```
1  using System;
2
3  namespace TIPE_trajectoire
4  {
5      public static class Program
6      {
7          [STAThread]
8          static void Main()
9          {
10             using (var game = new Game1())
11                 game.Run();
12         }
13     }
14 }
15
```

```
1  using Microsoft.Xna.Framework;
2  using Microsoft.Xna.Framework.Content;
3  using Microsoft.Xna.Framework.Graphics;
4  using Microsoft.Xna.Framework.Input;
5  using System;
6  using System.Collections.Generic;
7  using System.Diagnostics;
8  using TIPE_trajectoire.items;
9
10 namespace TIPE_trajectoire
11 {
12     public class Game1 : Game
13     {
14         private GraphicsDeviceManager _graphics;
15         private SpriteBatch _spriteBatch;
16
17         private Scene scene;
18
19         public Game1()
20         {
21             _graphics = new GraphicsDeviceManager(this);
22             Content.RootDirectory = "Content";
23             IsMouseVisible = true;
24         }
25
26         protected override void Initialize() //charge les textures, cette
             fonction est appellée au lancement du code
27         {
28
29             _graphics.PreferredBackBufferWidth = 1250;
30             _graphics.PreferredBackBufferHeight = 800;
31             _graphics.ApplyChanges();
32             StaticRessources.ContentManager = Content;
33             StaticRessources.dot = Content.Load<Texture2D>("dot");
34             StaticRessources.line = Content.Load<Texture2D>("d");
35             scene = new Scene();
36             scene.Initialize();
37             base.Initialize();
38         }
39
40         protected override void LoadContent() //fonction répétée en boucle
41         {
42             _spriteBatch = new SpriteBatch(GraphicsDevice);
43
44
45         }
46
47         protected override void Update(GameTime gameTime) //fonction répétée
             en boucle
48         {
49             if (GamePad.GetState(PlayerIndex.One).Buttons.Back ==
               ButtonState.Pressed || Keyboard.GetState().IsKeyDown
               (Keys.Escape))
```

```
50                 Exit();
51
52             scene.Update(gameTime);
53             scene.Update(gameTime);
54             scene.Update(gameTime);
55             scene.Update(gameTime);
56             scene.Update(gameTime);
57
58             base.Update(gameTime);
59         }
60
61         protected override void Draw(GameTime gameTime) //fonction répétée en
               boucle
62         {
63             GraphicsDevice.Clear(Color.CornflowerBlue);
64
65             _spriteBatch.Begin();
66
67             scene.Draw(_spriteBatch);
68
69             _spriteBatch.End();
70
71             base.Draw(gameTime);
72         }
73     }
74
75     public static class StaticRessources //ressources utilisées dans le code
76     {
77         public static ContentManager ContentManager;
78         public static Texture2D dot;
79         public static Texture2D line;
80         public static float G = (float)(8.64 * Math.Pow(10, -13));
81         public static float dt = 50f;
82         public static float sunMass = 1.989f * (float)Math.Pow(10, 30);
83     }
84
85     public static class LineRenderer// extension aux bibiolthéques XNA
         utilisées pour dessiner nos objets, l'extension permet de dessiner des
         lignes droites
86     {
87         public static void DrawLine(this SpriteBatch spriteBatch, Vector2
             start, Vector2 end, Color color)
88         {
89             spriteBatch.Draw(StaticRessources.line, new Vector2(500, 300) +
                 start, null, color,
90                              (float)Math.Atan2(end.Y - start.Y, end.X -
                     start.X),
91                              new Vector2(0f, (float)
                     StaticRessources.line.Height / 2),
92                              new Vector2(Vector2.Distance(start, end), 5f),
93                              SpriteEffects.None, 0f);
94         }
95     }
```

```
96
97  }
98
99
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using TIPE_trajectoire.items;
5  using Microsoft.Xna.Framework;
6  using Microsoft.Xna.Framework.Content;
7  using Microsoft.Xna.Framework.Graphics;
8  using Microsoft.Xna.Framework.Input;
9  using System.Diagnostics;
10
11 namespace TIPE_trajectoire
