

Code Source



Travail Pratique Individuel MiFiSy

CFPT Informatique

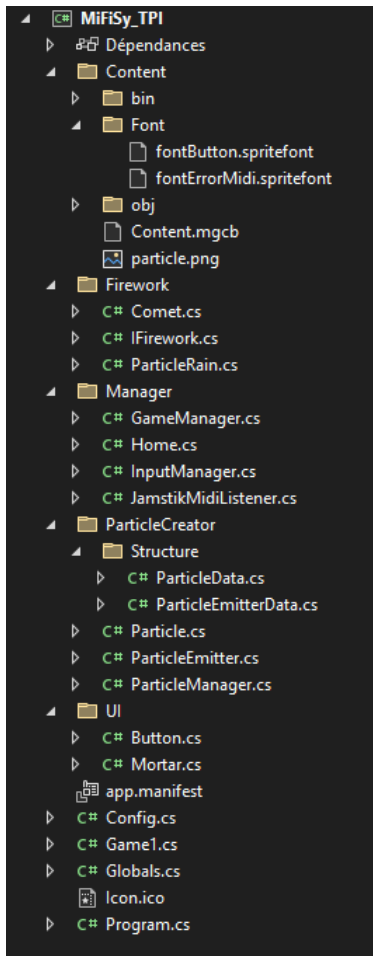
Yoann Meier

15 mai 2024

Table des matières

Structure des fichiers	2
Config.cs	3
Globals.cs	5
Game1.cs	7
ParticleData.cs	9
ParticleEmitterData.cs	10
Particle.cs	11
ParticleEmitter.cs	13
ParticleManager.cs	15
Button.cs	17
Mortar.cs	19
IFirework.cs	21
ParticleRain.cs	22
Comet.cs	25
JamstikMidiListener.cs	28
InputManager	30
Home.cs	31
GameManager.cs	33

Structure des fichiers



Config.cs

```

1 using Microsoft.Xna.Framework;
2 using System;
3 using System.Collections.Generic;
4 using System.Linq;
5 using System.Xml.Linq;
6 /*
7  * Auteur : Yoann Meier
8  * Date : 06/05/2024
9  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
10  * Description de la page : Class permettant de récupérer en static toutes les données du ↵
    fichier de configuration
11 */
12 namespace MiFiSy_TPI
13 {
14     internal class Config
15     {
16         private static XElement _configElement;
17
18         public Config()
19         {
20             _configElement = ↵
                XDocument.Load("config.xml").Descendants("Config").FirstOrDefault();
21         }
22
23         // Propriétés statiques pour accéder aux valeurs du fichier XML
24         public static string AUTHOR_FILE { get => ↵
            _configElement.Descendants("Author").FirstOrDefault().Value; }
25         public static string NAME_SEQUENCE { get => ↵
            _configElement.Descendants("NameSequence").FirstOrDefault().Value; }
26         public static string PATH_MUSIC { get => ↵
            _configElement.Descendants("PathMusic").FirstOrDefault().Value; }
27         public static string PATH_IMG { get => ↵
            _configElement.Descendants("PathImg").FirstOrDefault().Value; }
28         public static string PATH_SAVE_SEQUENCE { get => ↵
            _configElement.Descendants("PathSaveSequence").FirstOrDefault().Value; }
29         public static List<XElement> ALL_MORTAR { get => ↵
            _configElement.Descendants("Mortar").ToList(); }
30         public static Color COLOR_START_COMET { get => ↵
            Globals.GetColorFromElement(_configElement.Descendants("ColorStartComet")
31             .FirstOrDefault()); }
32         public static Color COLOR_END_COMET { get => ↵
            Globals.GetColorFromElement(_configElement.Descendants("ColorEndComet")
33             .FirstOrDefault()); }
34         public static Color COLOR_PARTICLE_RAIN_START { get => ↵
            Globals.GetColorFromElement(_configElement.Descendants("ColorStartParticleRain")
35             .FirstOrDefault()); }
36         public static Color COLOR_PARTICLE_RAIN_END { get => ↵
            Globals.GetColorFromElement(_configElement.Descendants("ColorEndParticleRain")
37             .FirstOrDefault()); }
38         public static int PARTICLE_RAIN_SIZE { get => ↵
            Convert.ToInt32(_configElement.Descendants("ParticleRain").FirstOrDefault().
39             Attribute("sizeParticle").Value); }
40         public static int PARTICLE_RAIN_NB { get => ↵
            Convert.ToInt32(_configElement.Descendants("ParticleRain").FirstOrDefault().
41             Attribute("nbParticle").Value); }
42         public static float PARTICLE_RAIN_LIFESPAN { get => ↵
            float.Parse(_configElement.Descendants("ParticleRain").FirstOrDefault().
43             Attribute("lifeSpan").Value); }
44         public static float PARTICLE_RAIN_TIME_SPAWN { get => ↵
            float.Parse(_configElement.Descendants("ParticleRain").FirstOrDefault().
45             Attribute("timeSpawn").Value); }
46         public static float PARTICLE_RAIN_SPEED { get => ↵
            float.Parse(_configElement.Descendants("ParticleRain").FirstOrDefault().
47             Attribute("defaultSpeed").Value); }
48         public static int COMET_MAIN_SIZE { get => ↵
            Convert.ToInt32(_configElement.Descendants("Comet").FirstOrDefault().
49             Attribute("sizeMainParticle").Value); }
50         public static int COMET_OTHER_SIZE { get => ↵
            Convert.ToInt32(_configElement.Descendants("Comet").FirstOrDefault().
51             Attribute("sizeOtherParticle").Value); }

```

```
52 |         public static float COMET_DEFAULT_SPEED { get => ↵  
53 |             float.Parse(_configElement.Descendants("Comet").FirstOrDefault().  
54 |                 Attribute("defaultSpeed").Value); }  
55 |         public static float COMET_DEFAULT_LIFESPAN { get => ↵  
56 |             float.Parse(_configElement.Descendants("Comet").FirstOrDefault().  
57 |                 Attribute("defaultLifespan").Value); }  
    |     }  
    | }
```

Globals.cs

```
1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Content;
3 using Microsoft.Xna.Framework.Graphics;
4 using MiFiSy_TPI.Firework;
5 using MiFiSy_TPI.Manager;
6 using System;
7 using System.Collections.Generic;
8 using System.Xml.Linq;
9 /*
10  * Auteur : Yoann Meier
11  * Date : 06/05/2024
12  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
13  * Description de la page : Page contenant des valeurs static nécessaires dans plusieurs pages
14  */
15 namespace MiFiSy_TPI
16 {
17     internal static class Globals
18     {
19         public static float TotalSeconds { get; set; }
20
21         /// <summary>
22         /// Enum de toutes les pages de l'application
23         /// </summary>
24         public enum AllPage
25         {
26             Home,
27             Game,
28         }
29
30         /// <summary>
31         /// Page actuel
32         /// </summary>
33         public static AllPage ActualPage { get; set; }
34
35         public static ContentManager Content { get; set; }
36
37         public static SpriteBatch SpriteBatch { get; set; }
38
39         public static SpriteFont FontButton { get; set; }
40
41         public static GraphicsDevice GraphicsDevice { get; set; }
42
43         public static Random Random { get; set; } = new Random();
44
45         /// <summary>
46         /// Largeur de l'écran
47         /// </summary>
48         public static int ScreenWidth { get; set; }
49
50         /// <summary>
51         /// Hauteur de l'écran
52         /// </summary>
53         public static int ScreenHeight { get; set; }
54
55         public static string MusicSelectedName { get; set; }
56
57         public static Home home { get; set; }
58
59         public static GameManager GameManager { get; set; }
60
61         /// <summary>
62         /// Liste de feu d'artifice
63         /// </summary>
64         public static List<IFirework> LstFirework { get; set; }
65
66         public static void Update(GameTime gt)
67         {
68             TotalSeconds = (float)gt.ElapsedGameTime.TotalSeconds;
69         }
70
71         /// <summary>
```

```
72     /// Retourne un float aléatoire entre min et max
73     /// </summary>
74     /// <param name="min">nombre minimum</param>
75     /// <param name="max">nombre maximum</param>
76     /// <returns>nombre aléatoire</returns>
77     public static float RandomFloat(float min, float max)
78     {
79         return (float)(Random.NextDouble() * (max - min)) + min;
80     }
81
82     /// <summary>
83     /// Retourne un nombre aléatoire entre min et max
84     /// </summary>
85     /// <param name="min">nombre minimum</param>
86     /// <param name="max">nombre maximum</param>
87     /// <returns>nombre aléatoire</returns>
88     public static int RandomInt(int min, int max)
89     {
90         return Random.Next(min, max + 1);
91     }
92
93     /// <summary>
94     /// Méthode pour récupérer la couleur à partir d'un élément XML
95     /// </summary>
96     /// <param name="colorElement">XElement contenant les attributs "r", "g" et "b"</param>
97     public static Color GetColorFromElement(XElement colorElement)
98     {
99         if (colorElement.Attribute("r") != null && colorElement.Attribute("g") != null && colorElement.Attribute("b") != null)
100         {
101             int r = Convert.ToInt32(colorElement.Attribute("r").Value);
102             int g = Convert.ToInt32(colorElement.Attribute("g").Value);
103             int b = Convert.ToInt32(colorElement.Attribute("b").Value);
104             return new Color(r, g, b);
105         }
106         return Color.White;
107     }
108 }
109 }
```

