DrawEllipseRectangleColor(PaintEventArgs e, int x, int z,
               Color penColor)
152         {
```

```
153             // Create pen.
154             Pen Pen = new Pen(penColor, 3);
155 
156             // Create rectangle for ellipse.
157             Rectangle rect = new Rectangle(x - (CIRCLESIZE / 2), z -
                  (CIRCLESIZE / 2), CIRCLESIZE, CIRCLESIZE);
158 
159             // Draw ellipse to screen.
160             e.Graphics.DrawEllipse(Pen, rect);
161         }
162 
163         /// <summary>
164         /// Draw a line beteween two points (center of palm to finger)
165         /// </summary>
166         /// <param name="e">Paint event</param>
167         /// <param name="x">Horizontal coordinate of finger</param>
168         /// <param name="z">Vertical coordinate of finger</param>
169         /// <param name="penColor">color of the finger</param>
170         private void DrawLinePointColor(PaintEventArgs e, int x, int z, Color
              penColor)
171         {
172             // Create pen.
173             Pen Pen = new Pen(penColor, 3);
174 
175             // Create points that define line.
176             Point point1 = new Point(Convert.ToInt32(palmPanelPos.x),
                  Convert.ToInt32(palmPanelPos.z));
177             Point point2 = new Point(x, z);
178 
179             // Draw line to screen.
180             e.Graphics.DrawLine(Pen, point1, point2);
181         }
182         #endregion
183 
184         #region transform norm to panel position
185         /// <summary>
186         /// Calculate the position on the panel with the normalized vector
187         /// </summary>
188         /// <returns>A list of vector with the finger's position to the palm</
              returns>
189         public List<Vector> normToPalmPanelPos()
190         {
191             float scaleFactor = PanelHeight + CIRCLESIZE;
192             List<Vector> fingersPanelPos = new List<Vector>();
193             Vector originToPalm = new Vector(Hand.PalmNormPos.x, 0,
                  Hand.PalmNormPos.z);
194             List<Vector> originToFingers = new List<Vector>();
195 
196             for (int i = 0; i < Hand.FingersNormPos.Count; i++)
197             {
198                 originToFingers.Add(new Vector(Hand.FingersNormPos[i].x, 0,
                      Hand.FingersNormPos[i].z));
199 
200                 fingersPanelPos.Add(new Vector((-originToPalm +
                      originToFingers[i]) * scaleFactor + palmPanelPos));
201             }
```

```
202
203             return fingersPanelPos;
204         }
205
206         /// <summary>
207         /// Calculate panel position of the modele hand
208         /// </summary>
209         /// <param name="modele">the current modele</param>
210         /// <returns>A list of positions</returns>
211         public List<Vector> normToPalmPanelModelePos(MyHand modele)
212         {
213             float scaleFactor = PanelHeight + CIRCLESIZE;
214             List<Vector> modelePanelPos = new List<Vector>();
215             Vector originToPalm = new Vector(modele.PalmNormPos.x, 0,
                  modele.PalmNormPos.z);
216             List<Vector> originToFingers = new List<Vector>();
217
218             for (int i = 0; i < modele.FingersNormPos.Count; i++)
219             {
220                 originToFingers.Add(new Vector(modele.FingersNormPos[i].x, 0,
                      modele.FingersNormPos[i].z));
221
222                 modelePanelPos.Add(new Vector((-originToPalm + originToFingers
                      [i]) * scaleFactor + palmPanelPos));
223             }
224
225             ModelePanelPos = modelePanelPos;
226
227             return modelePanelPos;
228         }
229         #endregion
230     }
231 }
232
```

```
1  /*
2   * Author  : Dubas Loïc
3   * Class   : I.FA-P3B
4   * School  : CFPT-I
5   * Date    : June 2018
6   * Descr.  : serialize and deserialize functions and delete save position
7   * Version : 1.0
8   * Ext. dll: LeapCSharp.NET4.5
9   */
10
11 using System;
12 using System.Collections.Generic;
13 using System.Linq;
14 using System.Text;
15 using System.Threading.Tasks;
16 // References to add
17 using Leap;
18 using System.Diagnostics;
19 using System.Xml;
20 using System.Xml.Serialization;
21 using System.IO;
22
23 namespace fingers_cloner
24 {
25     [Serializable]
26     public class Serialization
27     {
28         #region Intialization
29         // serialize file name, directory name and file path
30         private string _positionName;
31         private string _dirName;
32         private string _filePath;
33
34         // store all positions serialized
35         List<MyHand> allPositions;
36
37         // store all the files name
38         List<string> allFilesName;
39
40         // hand to serialize
41         private MyHand _handToSerialize;
42         internal MyHand HandToSerialize { get => _handToSerialize; set =>
             _handToSerialize = value; }
43         public string PositionName { get => _positionName; set =>
             _positionName = value; }
44         // directory name and file path
45         public string DirName { get => _dirName; set => _dirName = value; }
46         public string FilePath { get => _filePath; set => _filePath = value; }
47         #endregion
48
49         /// <summary>
50         /// default constructor - initialize directory name
51         /// </summary>
52         public Serialization()
53         {
54             DirName = "serial";
```

```
55
56              Path.GetFileName(DirName);
57          }
58
59          /// <summary>
60          /// serialize a given MyHand object
61          /// </summary>
62          /// <param name="Hand">the hand to serialize</param>
63          public void serialize(MyHand Hand)
64          {
65              PositionName = Hand.Name;
66              FilePath = DirName + "/" + PositionName + ".xml";
67
68              XmlSerializer serializer = new XmlSerializer(typeof(MyHand));
69              StreamWriter file = new StreamWriter(FilePath);
70              serializer.Serialize(file, Hand);
71              file.Close();
72          }
73
74          /// <summary>
75          /// deserialize all xml files in serial directory
76          /// </summary>
77          /// <returns>a list of all the serialize hands</returns>
78          public List<MyHand> deserialize()
79          {
80              allPositions = new List<MyHand>();
81              allFilesName = getFilesName();
82
83              if (Directory.Exists(DirName))
84              {
85                  XmlSerializer serializer = new XmlSerializer(typeof(MyHand));
86                  foreach (string position in allFilesName)
87                  {
88                      FileStream stream = new FileStream(position,
                        FileMode.Open);
89                      allPositions.Add((MyHand)serializer.Deserialize(stream));
90                      stream.Close();
91                  }
92              }
93
94              return allPositions;
95          }
96
97          /// <summary>
98          /// get all the files name in the serial directory
99          /// </summary>
100         /// <returns>a list of all the names of the positions</returns>
101         public List<string> getFilesName()
102         {
103             allFilesName = new List<string>();
104
105             foreach (string fileName in Directory.GetFiles(DirName))
106             {
107                 allFilesName.Add(fileName);
108             }
109
```

```
110             return allFilesName;
111         }
112
113         /// <summary>
114         /// delete a saved position
115         /// </summary>
116         /// <param name="posName">the name of the position to delete</param>
117         public void deletePosition(string posName) {
118             FilePath = DirName + "/" + posName + ".xml";
119
120             File.Delete(FilePath);
121         }
122     }
123 }
124
```