12 {
13     public class Scene
14     {
15
16         public List<Solid> solids;
17
18         public Scene()
19         {
20
21         }
22
23         public void Initialize()
24         {
25
26             float dtheta = 0.5f*(float)Math.Sqrt(4 * Math.Pow(Math.PI,2) *
                 Math.Pow(188.75 * Math.Pow(10,6),3)/Math.Pow(227.9f * (float)
                 Math.Pow(10, 6), 3));//angle parcouru par Mars pendant le
                 trajet de la fusée
27             solids = new List<Solid>();
28             //ajout des planères, du soleil, des fusées
29             solids.Add(new Solid((float)(2 * Math.Pow(10,30)), new Vector2(),
                  new Vector2()));
30             solids.Add(Luminary.CreateOrbit(57.91f * (float)Math.Pow(10, 6),
                 (float)Math.PI, 3.285f * (float)Math.Pow(10, 23)));
31             solids.Add(Luminary.CreateOrbit(108.2f * (float)Math.Pow(10, 6),
                 10, 4.867f * (float)Math.Pow(10, 24)));
32             solids.Add(Luminary.CreateOrbit(149.6f * (float)Math.Pow(10, 6),
                 20, 5.972f * (float)Math.Pow(10, 24)));
33             solids.Add(Luminary.CreateOrbit(227.9f * (float)Math.Pow(10, 6),
                 20 + (float)Math.PI - dtheta, 6.39f * (float)Math.Pow(10,
                 23)));
34             solids.Add(Luminary.CreateOrbit(778.5f * (float)Math.Pow(10, 6),
                 40, 5.683f * (float)Math.Pow(10, 27)));
35             solids.Add(Luminary.CreateOrbit(1434f * (float)Math.Pow(10, 6),
                 50, 1.8898f * (float)Math.Pow(10, 26)));
36             solids.Add(Luminary.CreateOrbit(2871f * (float)Math.Pow(10, 6),
                 60, 8.681f * (float)Math.Pow(10, 25)));
37             solids.Add(Luminary.CreateOrbit(4495f * (float)Math.Pow(10, 6),
                 70, 1.024f * (float)Math.Pow(10, 26)));
38             solids.Add(Rocket.CreateOrbit(20, 1000000, Rocket.State.classic,
                 solids[3], solids[4]));//on change le rocket state suivant les
                 approximations physiques faites
```

```
39              //change la coulleur de la terre, du soleil et mars, et active le ↪
                   dessin de certaines trajectoires
40              solids[4].color = Color.Red; solids[4].DrawTrail = true;
41              solids[3].color = Color.Blue; solids[3].DrawTrail = true;
42              solids[solids.Count-1].DrawTrail = true;
43              solids[0].color = Color.Yellow;
44          }
45
46          public void Update(GameTime gameTime)
47          {
48                  foreach (Solid solid in solids)
49                  {
50                      solid.UpdateS(gameTime, solids);
51                  }
52                  foreach (Solid solid in solids)
53                  {
54                      solid.UpdateP(gameTime);
55                  }
56          }
57
58          public void Draw(SpriteBatch spriteBatch)
59          {
60              foreach (Solid solid in solids)
61              {
62                  solid.Draw(spriteBatch);
63              }
64
65          }
66
67      }
68  }
69
```

```
1  using Microsoft.Xna.Framework;
2  using Microsoft.Xna.Framework.Graphics;
3  using System;
4  using System.Collections.Generic;
5
6
7  namespace TIPE_trajectoire.items
8  {
9      public class Solid
10     {
11
12         public Trail trail = new Trail(); //dessine les trajectoires
13         public bool DrawTrail = false;
14
15         public float mass;
16         public Vector2 position;
17         public Vector2 speed;
18         public Color color = Color.White;
19         public Solid()// on doit avoir ce constructeur vide pour les classes
             filles
20         {
21
22         }
23         public Solid(float mass, Vector2 position, Vector2 speed) //crée un
             solid
24         {
25             this.mass = mass;
26             this.speed = speed;
27             this.position = position;
28         }
29         public virtual void UpdateS(GameTime gameTime, List<Solid> solids) //
             mise à jour de la vitesse
30         {
31             if(new Random().NextDouble() < 0.1)
32                 trail.points.Add(position / 1000000);
33         }
34         public virtual void UpdateP(GameTime gameTime) //mise à jour de la