Game1.cs

```
1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Graphics;
3 using Microsoft.Xna.Framework.Media;
4 using MiFiSy_TPI.Firework;
5 using MiFiSy_TPI.Manager;
6 using MiFiSy_TPI.ParticleCreator;
7 using System.Collections.Generic;
8 /*
9  * Auteur : Yoann Meier
10 * Date : 06/05/2024
11 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
12 * Description de la page : Page principale de l'application
13 */
14 namespace MiFiSy_TPI
15 {
16     public class Game1 : Game
17     {
18         private GraphicsDeviceManager _graphics;
19         private SpriteBatch _spriteBatch;
20         private JamstikMidiListener _jamstikMidiListener;
21
22         public Game1()
23         {
24             _graphics = new GraphicsDeviceManager(this);
25             Content.RootDirectory = "Content";
26             IsMouseVisible = true;
27         }
28
29         protected override void Initialize()
30         {
31             // taille de l'application en fonction de la taille de l'écran
32             _graphics.PreferredBackBufferWidth = ←
33                 GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Width;
34             _graphics.PreferredBackBufferHeight = ←
35                 GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Height;
36             _graphics.IsFullScreen = false;
37             _graphics.ApplyChanges();
38
39             Globals.ScreenWidth = _graphics.PreferredBackBufferWidth;
40             Globals.ScreenHeight = _graphics.PreferredBackBufferHeight;
41             Globals.Content = Content;
42             Globals.GraphicsDevice = GraphicsDevice;
43             Globals.ActualPage = Globals.AllPage.Home;
44             Globals.lstFirework = new List<IFirework>();
45             Globals.MusicSelectedName = "";
46             Globals.FontButton = Content.Load<SpriteFont>("Font/fontButton");
47
48             // Permet de mettre en boucle les musiques
49             MediaPlayer.IsRepeating = true;
50
51             new Config();
52             Globals.home = new Home();
53             base.Initialize();
54         }
55
56         protected override void LoadContent()
57         {
58             _spriteBatch = new SpriteBatch(GraphicsDevice);
59             Globals.SpriteBatch = _spriteBatch;
60             _jamstikMidiListener = new ←
61                 JamstikMidiListener(Content.Load<SpriteFont>("Font/fontErrorMidi"));
62         }
63
64         protected override void Update(GameTime gameTime)
65         {
66             Globals.Update(gameTime);
67             InputManager.Update();
68             switch (Globals.ActualPage)
69             {
70                 // Page d'accueil
71                 case Globals.AllPage.Home:
```



```
69         Globals.home.Update();
70         break;
71         // Page de jeu
72         case Globals.AllPage.Game:
73             ParticleManager.Update();
74             Globals.GameManager.Update();
75             break;
76     }
77     base.Update(gameTime);
78 }
79
80 protected override void Draw(GameTime gameTime)
81 {
82     GraphicsDevice.Clear(Color.Black);
83     Globals.SpriteBatch.Begin();
84     switch (Globals.ActualPage)
85     {
86         // Page d'accueil
87         case Globals.AllPage.Home:
88             _jamstikMidiListener.DrawErrorNotConnected();
89             Globals.home.Draw();
90             break;
91         // Page de jeu
92         case Globals.AllPage.Game:
93             Globals.GameManager.Draw();
94             ParticleManager.Draw();
95             break;
96     }
97     Globals.SpriteBatch.End();
98     base.Draw(gameTime);
99 }
100 }
101 }
```

ParticleData.cs

```
1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Graphics;
3 /*
4  * Auteur : Yoann Meier
5  * Date : 06/05/2024
6  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
7  * Description de la page : Struct contenant les variables d'une particule (vient de : ↵
8     https://www.youtube.com/watch?v=-4_kj_gyWRY)
9  */
10 namespace MiFiSy_TPI.ParticleCreator.Structure
11 {
12     internal struct ParticleData
13     {
14         public Texture2D texture = Globals.Content.Load<Texture2D>("particle");
15         public float lifespan = 2f;
16         public Color colorStart = Color.Yellow;
17         public Color colorEnd = Color.Red;
18         public float opacityStart = 1f;
19         public float opacityEnd = 0f;
20         public float sizeStart = 32f;
21         public float sizeEnd = 4f;
22         public float speed = 100f;
23         public float angle = 0f;
24
25         public ParticleData()
26         {
27         }
28     }
```

ParticleEmitterData.cs

```
1  /*
2  * Auteur : Yoann Meier
3  * Date : 06/05/2024
4  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
5  * Description de la page : Struct contenant les variables d'un emitteur de particules ←
6  * (vient de : https://www.youtube.com/watch?v=-4\_kj\_gyWRY)
7  */
8  namespace MiFiSy_TPI.ParticleCreator.Structure
9  {
10     internal struct ParticleEmitterData
11     {
12         public ParticleData particleData = new ParticleData();
13         public float angle = 0f;
14         public float angleVariance = 0f;
15         public float lifespanMin = 0.1f;
16         public float lifespanMax = 2f;
17         public float speedMin = 10f;
18         public float speedMax = 100f;
19         public float interval = 1f;
20         public int emitCount = 1;
21         public bool decreasedLifespan = false;
22         public float nbDecreasedLifespan = 0.05f;
23         public bool randomPosX = false;
24         public float intervalPos = 0.01f;
25         public ParticleEmitterData()
26         {
27         }
28     }
```

Particle.cs

```
1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Graphics;
3 using MiFiSy_TPI.ParticleCreator.Structure;
4 using System;
5 /*
6  * Auteur : Yoann Meier
7  * Date : 06/05/2024
8  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
9  * Description de la page : Class d'une particule (vient de : ↩
10    https://www.youtube.com/watch?v=-4_kj_gyWRY)
11 */
12 namespace MiFiSy_TPI.ParticleCreator
13 {
14     internal class Particle
15     {
16         private ParticleData _data;
17         private Vector2 _position;
18         private float _lifespanLeft;
19         private float _lifespanAmount;
20         private Color _color;
21         private float _opacity;
22         public bool isFinished = false;
23         private float _scale;
24         private Vector2 _origin;
25         private Vector2 _direction;
26
27         public Vector2 Position { get => _position; set => _position = value; }
28         internal ParticleData Data { get => _data; set => _data = value; }
29
30         public Particle(Vector2 pos, ParticleData data)
31         {
32             _data = data;
33             _lifespanLeft = data.lifespan;
34             _lifespanAmount = 1f;
35             _position = pos;
36             _color = data.colorStart;
37             _opacity = data.opacityStart;
38             _origin = new Vector2(_data.texture.Width / 2, _data.texture.Height / 2);
39
40             SetAngleAndDirection();
41
42             /// <summary>
43             /// Calcul la direction de la particule avec l'angle
44             /// </summary>
45             public void SetAngleAndDirection()
46             {
47                 if (_data.speed != 0)
48                 {
49                     // Converti l'angle en radians
50                     _data.angle = MathHelper.ToRadians(_data.angle);
51                     // Calcul la direction grace à l'angle
52                     _direction = new Vector2((float)Math.Sin(_data.angle), ↩
53                                             -(float)Math.Cos(_data.angle));
54                 }
55                 else
56                 {
57                     _direction = Vector2.Zero;
58                 }
59             }
60
61             public void Update()
62             {
63                 _lifespanLeft -= Globals.TotalSeconds;
64                 if (_lifespanLeft <= 0f)
65                 {
66                     isFinished = true;
67                     return;
68                 }
69
70                 // Calcule le temps de vie restant
```

```
70     _lifespanAmount = _lifespanLeft / _data.lifespan;
71
72     // Melange la couleur finale et la couleur initiale en fonction du lifespan
73     _color = Color.Lerp(_data.colorEnd, _data.colorStart, _lifespanAmount);
74
75     // Melange l'opacité finale et l'opacité initiale en fonction du lifespan
76     _opacity = MathHelper.Lerp(_data.opacityEnd, _data.opacityStart, _lifespanAmount);
77
78     // Melange la taille finale et la taille initiale en fonction du lifespan, puis ↵
79     // ajuste l'échelle par rapport à la largeur de la texture.
80     _scale = MathHelper.Lerp(_data.sizeEnd, _data.sizeStart, _lifespanAmount) / ↵
81     _data.texture.Width;
82
83     // Met à jour la position de la particule en fonction de sa direction, de sa ↵
84     // vitesse, du temps écoulé et des dimensions de l'écran.
85     _position.X += _direction.X * _data.speed * Globals.TotalSeconds / ↵
86     Globals.ScreenWidth;
87     _position.Y += _direction.Y * _data.speed * Globals.TotalSeconds / ↵
88     Globals.ScreenHeight;
89 }
90
91 public void Draw()
92 {
93     Globals.SpriteBatch.Draw(_data.texture, new Vector2(_position.X * ↵
94     Globals.ScreenWidth, _position.Y * Globals.ScreenHeight), null, _color * ↵
95     _opacity, 0f, _origin, _scale, SpriteEffects.None, 1f);
96 }
97 }
```

