```
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 
```