             position
35         {
36             position += speed * StaticRessources.dt * (float)
                 gameTime.ElapsedGameTime.TotalSeconds;
37         }
38
39         public virtual Vector2 Champ(Vector2 pos) //champ gravitationel de
             l'object en un point
40         {
41             return mass * StaticRessources.G * (position - pos) / (((float)
                 Math.Pow(Vector2.Distance(pos, position), 3)));
42         }
43
44         public virtual void Draw(SpriteBatch spriteBatch) //dessinne l'object
45         {
46             spriteBatch.Draw(StaticRessources.dot, new Vector2(500, 300) +
```

```
                position / 1000000, null, color, 0, new Vector2(), 0.05f,
                SpriteEffects.None, 0);
            if (DrawTrail)
                trail.Draw(spriteBatch, color);
        }


    }
}

```

```
1  using Microsoft.Xna.Framework;
2  using Microsoft.Xna.Framework.Graphics;
3  using System.Collections.Generic;
4
5  namespace TIPE_trajectoire.items
6  {
7      public class Trail
8      {
9
10         public List<Vector2> points = new List<Vector2>(); //points de la
             trajectoire
11
12         public void Draw(SpriteBatch spriteBatch, Color color)
13         {
14             for (int i = 0; i < points.Count - 1; i++)
15             {
16                 spriteBatch.DrawLine(points[i], points[i + 1], color);
17             }
18         }
19     }
20 }
21
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Text;
5  using Microsoft.Xna.Framework;
6  using Microsoft.Xna.Framework.Content;
7  using Microsoft.Xna.Framework.Graphics;
8  using Microsoft.Xna.Framework.Input;
9  using TIPE_trajectoire.items;
10
11 namespace TIPE_trajectoire.items
12 {
13     public class Luminary : Solid // code le soleil et les planètes
14     {
15         public Luminary(float mass, Vector2 position, Vector2 speed) // crée
             un astre
16         {
17             this.mass = mass;
18             this.position = position;
19             this.speed = speed;
20         }
21
22         public override void UpdateS(GameTime gameTime, List<Solid>
             solids) // mise à jour de la vitesse
23         {
24
25             if (gameTime.ElapsedGameTime.TotalHours < StaticRessources.dt)
26             {
27                 foreach (Solid solid in solids)
28                 {
29                     if (solid != this)
30                     {
31                         speed += StaticRessources.dt * solid.Champ(position)
                       * (float)gameTime.ElapsedGameTime.TotalSeconds;
32                     }
33                 }
34             }
35
36             base.UpdateS(gameTime, solids); // exécute le code de la classe
                 mère
37         }
38         public static Luminary CreateOrbit(float radius, float angle, float
             mass) //crée un astre sur une orbite
39         {
40
41             Vector2 pos = new Vector2((float)Math.Cos(angle) * radius,
                 (float)Math.Sin(angle) * radius);
42             Vector2 sp = (float)Math.Sqrt(StaticRessources.sunMass *
                 StaticRessources.G / radius) *
43                 Vector2.Normalize(Vector2.Transform(pos,
                     Matrix.CreateRotationZ((float)Math.PI / 2)));
44             return new Luminary(mass, pos, sp);
45         }
```

```
46
47      }
48  }
49
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Text;
5  using Microsoft.Xna.Framework;
6  using Microsoft.Xna.Framework.Content;
7  using Microsoft.Xna.Framework.Graphics;
8  using Microsoft.Xna.Framework.Input;
9  using TIPE_trajectoire.items;
10
11
12 namespace TIPE_trajectoire.items
13 {
14     public class Rocket : Solid // fusée
15     {
16
17         public enum State { classic, sphereinfluence } //option pour le
             choix des hypothèses de physique aplliquées à la fusée
18
19         private State _state;
20         public State state //permet de changer la coulleur de la fusée en
             changeant les hypothèses de physique aplliquées
21         {
22             get { return _state; }
23
24             set
25             {
26                 if (value == State.classic)
27                 {
28                     color = Color.Green;
29                 }
30                 else