ParticleEmitter.cs

```

1 using Microsoft.Xna.Framework;
2 using MiFiSy_TPI.ParticleCreator.Structure;
3 /*
4  * Auteur : Yoann Meier
5  * Date : 06/05/2024
6  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
7  * Description de la page : Class d'une particule (vient de : ↵
8     https://www.youtube.com/watch?v=-4_kj_gyWRY)
9  */
10 namespace MiFiSy_TPI.ParticleCreator
11 {
12     internal class ParticleEmitter
13     {
14         private ParticleEmitterData data;
15         private float _intervalLeft;
16         private Vector2 _emitPosition;
17         public bool destroy;
18
19         internal ParticleEmitterData Data { get => data; set => data = value; }
20
21         public ParticleEmitter(Vector2 emitPosition, ParticleEmitterData data)
22         {
23             _emitPosition = emitPosition;
24             this.data = data;
25             _intervalLeft = data.interval;
26             destroy = false;
27         }
28
29         /// <summary>
30         /// Émet une nouvelle particule avec une position
31         /// </summary>
32         /// <param name="pos">La position à partir de laquelle émettre la particule</param>
33         public void Emit(Vector2 pos)
34         {
35             ParticleData d = data.particleData;
36             // Random lifespan, speed, angle
37             d.lifespan = Globals.RandomFloat(data.lifespanMin, data.lifespanMax);
38             d.speed = Globals.RandomFloat(data.speedMin, data.speedMax);
39             d.angle = Globals.RandomFloat(data.angle - data.angleVariance, data.angle + ↵
40                 data.angleVariance);
41
42             if (data.randomPosX)
43             {
44                 // Position random X
45                 float xPosition = pos.X * Globals.ScreenWidth;
46                 float randomX = Globals.RandomFloat(xPosition - data.intervalPos * ↵
47                     Globals.ScreenWidth, xPosition + data.intervalPos * Globals.ScreenWidth) ↵
48                     / Globals.ScreenWidth;
49                 pos.X = randomX;
50             }
51             Particle p = new Particle(pos, d);
52             ParticleManager.AddParticle(p);
53         }
54
55         public void Update()
56         {
57             _intervalLeft -= Globals.TotalSeconds;
58             if (_intervalLeft <= 0f)
59             {
60                 // Réinitialise le temps restant
61                 _intervalLeft += data.interval;
62                 // Emet les nouvelles particules
63                 for (int i = 0; i < data.emitCount; i++)
64                 {
65                     Emit(_emitPosition);
66                 }
67
68                 // Diminue le lifespan des prochaines particules
69                 if (data.decreasedLifespan)
70                 {

```

```
67 | data.lifespanMin -= data.nbDecreasedLifespan;  
68 | data.lifespanMax -= data.nbDecreasedLifespan;  
69 |  
70 |     }  
71 | }  
72 | }  
73 | }
```

ParticleManager.cs

```
1 using System;
2 using System.Collections.Generic;
3 /*
4  * Auteur : Yoann Meier
5  * Date : 06/05/2024
6  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
7  * Description de la page : Class d'une particule (vient de : ↩
8  * https://www.youtube.com/watch?v=-4_kj_gyWRY)
9  */
10 namespace MiFiSy_TPI.ParticleCreator
11 {
12     internal class ParticleManager
13     {
14         private static List<Particle> _particles = new List<Particle>();
15         private static List<ParticleEmitter> _particleEmitters = new List<ParticleEmitter>();
16
17         /// <summary>
18         /// Ajoute une particule dans la liste
19         /// </summary>
20         public static void AddParticle(Particle p)
21         {
22             _particles.Add(p);
23         }
24
25         /// <summary>
26         /// Ajoute un émetteur de particules
27         /// </summary>
28         public static void AddParticleEmitter(ParticleEmitter e)
29         {
30             _particleEmitters.Add(e);
31         }
32
33         public static void Update()
34         {
35             // Supprime les particules et émetteur finis
36             _particles.RemoveAll(p => p.isFinished);
37             _particleEmitters.RemoveAll(p => p.destroy);
38
39             try
40             {
41                 _particles.ForEach(p => p.Update());
42                 _particleEmitters.ForEach(e => e.Update());
43             }
44             catch (InvalidOperationException) { /* Il arrive parfois qu'une particule ou é↩
45                 metteur soit ajouté pendant la mise à jour */ }
46
47             /// <summary>
48             /// Supprime une paricule
49             /// </summary>
50             public static void RemoveParticle(Particle p)
51             {
52                 _particles.Remove(p);
53             }
54
55             /// <summary>
56             /// Supprime un émetteur de paricules
57             /// </summary>
58             public static void RemoveParticleEmitter(ParticleEmitter p)
59             {
60                 _particleEmitters.Remove(p);
61             }
62
63             /// <summary>
64             /// Affiche les particules
65             /// </summary>
66             public static void Draw()
67             {
68                 try
69                 {
```



```
69         _particles.ForEach(p => p.Draw());
70     }
71     catch (InvalidOperationException) { /* Il arrive parfois qu'une particule ou é↵
72         metteur soit ajouté pendant l'affichage */ }
73 }
74 /// <summary>
75 /// Supprime toutes les particules et émetteur
76 /// </summary>
77 public static void ClearParticle()
78 {
79     _particleEmitters.Clear();
80     _particles.Clear();
81 }
82 }
83 }
```

Button.cs

```

1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Graphics;
3 using Microsoft.Xna.Framework.Media;
4 using MiFiSy_TPI.Manager;
5 using MiFiSy_TPI.ParticleCreator;
6 using System;
7 /*
8  * Auteur : Yoann Meier
9  * Date : 06/05/2024
10 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
11 * Description de la page : Class d'un bouton
12 */
13 namespace MiFiSy_TPI.UI
14 {
15     internal class Button
16     {
17         private Vector2 _position;
18         private Vector2 _textPosition;
19         private Texture2D _texture;
20         private string _text;
21         private float _widthRectangle;
22         private float _heightRectangle;
23         private Color _backgroundColor;
24         private Color _textColor;
25         private float _padding;
26         private float _scale;
27         private string _action;
28         private bool _isPressed;
29
30         public Rectangle Rectangle { get => new Rectangle((int)(_position.X * ↵
31             Globals.ScreenWidth), (int)(_position.Y * Globals.ScreenHeight), _texture.Width, ↵
32             _texture.Height); }
33         public bool IsPressed { get => _isPressed; set => _isPressed = value; }
34         public string Text { get => _text; set => _text = value; }
35         public Color TextColor { get => _textColor; set => _textColor = value; }
36
37         public Button(Vector2 position, float width, float height, string text, Color ↵
38             backgroundColor, Color textColor, string action, float padding = 0.2f)
39         {
40             _position = position;
41             _widthRectangle = width;
42             _heightRectangle = height;
43             _text = text;
44             _backgroundColor = backgroundColor;
45             _textColor = textColor;
46             _padding = padding;
47             _action = action;
48             _scale = 1;
49             IsPressed = false;
50
51             SetTexture();
52             SetTextPositionAndScale();
53         }
54
55         /// <summary>
56         /// Crée la texture du rectangle du bouton avec ses dimensions et sa couleur
57         /// </summary>
58         public void SetTexture()
59         {
60             int width = (int)(_widthRectangle * Globals.ScreenWidth);
61             int height = (int)(_heightRectangle * Globals.ScreenHeight);
62             _texture = new Texture2D(Globals.GraphicsDevice, width, height);
63             Color[] colorData = new Color[width * height];
64             for (int i = 0; i < colorData.Length; ++i)
65             {
66                 colorData[i] = _backgroundColor;
67             }
68             _texture.SetData(colorData);
69         }
70
71         /// <summary>