31                 {
32                     color = Color.Black;
33                 }
34                 _state = value;
35             }
36         }
37         public Rocket(float mass, Vector2 position, Vector2 speed, State
             state) //crée une fusée
38         {
39             this.mass = mass;
40             this.speed = speed;
41             this.position = position;
42             this.state = state;
43         }
44
45         public override void UpdateS(GameTime gameTime, List<Solid>
             solids) //mise à jour de la vitesse
46         {
47
48             switch(state)
49             {
```

```
50                  case State.classic://on considère tout les champs gravitationnels
51
52                      if (gameTime.ElapsedGameTime.TotalHours < StaticRessources.dt)
53                      {
54                          foreach (Solid solid in solids)
55                          {
56                              if (solid != this)
57                              {
58                                  speed += StaticRessources.dt * solid.Champ(position) * (float)gameTime.ElapsedGameTime.TotalSeconds;
59                              }
60
61
62                          }
63                      }
64                      break;
65                  case State.sphereinfluence: // on ne considère que le champ gravitationnel le plus grand
66
67                      List<(float,int)> normes = new List<(float,int)>();
68
69                      if (gameTime.ElapsedGameTime.TotalHours < StaticRessources.dt)
70                      {
71                          for(int i = 0;i < solids.Count;i++)
72                          {
73                              if (solids[i] != this)
74                              {
75                                  normes.Add((Vector2.Distance(solids[i].Champ(position),new Vector2()),i));
76                              }
77                          }
78
79                          normes.Sort();//tri par ordre lexicographique
80                          int o = normes[normes.Count-1].Item2;
81                          speed += solids[o].Champ(position) * StaticRessources.dt * (float)gameTime.ElapsedGameTime.TotalSeconds;
82                      }
83                      break;
84
85              }
86
87              base.UpdateS(gameTime, solids);
88          }
89
90          public override Vector2 Champ(Vector2 pos) //on considère le champ de la fusée comme nul
91          {
92              return new Vector2();
```

```
93          }
94          public static Rocket CreateOrbit(float angle, float mass, State
              state, Solid origin, Solid target)//crée un fusée sur un
              trajectoire
95          {
96              float radius = Vector2.Distance(origin.position, new Vector2());
97              float targetedradius = Vector2.Distance(target.position, new
                  Vector2());
98              float decalage = 2*(float)Math.Asin(0.5 * Math.Sqrt
                  (origin.mass / StaticRessources.sunMass));
99              float speed = (float)Math.Sqrt(StaticRessources.sunMass *
                  StaticRessources.G * ((2 / radius) - (1 / (0.5 * (radius +
                  targetedradius)))));
100             if (state == State.classic)
101                 speed += 1892;
102             Vector2 pos = new Vector2((float)Math.Cos(angle+decalage) *
                  radius, (float)Math.Sin(angle+decalage) * radius);
103             Vector2 sp = speed * Vector2.Normalize(Vector2.Transform(pos,
                  Matrix.CreateRotationZ((float)Math.PI / 2)));
104             return new Rocket(mass, pos, sp, state);
105         }
106
107
108     }
109 }
110
```

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Diagnostics;
 4  using System.Linq;
 5  using System.Threading.Tasks;
 6  using System.Windows.Forms;
 7
 8  namespace trajectoires_fusée
 9  {
10      static class Program
11      {
12          [STAThread]
13          static void Main()
14          {
15              Application.Run(new Form1());
16          }
17      }
18  }
19
```

```
1  using System;
2  using System.Collections.Generic;
3  using System.Windows.Forms;
4
5  namespace trajectoires_fusée
6  {
7      public partial class Form1 : Form
8      {
9
10         public decimal isp = 330;
11         public decimal g0 = 10;
12         public decimal q = 19696; //débit masssique lors du décollage
13         public decimal mass = 5000000; //masse initiale de la fusée
14         public decimal lambda = (decimal)(4.5 * Math.PI * Math.PI * 0.5  *
             1.5); //coeffiscient utile dans le calcul des forces de frottement
15
16         private List<decimal> x = new List<decimal>(); //échelle de temps
17         private List<decimal> acc = new List<decimal>(); //accélération au
             fil du temps
18         private List<decimal> y = new List<decimal>(); //vitesse au fil du
             temps
19         private List<decimal> w = new List<decimal>(); //altitude au fil du
             temps
20         private List<decimal> mm = new List<decimal>(); //masse volumique de
              l'air traversé au fil du temps
21         public static int n = 1000000; //nombre de passage de boucle
22
23         private static decimal G = (decimal)(6.66 * Math.Pow(10, -11)); //
             contante gravitationnelle
24         private static decimal earthmass = (decimal)(5.972 * Math.Pow(10,
             24)); //masse de la terre
25         private static decimal earthradius = (decimal)(6371000); //rayon de
             la terre
26
27         private static decimal marsmass = (decimal)(6.417 * Math.Pow(10,
             23)); //masse de Mars
28         private static decimal marsradius = 3389500; //rayon de Mars
29         private static decimal marsrocketmass = 254246; //masse de la fusée
             quand elle arrive près de Mars
30         private static decimal marslambda = (decimal)(9 * 50 * 0.5 *
             1.5); //coeffiscient utile dans le calcul des forces de frottement
31
32         public Form1()
33         {
34             InitializeComponent();
35         }
36
37         public void UpdateChart(object sender, EventArgs e) //action à
             effectuer lors que l'on veut tracer la position au fil du temps
             lors du décollage
38         {
39             ResoudrePhase1(n, 0, 0, 0, 170);
40             Chart.Series["coordonées"].Points.Clear();
```

```
41              for (int i = 0; i < x.Count / 100; i += 1)
42              {
43                  Chart.Series["coordonées"].Points.AddXY(x[i*100].ToString(),
                        w[i*100]);
44              }
45          }
46
47
48          private void ResoudrePhase1(int n, decimal z0 = 0, decimal v0 = 0,
              decimal dv0 = 0, decimal stop = 1) // fonction qui résout les
              équations différentielles
49          {
50
51              decimal time = stop;
52              decimal dt = time / (decimal)n;
53              decimal t = 0;
54              List<decimal> z = new List<decimal>();
55              List<decimal> v = new List<decimal>();
56              List<decimal> dv = new List<decimal>();
57              List<decimal> tt = new List<decimal>();
58              dv.Clear();
59              v.Clear();
60              z.Clear();
61              tt.Clear();
62              dv.Add(0);
63              v.Add(v0);
64              z.Add(z0);
65              mm.Add(massevolumique(z0));
66              tt.Add(0);
67
68              int o = n;
69
70              for (int i = 0; i < n; i++)
71              {
72
73                  decimal[] l = equation(t, z[i], v[i], dv[i], dt, i);
74                  z.Add(l[0]);
75                  v.Add(l[1]);
76                  dv.Add(l[2]);
77                  t += dt;
78                  tt.Add(t);
79
80              }
81              acc = dv;
82              x = tt;
83              y = v;
84              w = z;
85          }
86          public decimal[] equation(decimal t, decimal z, decimal v, decimal
              dv, decimal dt, int i) // méthode d'euler
87          {
88              decimal nexta;
89
```

```
90              if (z < 40000)
91              {
92                  nexta = (isp * g0 * q) * (1 / (mass - (q * t))) +
                      effgravitationalforce(z, t)
93                  - (1 / (mass - q * t)) * frott(t, z, v, i);
94              }
95              else
96              {
97                  nexta = isp * g0 * q * (1 / (mass - (q * t))) +
                      effgravitationalforce(z, t);