```

```

69     /// Calcule la position et la taille du text par rapport à la largeur du rectangle ←
70     qu'il contient
71     /// </summary>
72     public void SetTextPositionAndScale()
73     {
74         if (Globals.FontButton.MeasureString(_text).X != 0)
75         {
76             // Calcul du facteur d'échelle pour le texte
77             float scaleX = (Rectangle.Width * (1 - _padding)) / ←
78                 Globals.FontButton.MeasureString(_text).X;
79             float scaleY = (Rectangle.Height * (1 - _padding)) / ←
80                 Globals.FontButton.MeasureString(_text).Y;
81             _scale = Math.Min(scaleX, scaleY);
82         }
83         else
84         {
85             _scale = 1;
86         }
87
88         _textPosition.X = Rectangle.X + (Rectangle.Width - ←
89             Globals.FontButton.MeasureString(_text).X * _scale) / 2;
90         _textPosition.Y = Rectangle.Y + (Rectangle.Height - ←
91             Globals.FontButton.MeasureString(_text).Y * _scale) / 2;
92     }
93     public void Update()
94     {
95         if (InputManager.HasClicked && Rectangle.Contains(InputManager.MousePosition))
96         {
97             switch (_action)
98             {
99                 case "goBack":
100                     // Retour à l'accueil
101                     MediaPlayer.Stop();
102                     Globals.LstFirework.Clear();
103                     ParticleManager.ClearParticle();
104                     Globals.MusicSelectedName = "";
105                     Globals.home = new Home();
106                     Globals.ActualPage = Globals.AllPage.Home;
107                     break;
108                 case "playReplay":
109                     Globals.LstFirework.Clear();
110                     ParticleManager.ClearParticle();
111                     IsPressed = true;
112                     break;
113                 case "addMusic":
114                 case "save":
115                     IsPressed = true;
116                     break;
117                 case "play":
118                     Globals.GameManager = new GameManager(true, ←
119                         Globals.MusicSelectedName);
120                     Globals.ActualPage = Globals.AllPage.Game;
121                     break;
122                 default:
123                     break;
124             }
125         }
126         else
127         {
128             IsPressed = false;
129         }
130     }
131     /// <summary>
132     /// Affiche le bouton
133     /// </summary>
134     public void Draw()
135     {
136         Globals.SpriteBatch.Draw(_texture, Rectangle, Color.White);
137         Globals.SpriteBatch.DrawString(Globals.FontButton, _text, _textPosition, ←
138             _textColor, 0f, Vector2.Zero, _scale, SpriteEffects.None, 0f);
139     }
140 }

```

Mortar.cs

```

1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Graphics;
3 using System;
4 /*
5  * Auteur : Yoann Meier
6  * Date : 06/05/2024
7  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
8  * Description de la page : Class d'un mortier
9  */
10 namespace MiFiSy_TPI.UI
11 {
12     internal class Mortar
13     {
14         private Vector2 _position;
15         private Color _color;
16         private float _width;
17         private float _height;
18         private Texture2D _texture;
19         private float _angle;
20
21         public Rectangle Rectangle { get => new Rectangle((int)(Position.X * ↵
22             Globals.ScreenWidth), (int)(Position.Y * Globals.ScreenHeight), _texture.Width, ↵
23             _texture.Height); }
24         public Vector2 Position { get => _position; set => _position = value; }
25         public float Width { get => _width; set => _width = value; }
26         public float Angle { get => _angle; set => _angle = value; }
27         public float Height { get => _height; set => _height = value; }
28
29         public Mortar(Vector2 position, float width, float height, float angle, Color color)
30         {
31             Position = position;
32             _color = color;
33             Width = width;
34             Height = height;
35             Angle = Globals.RandomFloat(-angle, angle);
36
37             // Converti l'angle en radians
38             if (Angle >= 0)
39             {
40                 Angle = MathHelper.ToRadians(Angle);
41             }
42             else
43             {
44                 Angle = -MathHelper.ToRadians(Math.Abs(Angle));
45             }
46             SetTexture();
47         }
48
49         /// <summary>
50         /// Crée la texture du rectangle du bouton avec ses dimensions et sa couleur
51         /// </summary>
52         public void SetTexture()
53         {
54             int width = (int)(Width * Globals.ScreenWidth);
55             int height = (int)(Height * Globals.ScreenHeight);
56             _texture = new Texture2D(Globals.GraphicsDevice, width, height);
57             Color[] colorData = new Color[width * height];
58             for (int i = 0; i < colorData.Length; ++i)
59             {
60                 colorData[i] = _color;
61             }
62             _texture.SetData(colorData);
63         }
64
65         /// <summary>
66         /// Affiche le mortier
67         /// </summary>
68         public void Draw()
69         {
70             Globals.SpriteBatch.Draw(_texture, Rectangle, null, _color, Angle, ↵
71                 Vector2.Zero, SpriteEffects.None, 0);
72         }
73     }
74 }

```

```
69 |         }  
70 |     }  
71 | }
```

IFirework.cs

```
1 using Microsoft.Xna.Framework;
2 /*
3  * Auteur : Yoann Meier
4  * Date : 06/05/2024
5  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
6  * Description de la page : Interface des particules pour n'avoir qu'une seule liste
7  */
8 namespace MiFiSy_TPI.Firework
9 {
10     public interface IFirework
11     {
12         /// <summary>
13         /// position du feu d'artifice au départ
14         /// </summary>
15         Vector2 StartPosition { get; set; }
16
17         /// <summary>
18         /// durée de vie du feu d'artifice
19         /// </summary>
20         float Lifespan { get; set; }
21
22         /// <summary>
23         /// Temps après le début du mode libre où ce feu d'artifice est créé
24         /// </summary>
25         float LaunchTime { get; set; }
26
27         /// <summary>
28         /// Vitesse de départ
29         /// </summary>
30         float StartSpeed { get; set; }
31
32         /// <summary>
33         /// Méthode update pour supprimer les anciennes particules
34         /// </summary>
35         void Update();
36     }
37 }
```

ParticleRain.cs

```

1 using Microsoft.Xna.Framework;
2 using MiFiSy_TPI.ParticleCreator;
3 using MiFiSy_TPI.ParticleCreator.Structure;
4 using System;
5 using System.Collections.Generic;
6 /*
7  * Auteur : Yoann Meier
8  * Date : 06/05/2024
9  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
10 * Description de la page : Class d'une pluie de particule, version 2
11 */
12 namespace MiFiSy_TPI.Firework
13 {
14     public class ParticleRain : IFirework
15     {
16         private List<Particle> _lstMainParticles;
17         private float _lifespan;
18         private float _timerLife;
19         private float _timerSpawn;
20         private float _launchTime;
21         private Vector2 _startPosition;
22         private float _startSpeed;
23         private float _nbParticle;
24         private Color _colorStart;
25         private Color _colorEnd;
26         private float _size;
27
28         public float Lifespan { get => _lifespan; set => _lifespan = value; }
29         public float LaunchTime { get => _launchTime; set => _launchTime = value; }
30         public Vector2 StartPosition { get => _startPosition; set => _startPosition = ←
            value; }
31         public float StartSpeed { get => _startSpeed; set => _startSpeed = value; }
32
33         /// <summary>
34         /// Constructeur de la classe utilisée dans le jeu libre : créer la pluie de ←
35         /// particule en fonction de paramètre du fichier de configuration
36         /// </summary>
37         /// <param name="speed">vitesse de départ du feu d'artifice</param>
38         /// <param name="lifespan">durée de vie du feu d'artifice</param>
39         /// <param name="launchTime">Le temps à laquelle l'effet a été crée, seulement ←
40         /// utilisé pour la sauvegarde</param>
41         /// <param name="distanceFromBorder">distance pour ne pas créer la particule en ←
42         /// dehors ou sur le bord de l'écran</param>
43         public ParticleRain(float speed, float lifespan, float launchTime, float ←
            distanceFromBorder = 100)
44         {
45             LaunchTime = launchTime;
46             Lifespan = lifespan;
47             StartSpeed = speed;
48             _nbParticle = Config.PARTICLE_RAIN_NB;
49             _colorStart = Config.COLOR_PARTICLE_RAIN_START;
50             _colorEnd = Config.COLOR_PARTICLE_RAIN_END;
51             _size = Config.PARTICLE_RAIN_SIZE;
52
53             _timerLife = 0;
54             _timerSpawn = 0;
55             // Position aléatoire du feu d'artifice sur la partie haute de l'écran
56             StartPosition = new Vector2(Globals.RandomFloat(distanceFromBorder, ←
                Globals.ScreenWidth - distanceFromBorder) / Globals.ScreenWidth, ←
                Globals.RandomFloat(distanceFromBorder, Globals.ScreenHeight / 2) / ←
                Globals.ScreenHeight);
57             _lstMainParticles = new List<Particle>();
58
59             for (int i = 0; i < _nbParticle; i++)
60             {
61                 float angle = 360 / _nbParticle * i;
62                 // Vitesse aléatoire entre 0 et le maximum
63                 float newSpeed = Globals.RandomFloat(0, speed);
64                 ParticleData particleData = new ParticleData()
65                 {
66                     angle = angle,

```

```

64         speed = newSpeed,
65         colorStart = _colorStart,
66         colorEnd = _colorEnd,
67         sizeStart = _size,
68         sizeEnd = _size,
69         lifespan = Lifespan,
70     };
71     Particle p = new Particle(StartPosition, particleData);
72     _lstMainParticles.Add(p);
73     ParticleManager.AddParticle(p);
74 }
75
76 /// <summary>
77 /// Constructeur de la classe utilisée dans le jeu replay : créer la pluie de ←
78 /// particules en fonction de paramètre du fichier qui est rejoué
79 /// </summary>
80 /// <param name="position">position de départ</param>
81 /// <param name="speed">vitesse de départ du feu d'artifice</param>
82 /// <param name="lifespan">durée de vie du feu d'artifice</param>
83 /// <param name="colorStart">couleur de départ</param>
84 /// <param name="colorEnd">couleur de fin</param>
85 /// <param name="size">taille des particules</param>
86 /// <param name="nbParticle">nombre de particule a générer</param>
87 public ParticleRain(Vector2 position, float speed, float lifespan, Color ←
88     colorStart, Color colorEnd, float size, float nbParticle)
89 {
90     LaunchTime = 0f;
91     Lifespan = lifespan;
92     StartSpeed = speed;
93     _nbParticle = nbParticle;
94     _colorStart = colorStart;
95     _colorEnd = colorEnd;
96     _size = size;
97
98     _timerLife = 0;
99     _timerSpawn = 0;
100     StartPosition = position;
101
102     _lstMainParticles = new List<Particle>();
103
104     for (int i = 0; i < _nbParticle; i++)
105     {
106         float angle = 360 / _nbParticle * i;
107         // Vitesse aléatoire entre 0 et le maximum
108         float newSpeed = Globals.RandomFloat(0, speed);
109         ParticleData particleData = new ParticleData()
110         {
111             angle = angle,
112             speed = newSpeed,
113             colorStart = _colorStart,
114             colorEnd = _colorEnd,
115             sizeStart = _size,
116             sizeEnd = _size,
117             lifespan = lifespan,
118         };
119         Particle p = new Particle(StartPosition, particleData);
120         _lstMainParticles.Add(p);
121         ParticleManager.AddParticle(p);
122     }
123
124     /// <summary>
125     /// Méthode qui fais apparaître de nouvelles particules si la durée de vie n'est ←
126     /// pas atteinte, sinon supprime les particules
127     /// </summary>
128     public void Update()
129     {
130         _timerLife += Globals.TotalSeconds;
131         _timerSpawn += Globals.TotalSeconds;
132         // Supprime en fin de vie
133         if (_timerLife >= Lifespan)
134         {
135             _lstMainParticles.Clear();
136         }
137
138         if (_lstMainParticles.Count != 0)
139         {
140             if (_timerSpawn >= Config.PARTICLE_RAIN_TIME_SPAWN)
141             {
142                 // Ajoute une particule immobile sur chaque particule en mouvement
143                 for (int i = 0; i < _nbParticle; i++)
144                 {
145                     ParticleData particleData = new ParticleData()
146                     {
147                         angle = MathHelper.ToDegrees(_lstMainParticles[i].Data.angle),
148                         speed = 0,
149                         colorStart = _colorStart,

```



```
148         colorEnd = _colorEnd,
149         sizeStart = _size,
150         sizeEnd = _size,
151         lifespan = Lifespan - _timerLife,
152     };
153     Particle p = new Particle(_lstMainParticles[i].Position, ←
        particleData);
154     ParticleManager.AddParticle(p);
155 }
156 _timerSpawn = 0;
157 }
158
159 // Si un tiers du temps total est passé, les particules en mouvement tombent
160 if (_timerLife >= Lifespan / 3)
161 {
162     foreach (Particle item in _lstMainParticles)
163     {
164         ParticleData data = item.Data;
165         int angleAdd = MathHelper.ToDegrees(data.angle) < 180 ? 1 : -1;
166         data.angle = MathHelper.ToDegrees(data.angle) + angleAdd;
167         item.Data = data;
168         item.SetAngleAndDirection();
169     }
170 }
171 }
172 }
173 }
174 }
```

Comet.cs

```

1 using Microsoft.Xna.Framework;
2 using MiFiSy_TPI.ParticleCreator;
3 using MiFiSy_TPI.ParticleCreator.Structure;
4 /*
5  * Auteur : Yoann Meier
6  * Date : 06/05/2024
7  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
8  * Description de la page : Classe de création du feu d'artifice de la comète
9  */
10 namespace MiFiSy_TPI.Firework
11 {
12     public class Comet : IFirework
13     {
14         private Particle _mainParticle;
15         private ParticleEmitter _emitter;
16         private float _lifespan;
17         private float _timerLife;
18         private float _launchTime;
19         private Vector2 _startPosition;
20         private float _startAngle;
21         private float _startSpeed;
22
23         public float LaunchTime { get => _launchTime; set => _launchTime = value; }
24         public Vector2 StartPosition { get => _startPosition; set => _startPosition = ←
            value; }
25         internal Particle MainParticle { get => _mainParticle; set => _mainParticle = ←
            value; }
26         public float StartAngle { get => _startAngle; set => _startAngle = value; }
27         public float StartSpeed { get => _startSpeed; set => _startSpeed = value; }
28         public float Lifespan { get => _lifespan; set => _lifespan = value; }
29
30
31         /// <summary>
32         /// Constructeur de la classe utilisée dans le jeu libre : créer la comète en ←
            fonction de paramètre du fichier de configuration
33         /// </summary>
34         /// <param name="position">Position de départ de la comète</param>
35         /// <param name="angle">angle de la comète</param>
36         /// <param name="speed">vitesse de la comète</param>
37         /// <param name="lifespan">durée de vie de la comète</param>
38         /// <param name="launchTime">Le temps à laquelle l'effet a été crée, seulement ←
            utilisé pour la sauvegarde</param>
39         public Comet(Vector2 position, float angle, float speed, float lifespan, float ←
            launchTime)
40         {
41             LaunchTime = launchTime;
42             StartPosition = position;
43             StartAngle = MathHelper.ToDegrees(angle);
44             StartSpeed = speed;
45             Lifespan = lifespan;
46             _timerLife = 0;
47
48             // Créer la particule principale, la tête
49             ParticleData particleData = new ParticleData()
50             {
51                 angle = StartAngle,
52                 speed = StartSpeed,
53                 lifespan = Lifespan,
54                 colorStart = Config.COLOR_START_COMET,
55                 colorEnd = Config.COLOR_END_COMET,
56                 sizeStart = Config.COMET_MAIN_SIZE,
57                 sizeEnd = Config.COMET_MAIN_SIZE,
58             };
59             _mainParticle = new Particle(position, particleData);
60             ParticleManager.AddParticle(_mainParticle);
61
62             // créer l'émetteur qui suit la tête, la queue
63             ParticleEmitterData ped = new ParticleEmitterData()
64             {
65                 interval = 0.01f,
66                 emitCount = 5,