98              }
99
100
101             decimal nextv = v + dv * dt;
102             decimal nextz = z + v * dt;
103
104             return new decimal[3] { nextz, nextv, nexta };
105
106         }
107
108         public decimal frott(decimal t, decimal z, decimal v, int i) //
              retourne la force de frottement
109         {
110             return lambda * speedsquared(v) * massevolumique(z);
111         }
112
113         public decimal massevolumique(decimal z) //retourne de la masse
              volumique de l'atmosphère terrestre à une altitude z
114         {
115             return (decimal)(352.995 * Math.Pow(1 - (0.0000225577 *
                  decimal.ToDouble(z)), 5.25516) / (288.15 - 0.0065 *
                  decimal.ToDouble(z)));
116         }
117
118         private decimal speedsquared(decimal v) //retourne le carré de v
119         {
120             return (decimal)(Math.Pow(decimal.ToDouble(v), 2));
121         }
122         public decimal effgravitationalforce(decimal z, decimal t) //
              retourne la force de gravitation à une altitude z
123         {
124
125             decimal _value = -G * earthmass / (earthradius + z);
126             _value *= 1 / (earthradius + z);
127             return _value;
128
129         }
130
131         //les fonctions qui suivent sont des adaptations des précédentes à
              l'atterissage sur Mars
132
133         private void button1_Click(object sender, EventArgs e)
134         {
```

```
135             ResoudrePhase2(n, 100000, 0, 500);
136             Chart.Series["coordonées"].Points.Clear();
137             for (int i = 0; i < x.Count / 100; i += 1)
138             {
139                 Chart.Series["coordonées"].Points.AddXY(x[100 * i].ToString
                    (), y[100 * i]);
140             }
141         }
142
143
144
145         private void ResoudrePhase2(int n, decimal z0 = 0, decimal v0 = 0,
              decimal stop = 1)
146         {
147             decimal time = stop;
148             decimal dt = time / (decimal)n;
149             decimal t = 0;
150             List<decimal> z = new List<decimal>();
151             List<decimal> v = new List<decimal>();
152             List<decimal> dv = new List<decimal>();
153             List<decimal> tt = new List<decimal>();
154             dv.Clear();
155             v.Clear();
156             z.Clear();
157             tt.Clear();
158             dv.Add(0);
159             v.Add(v0);
160             z.Add(z0);
161             tt.Add(0);
162
163             for (int i = 0; i < n; i++)
164             {
165
166                 decimal[] l = equation3(t, z[i], v[i], dv[i], dt, i);
167                 z.Add(l[0]);
168                 v.Add(l[1]);
169                 dv.Add(l[2]);
170                 t += dt;
171                 tt.Add(t);
172             }
173             acc = dv;
174             x = tt;
175             y = v;
176             w = z;
177         }
178
179         private decimal[] equation3(decimal t, decimal z, decimal v, decimal
              dv, decimal dt, int i)
180         {
181             decimal nexta = 0;
182
183             if (z < 30000 && z > 0)
184             {
```

```
185
186                 nexta = effgravitationalforcem(z, t)
187                 + (1 / (marsrocketmass)) * Frottmars(t, z, v, i);
188             }
189             else if (z > 0)
190             {
191                 nexta = effgravitationalforcem(z, t);
192             }
193             else
194             {
195                 nexta = 0;
196                 v = 0;
197                 z = 0;
198             }
199
200             decimal nextv = v + dv * dt;
201             decimal nextz = z + v * dt;
202
203             return new decimal[3] { nextz, nextv, nexta };
204         }
205
206         public decimal effgravitationalforcem(decimal z, decimal t)
207         {
208
209             decimal _value = -G * marsmass / (marsradius + z);
210             _value *= 1 / (marsradius + z);
211             return _value;
212
213         }
214
215         public decimal massevolumiquemars(decimal z)
216         {
217             return (decimal)(3.17 * Math.Pow(1 - (0.00000847 *
                  decimal.ToDouble(z)), 7.85374) / (295.15 - 0.0025 *