```

```

67         lifespanMin = Lifespan,
68         lifespanMax = Lifespan,
69         angle = StartAngle,
70         randomPosX = true,
71         intervalPos = 0.003f,
72         decreasedLifespan = true,
73         nbDecreasedLifespan = 0.05f,
74         speedMin = StartSpeed,
75         speedMax = StartSpeed,
76         particleData = new ParticleData()
77     {
78         colorStart = Config.COLOR_START_COMET,
79         colorEnd = Config.COLOR_END_COMET,
80         sizeStart = Config.COMET_OTHER_SIZE,
81         sizeEnd = Config.COMET_OTHER_SIZE,
82     }
83 };
84 _emitter = new ParticleEmitter(_mainParticle.Position, ped);
85 ParticleManager.AddParticleEmitter(_emitter);
86 }
87
88 /// <summary>
89 /// Constructeur de la classe utilisée dans le jeu replay : créer la comète en ←
90 /// fonction de paramètre du fichier qui est rejoué
91 /// </summary>
92 /// <param name="position">Position de départ de la comète</param>
93 /// <param name="angle">angle de la comète</param>
94 /// <param name="speed">vitesse de la comète</param>
95 /// <param name="lifespan">durée de vie de la comète</param>
96 /// <param name="colorStart">couleur de départ de la comète</param>
97 /// <param name="colorEnd">couleur de fin de la comète</param>
98 /// <param name="mainSize">taille des particules de la tête</param>
99 /// <param name="otherSize">taille des particules de la queue</param>
100 public Comet(Vector2 position, float angle, float speed, float lifespan, Color ←
101     colorStart, Color colorEnd, float mainSize, float otherSize)
102 {
103     LaunchTime = 0f;
104     StartPosition = position;
105     StartAngle = angle;
106     StartSpeed = speed;
107     Lifespan = lifespan;
108     _timerLife = 0;
109
110     // Créer la particule principale, la tête
111     ParticleData particleData = new ParticleData()
112     {
113         angle = StartAngle,
114         speed = StartSpeed,
115         lifespan = Lifespan,
116         colorStart = colorStart,
117         colorEnd = colorEnd,
118         sizeStart = mainSize,
119         sizeEnd = mainSize,
120     };
121     _mainParticle = new Particle(position, particleData);
122     ParticleManager.AddParticle(_mainParticle);
123
124     // créer l'émetteur qui suit la tête, la queue
125     ParticleEmitterData ped = new ParticleEmitterData()
126     {
127         interval = 0.01f,
128         emitCount = 5,
129         lifespanMin = Lifespan,
130         lifespanMax = Lifespan,
131         angle = StartAngle,
132         randomPosX = true,
133         intervalPos = 0.003f,
134         decreasedLifespan = true,
135         nbDecreasedLifespan = 0.05f,
136         speedMin = StartSpeed,
137         speedMax = StartSpeed,
138         particleData = new ParticleData()
139         {
140             colorStart = colorStart,
141             colorEnd = colorEnd,
142             sizeStart = otherSize,
143             sizeEnd = otherSize,
144         }
145     };
146     _emitter = new ParticleEmitter(_mainParticle.Position, ped);
147     ParticleManager.AddParticleEmitter(_emitter);
148 }
149
150 public void Update()
151 {
152     _timerLife += Globals.TotalSeconds;
153     if (_timerLife >= Lifespan)

```

```
152         {  
153             ParticleManager.RemoveParticleEmitter(_emitter);  
154             ParticleManager.RemoveParticle(_mainParticle);  
155         }  
156     }  
157 }  
158 }
```

JamstikMidiListener.cs

```

1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Graphics;
3 using NAudio.Midi;
4 /*
5  * Auteur : Yoann Meier
6  * Date : 06/05/2024
7  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
8  * Description de la page : Classe pour gérer les entrées MIDI
9  */
10 namespace MiFiSy_TPI.Manager
11 {
12     internal class JamstikMidiListener
13     {
14         private MidiIn _midi;
15         private bool _isConnected;
16         private SpriteFont _font;
17
18         /// <summary>
19         /// Constructeur de la classe, connection à la guitare
20         /// </summary>
21         /// <param name="font">font pour le message d'erreur si la guitare n'est pas ↵
22         /// trouvé</param>
23         public JamstikMidiListener(SpriteFont font)
24         {
25             _font = font;
26             _isConnected = false;
27
28             if (MidiIn.NumberOfDevices != 0)
29             {
30                 for (int i = 0; i < MidiIn.NumberOfDevices; i++)
31                 {
32                     MidiInCapabilities capabilities = MidiIn.DeviceInfo(i);
33                     // Connexion au Jamstik
34                     if (capabilities.ProductName == "Jamstik")
35                     {
36                         _midi = new MidiIn(i);
37                         _midi.MessageReceived += MidiIn_MessageReceived;
38                         _midi.Start();
39                         _isConnected = true;
40                         break;
41                     }
42                 }
43             }
44
45             /// <summary>
46             /// Méthode appelé évènements MIDI, traite seulement les évènements de notes, on ou ↵
47             /// off
48             /// </summary>
49             private void MidiIn_MessageReceived(object sender, MidiInMessageEventArgs e)
50             {
51                 // Si on est dans le jeu en mode libre
52                 if (Globals.ActualPage == Globals.AllPage.Game && Globals.GameManager.Mode)
53                 {
54                     MidiEvent midiEvent = MidiEvent.FromRawMessage(e.RawMessage);
55                     if (midiEvent is NoteEvent noteEvent)
56                     {
57                         // Lorsqu'une note est jouée
58                         if (noteEvent.CommandCode == MidiCommandCode.NoteOn)
59                         {
60                             // Corde 1 jouée
61                             if (noteEvent.Channel - 1 == 1)
62                             {
63                                 Globals.GameManager.CreateComete(noteEvent.Velocity);
64                             }
65                             // Corde 2 jouée
66                             else if (noteEvent.Channel - 1 == 2)
67                             {
68                                 Globals.GameManager.CreateParticleRain(noteEvent.Velocity);
69                             }
70                         }
71                     }
72                 }
73             }
74         }
75     }
76 }

```

```
70     }
71   }
72 }
73
74 /// <summary>
75 /// Affiche un message d'erreur si il n'y a pas de connexion
76 /// </summary>
77 public void DrawErrorNotConnected()
78 {
79     if (!_isConnected)
80     {
81         Globals.SpriteBatch.DrawString(_font, "Aucune entree MIDI trouvee", new Vector2(Globals.ScreenWidth / 2, Globals.ScreenHeight / 2), Color.Red);
82     }
83 }
84 }
85 }
```

InputManager

```
1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Input;
3 /*
4  * Auteur : Yoann Meier
5  * Date : 06/05/2024
6  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
7  * Description de la page : Classe pour gérer la souris
8  */
9 namespace MiFiSy_TPI.Manager
10 {
11     internal class InputManager
12     {
13         private static MouseState _lastMouseState;
14         private static bool _hasSpaceKeyUp;
15         private static bool _hasEnterKeyUp;
16
17         public static bool HasClicked { get; private set; }
18         public static Vector2 MousePosition { get; private set; }
19         public static bool IsKeyParticleRainPressed { get; private set; }
20         public static bool IsKeyCometPressed { get; private set; }
21
22         public static void Update()
23         {
24             var mouseState = Mouse.GetState();
25
26             HasClicked = mouseState.LeftButton == ButtonState.Pressed && ←
27                 _lastMouseState.LeftButton == ButtonState.Released;
28             MousePosition = mouseState.Position.ToVector2();
29
30             _lastMouseState = mouseState;
31
32             if (Keyboard.GetState().IsKeyUp(Keys.Space))
33             {
34                 _hasSpaceKeyUp = true;
35             }
36
37             if (Keyboard.GetState().IsKeyDown(Keys.Space) && _hasSpaceKeyUp)
38             {
39                 IsKeyParticleRainPressed = true;
40                 _hasSpaceKeyUp = false;
41             }
42             else
43             {
44                 IsKeyParticleRainPressed = false;
45             }
46
47             if (Keyboard.GetState().IsKeyUp(Keys.Enter))
48             {
49                 _hasEnterKeyUp = true;
50             }
51
52             if (Keyboard.GetState().IsKeyDown(Keys.Enter) && _hasEnterKeyUp)
53             {
54                 IsKeyCometPressed = true;
55                 _hasEnterKeyUp = false;
56             }
57             else
58             {
59                 IsKeyCometPressed = false;
60             }
61         }
62     }
63 }
```

Home.cs

```
1 using Microsoft.Xna.Framework;
2 using MiFiSy_TPI.UI;
3 using System;
4 using System.Collections.Generic;
5 using System.IO;
6 using System.Linq;
7 using System.Xml.Linq;
8 /*
9  * Auteur : Yoann Meier
10 * Date : 06/05/2024
11 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
12 * Description de la page : Classe de l'accueil
13 */
14 namespace MiFiSy_TPI.Manager
15 {
16     public class Home
17     {
18         private Button _btnPlay;
19         private Dictionary<string, Button> _lstReplay;
20         private List<Button> _lstBtnMusic;
21
22         private const int NB_FILE_MAX = 10;
23
24         /// <summary>
25         /// Constructeur de la classe, récupère les musiques, les séquences sauvegardés des ↵
26         /// dossiers défini dans le fichier de configuration
27         /// </summary>
28         public Home()
29         {
30             _btnPlay = new Button(new Vector2(0.3f, 0.5f), 0.17f, 0.05f, "Commencer en mode ↵
31             libre", Color.Gray, Color.White, "play");
32             _lstReplay = new Dictionary<string, Button>();
33             _lstBtnMusic = new List<Button>();
34
35             if (Directory.Exists(Config.PATH_SAVE_SEQUENCE))
36             {
37                 string[] allReplay = Directory.GetFiles(Config.PATH_SAVE_SEQUENCE);
38                 if (allReplay.Length > NB_FILE_MAX)
39                 {
40                     Array.Resize(ref allReplay, NB_FILE_MAX);
41                 }
42                 for (int i = 0; i < allReplay.Length; i++)
43                 {
44                     string fileType = allReplay[i].Split(".")[1];
45                     if (fileType == "xml")
46                     {
47                         string nameSequence = ↵
48                         XDocument.Load(allReplay[i]).Descendants("FireworkSequence").
49                         FirstOrDefault().Attribute("name").Value;
50                         _lstReplay.Add(allReplay[i], new Button(new Vector2(0.8f, ↵
51                         Globals.ScreenHeight / (float)(allReplay.Length + 1) * (i + 1) / ↵
52                         Globals.ScreenHeight, 0.1f, 0.05f, nameSequence, Color.Gray, ↵
53                         Color.White, "playReplay"));
54                     }
55                 }
56             }
57
58             if (Directory.Exists(Config.PATH_MUSIC))
59             {
60                 string[] allMusic = Directory.GetFiles(Config.PATH_MUSIC);
61                 if (allMusic.Length > NB_FILE_MAX)
62                 {
63                     Array.Resize(ref allMusic, NB_FILE_MAX);
64                 }
65                 for (int i = 0; i < allMusic.Length; i++)
66                 {
67                     string fileType = allMusic[i].Split(".")[1];
68                     if (fileType == "mp3" || fileType == "wav")
69                     {
70                         // ... (code continues)
71                     }
72                 }
73             }
74         }
75     }
76 }
```



```

66         string fileName = allMusic[i].Split('/')[1];
67         _lstBtnMusic.Add(new Button(new Vector2(0.1f, Globals.ScreenHeight ←
        / (float)(allMusic.Length + 1) * (i + 1) / ←
        Globals.ScreenHeight), 0.1f, 0.05f, fileName, Color.Gray, ←
        Color.White, "addMusic"));
68     }
69 }
70 }
71 }
72
73 public void Update()
74 {
75     _btnPlay.Update();
76     foreach (Button btnMusic in _lstBtnMusic)
77     {
78         btnMusic.Update();
79         if (btnMusic.IsPressed)
80         {
81             bool changeOk = true;
82             // Si aucune musique n'est sélectionné
83             if (Globals.MusicSelectedName != "")
84             {
85                 Button btnSelected = _lstBtnMusic.Find(x => x.Text == ←
                    Globals.MusicSelectedName);
86                 btnSelected.TextColor = Color.White;
87                 Globals.MusicSelectedName = "";
88                 // Si on a r'appuyé sur le même bouton, on ne le met pas en rouge
89                 if (btnSelected.Text == btnMusic.Text)
90                 {
91                     changeOk = false;
92                 }
93             }
94
95             // Change la couleur en rouge si c'est une nouvelle musique qui est ←
            sélectionné
96             if (changeOk)
97             {
98                 Globals.MusicSelectedName = btnMusic.Text;
99                 btnMusic.TextColor = Color.Red;
100             }
101         }
102     }
103
104     for (int i = 0; i < _lstReplay.Count; i++)
105     {
106         Button btn = _lstReplay.ElementAt(i).Value;
107         btn.Update();
108         if (btn.IsPressed)
109         {
110             Globals.GameManager = new GameManager(false, "", ←
                _lstReplay.ElementAt(i).Key);
111             Globals.ActualPage = Globals.AllPage.Game;
112         }
113     }
114 }
115
116 /// <summary>
117 /// Affiche tous les éléments de l'accueil
118 /// </summary>
119 public void Draw()
120 {
121     _btnPlay.Draw();
122     if (_lstBtnMusic.Count != 0)
123     {
124         Globals.SpriteBatch.DrawString(Globals.FontButton, "Choisir une musique : ←
            (optionnel)", new Vector2(_lstBtnMusic[0].Rectangle.X, ←
            _lstBtnMusic[0].Rectangle.Y - 50), Color.White);
125     }
126     _lstBtnMusic.ForEach(x => x.Draw());
127     if (_lstReplay.Count != 0)
128     {
129         Globals.SpriteBatch.DrawString(Globals.FontButton, "Revoir :", new ←
            Vector2(_lstReplay.ElementAt(0).Value.Rectangle.X, ←
            _lstReplay.ElementAt(0).Value.Rectangle.Y - 50), Color.White);
130     }
131     foreach (Button btn in _lstReplay.Values)
132     {
133         btn.Draw();
134     }
135 }
136 }
137 }

```

GameManager.cs

```
1 using Microsoft.Xna.Framework;
2 using Microsoft.Xna.Framework.Graphics;
3 using Microsoft.Xna.Framework.Input;
4 using Microsoft.Xna.Framework.Media;
5 using MiFiSy_TPI.Firework;
6 using MiFiSy_TPI.UI;
7 using System;
8 using System.Collections.Generic;
9 using System.IO;
10 using System.Linq;
11 using System.Xml.Linq;
12 /*
13  * Auteur : Yoann Meier
14  * Date : 06/05/2024
15  * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
16  * Description de la page : Classe de gestion du jeu libre et replay
17 */
18 namespace MiFiSy_TPI.Manager
19 {
20     internal class GameManager
21     {
22         private bool _mode;
23         private Button _menuButton;
24         private Button _saveButton;
25         private List<Mortar> _lstMortar;
26         private float _timerLaunch;
27         private Song _music;
28         private Texture2D _background;
29         private XElement _file;
30
31         // Message après sauvegarde
32         private float _timerSave;
33         private bool showMessageSave;
34
35         private const float TIME_MESSAGE_SAVE = 2f;
36
37         // Constantes des noms d'éléments et d'attributs pour la sauvegarde et la ←
38         // récupération des données lors du replay
39         private const string ATTRIBUTE_NAME = "name";
40         private const string ATTRIBUTE_CREATION_DATE = "creationDate";
41         private const string ATTRIBUTE_AUTHOR = "author";
42         private const string ATTRIBUTE_TIME_END = "timeEnd";
43         private const string ATTRIBUTE_POSITION_X = "positionX";
44         private const string ATTRIBUTE_POSITION_Y = "positionY";
45         private const string ATTRIBUTE_WIDTH = "width";
46         private const string ATTRIBUTE_HEIGHT = "height";
47         private const string ATTRIBUTE_ANGLE = "angle";
48         private const string ATTRIBUTE_SPEED = "speed";
49         private const string ATTRIBUTE_LIFESPAN = "lifeSpan";
50         private const string ATTRIBUTE_NB_PARTICLE = "nbParticle";
51         private const string ATTRIBUTE_MAIN_SIZE = "main";
52         private const string ATTRIBUTE_OTHER_SIZE = "other";
53         private const string ELEMENT_MORTAR = "mortar";
54         private const string ELEMENT_START = "start";
55         private const string ELEMENT_SIZE = "Size";
56         private const string ELEMENT_AUDIO = "Audio";
57         private const string ELEMENT_BACKGROUND = "Background";
58         private const string ELEMENT_FIREWORK_SEQUENCE = "FireworkSequence";
59         private const string ELEMENT_FIREWORK = "Firework";
60         private const string ELEMENT_COLOR_START = "ColorStart";
61         private const string ELEMENT_COLOR_END = "ColorEnd";
62         private const string ATTRIBUTE_R_COLOR = "r";
63         private const string ATTRIBUTE_G_COLOR = "g";
64         private const string ATTRIBUTE_B_COLOR = "b";
65         private const string ATTRIBUTE_TRACK = "track";
66         private const string ATTRIBUTE_IMG = "img";
67         private const string ATTRIBUTE_LAUNCH_TIME = "launchTime";
68         private const string ATTRIBUTE_TYPE_COMET = "Comet";
69         private const string ATTRIBUTE_TYPE_PARTICLE_RAIN = "ParticleRain";
70         private const string ATTRIBUTE_TYPE = "type";
```

```

70
71     public bool Mode { get => _mode; set => _mode = value; }
72
73     /// <summary>
74     /// Constructeur de la classe
75     /// </summary>
76     /// <param name="mode">Si mode = true, on est dans le mode libre, si mode = false, ↵
77     /// on est dans le mode replay</param>
78     /// <param name="musiqueName">nom de la musique, optionnel</param>
79     /// <param name="replayFileName">nom du replay, obligatoire dans le mode ↵
80     /// replay</param>
81     public GameManager(bool mode, string musiqueName = "", string replayFileName = "")
82     {
83         Mode = mode;
84         _lstMortar = new List<Mortar>();
85         _timerLauch = 0;
86         _timerSave = 0;
87         showMessageSave = false;
88
89         _menuButton = new Button(new Vector2(0.01f, 0.01f), 0.1f, 0.05f, "Accueil", ↵
90         Color.Gray, Color.White, "goBack");
91
92         if (Mode)
93         {
94             _saveButton = new Button(new Vector2(0.89f, 0.01f), 0.1f, 0.05f, ↵
95             "Sauvegarder", Color.Gray, Color.White, "save");
96
97             // Charge et lance la musique si une musique a été choisi
98             if (musiqueName != "")
99             {
100                 try
101                 {
102                     string fullPath = ↵
103                     Path.Combine(AppDomain.CurrentDomain.BaseDirectory, ↵
104                     Config.PATH_MUSIC, musiqueName);
105                     _music = Song.FromUri(Path.GetFileName(fullPath), new Uri(fullPath));
106                     MediaPlayer.Play(_music);
107                 }
108                 catch { /* le fichier n'existe plus ou ce n'est pas une musique */ }
109             }
110
111             // Charge l'image si un chemin est indiqué dans le fichier de configuration
112             if (Config.PATH_IMG != "")
113             {
114                 try
115                 {
116                     _background = Texture2D.FromFile(Globals.GraphicsDevice, ↵
117                     Config.PATH_IMG);
118                 }
119                 catch { /* Le fichier n'existe pas ou n'est pas un format image */ }
120             }
121
122             // Ajoute tous les mortiers spécifiés dans le fichier de configuration
123             if (Config.ALL_MORTAR.Count != 0)
124             {
125                 foreach (XElement mortar in Config.ALL_MORTAR)
126                 {
127                     AddMortarFromXElementToListMortar(mortar);
128                 }
129             }
130             // Sinon, ajoute 5 mortiers par défaut
131             else
132             {
133                 for (int i = 1; i <= 5; i++)
134                 {
135                     // (float)(5 + 1) : le float sert à ne pas arrondir à 0
136                     _lstMortar.Add(new Mortar(new Vector2(Globals.ScreenWidth / ↵
137                     (float)(5 + 1) * i / Globals.ScreenWidth, 1 - 0.15f), 0.025f, ↵
138                     0.15f, 10, Color.White));
139                 }
140             }
141         }
142         else
143         {
144             // Charge le fichier pour le rejouer
145             _file = XDocument.Load(replayFileName).Descendants(ELEMENT_FIREWORK_SEQUENCE).
146             FirstOrDefault();
147
148             // Créer tous les mortiers
149             foreach (XElement mortar in _file.Descendants(ELEMENT_MORTAR))
150             {
151                 AddMortarFromXElementToListMortar(mortar);
152             }
153
154             // Charge l'image si un chemin est indiqué dans le fichier de la séquence
155             if (_file.Descendants(ELEMENT_BACKGROUND).Attributes(ATTRIBUTE_IMG).
156             FirstOrDefault().Value != "")