                  decimal.ToDouble(z)));
218         }
219
220         public decimal Frottmars(decimal t, decimal z, decimal v, int i)
221         {
222             return marslambda * speedsquared(v) * massevolumiquemars(z);
223         }
224
225     }
226 }
227
```

```
1  namespace trajectoires_fusée
2  {
3      partial class Form1
4      {
5          private System.ComponentModel.IContainer components = null;
6
7          protected override void Dispose(bool disposing)
8          {
9              if (disposing && (components != null))
10             {
11                 components.Dispose();
12             }
13             base.Dispose(disposing);
14         }
15
16         #region Code généré par le Concepteur Windows Form je l'ai seulement
               légérement modifié
17
18         private void InitializeComponent()
19         {
20             System.Windows.Forms.DataVisualization.Charting.ChartArea
                   chartArea1 = new
                   System.Windows.Forms.DataVisualization.Charting.ChartArea();
21             System.Windows.Forms.DataVisualization.Charting.Legend legend1 =
                   new System.Windows.Forms.DataVisualization.Charting.Legend();
22             System.Windows.Forms.DataVisualization.Charting.Series series1 =
                   new System.Windows.Forms.DataVisualization.Charting.Series();
23             this.Chart = new
                   System.Windows.Forms.DataVisualization.Charting.Chart();
24             this.Button = new System.Windows.Forms.Button();
25             this.button1 = new System.Windows.Forms.Button();
26             ((System.ComponentModel.ISupportInitialize)
                   (this.Chart)).BeginInit();
27             this.SuspendLayout();
28
29             chartArea1.Name = "ChartArea1";
30             this.Chart.ChartAreas.Add(chartArea1);
31             legend1.Name = "Legend1";
32             this.Chart.Legends.Add(legend1);
33             this.Chart.Location = new System.Drawing.Point(37, 45);
34             this.Chart.Name = "Chart";
35             series1.ChartArea = "ChartArea1";
36             series1.ChartType =
                   System.Windows.Forms.DataVisualization.Charting.SeriesChartType
                   .Spline;
37             series1.Legend = "Legend1";
38             series1.Name = "coordonées";
39             this.Chart.Series.Add(series1);
40             this.Chart.Size = new System.Drawing.Size(1050, 476);
41             this.Chart.TabIndex = 0;
42             this.Chart.Text = "chart1";
43
44             this.Button.Location = new System.Drawing.Point(1131, 76);
```

```
45              this.Button.Name = "Button";
46              this.Button.Size = new System.Drawing.Size(113, 43);
47              this.Button.TabIndex = 1;
48              this.Button.Text = "Terre";
49              this.Button.UseVisualStyleBackColor = true;
50              this.Button.Click += new System.EventHandler(this.UpdateChart);
51
52              this.button1.Location = new System.Drawing.Point(1131, 125);
53              this.button1.Name = "button1";
54              this.button1.Size = new System.Drawing.Size(113, 45);
55              this.button1.TabIndex = 4;
56              this.button1.Text = "Mars";
57              this.button1.UseVisualStyleBackColor = true;
58              this.button1.Click += new System.EventHandler
                  (this.button1_Click);
59
60              this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 16F);
61              this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
62              this.ClientSize = new System.Drawing.Size(1308, 559);
63              this.Controls.Add(this.button1);
64              this.Controls.Add(this.Button);
65              this.Controls.Add(this.Chart);
66              this.Name = "Form1";
67              this.Text = "Form1";
68              ((System.ComponentModel.ISupportInitialize)(this.Chart)).EndInit
                  ();
69              this.ResumeLayout(false);
70
71          }
72
73          #endregion
74
75
76
77
78          private System.Windows.Forms.DataVisualization.Charting.Chart Chart;
79          private System.Windows.Forms.Button Button;
80          private System.Windows.Forms.Button button1;
81      }
82  }
83
84
```