```

```

148         {
149             try
150             {
151                 _background = Texture2D.FromFile(Globals.GraphicsDevice, ←
                    _file.Descendants(ELEMENT_BACKGROUND).Attributes(ATTRIBUTE_IMG).
                    FirstOrDefault().Value);
152             }
153             catch { /* Le fichier n'existe pas ou n'est pas un format image */ }
154         }
155     }
156
157     // Charge et lance la musique si elle est indiqué dans le fichier de la ←
    séquence
158     if (_file.Descendants(ELEMENT_AUDIO).Attributes(ATTRIBUTE_TRACK).
159         FirstOrDefault().Value != "")
160     {
161         try
162         {
163             string fullPath = ←
                Path.Combine(AppDomain.CurrentDomain.BaseDirectory, ←
                    _file.Descendants(ELEMENT_AUDIO).Attributes(ATTRIBUTE_TRACK).
                    FirstOrDefault().Value);
164             _music = Song.FromUri(Path.GetFileName(fullPath), new Uri(fullPath));
165             MediaPlayer.Play(_music);
166         }
167         catch { /* le fichier n'existe plus ou ce n'est pas une musique */ }
168     }
169 }
170
171 }
172
173
174 /// <summary>
175 /// Ajoute dans la liste de mortier un élément récupéré de fichier xml
176 /// </summary>
177 /// <param name="mortar"></param>
178 public void AddMortarFromXElementToListMortar(XElement mortar)
179 {
180     _lstMortar.Add(new Mortar(new ←
        Vector2(float.Parse(mortar.Attribute(ATTRIBUTE_POSITION_X).Value), ←
            float.Parse(mortar.Attribute(ATTRIBUTE_POSITION_Y).Value)), ←
            float.Parse(mortar.Attribute(ATTRIBUTE_WIDTH).Value),
181            float.Parse(mortar.Attribute(ATTRIBUTE_HEIGHT).Value), ←
            float.Parse(mortar.Attribute(ATTRIBUTE_ANGLE).Value), Color.White));
182 }
183
184 /// <summary>
185 /// Sauvegarde toute la séquence en XML
186 /// </summary>
187 public void SaveSequence()
188 {
189     string musicName = Globals.MusicSelectedName != "" ? Config.PATH_MUSIC + ←
        Globals.MusicSelectedName : string.Empty;
190     // Information global de la séquence, nom, auteur, date...
191     DateTime currentDate = DateTime.Now;
192     XDocument document = new XDocument(
193         new XElement(ELEMENT_FIREWORK_SEQUENCE,
194             new XAttribute(ATTRIBUTE_NAME, Config.NAME_SEQUENCE),
195             new XAttribute(ATTRIBUTE_CREATION_DATE, ←
                currentDate.ToString("yyyy-MM-dd")),
196             new XAttribute(ATTRIBUTE_AUTHOR, Config.AUTHOR_FILE),
197             new XAttribute(ATTRIBUTE_TIME_END, _timerLauch.ToString().Replace(".", ←
                ",")),
198             new XElement(ELEMENT_AUDIO,
199                 new XAttribute(ATTRIBUTE_TRACK, musicName)
200             ),
201             new XElement(ELEMENT_BACKGROUND,
202                 new XAttribute(ATTRIBUTE_IMG, Config.PATH_IMG)
203             )
204         )
205     );
206     XElement fireworkSequence = ←
        document.Descendants(ELEMENT_FIREWORK_SEQUENCE).FirstOrDefault();
207
208     // Ajoute les informations des mortiers
209     foreach (Mortar mortar in _lstMortar)
210     {
211         fireworkSequence.Add(
212             new XElement(ELEMENT_MORTAR,
213                 new XAttribute(ATTRIBUTE_POSITION_X, ←
                    mortar.Position.X.ToString().Replace(".", ",")),
214                 new XAttribute(ATTRIBUTE_POSITION_Y, ←
                    mortar.Position.Y.ToString().Replace(".", ",")),
215                 new XAttribute(ATTRIBUTE_WIDTH, ←
                    mortar.Width.ToString().Replace(".", ",")),
216                 new XAttribute(ATTRIBUTE_HEIGHT, ←
                    mortar.Height.ToString().Replace(".", ",")),
217                 new XAttribute(ATTRIBUTE_ANGLE, ←
                    MathHelper.ToDegrees(mortar.Angle).ToString().Replace(".", ",")))

```

```

218         );
219     };
220 }
221
222 // Ajoute les feux d'artifices
223 foreach (IFirework firework in Globals.LstFirework)
224 {
225     if (firework is Comet comet)
226     {
227         XElement cometElement = CreateCommonFireworkElement(comet, ←
228             ATTRIBUTE_TYPE_COMET);
229         cometElement.Add(
230             new XElement(ELEMENT_SIZE,
231                 new XAttribute(ATTRIBUTE_MAIN_SIZE, Config.COMET_MAIN_SIZE),
232                 new XAttribute(ATTRIBUTE_OTHER_SIZE, Config.COMET_OTHER_SIZE)
233             ),
234             new XElement(ELEMENT_START,
235                 new XAttribute(ATTRIBUTE_POSITION_X, ←
236                     comet.StartPosition.X.ToString().Replace(".", ",")),
237                 new XAttribute(ATTRIBUTE_POSITION_Y, ←
238                     comet.StartPosition.Y.ToString().Replace(".", ",")),
239                 new XAttribute(ATTRIBUTE_ANGLE, ←
240                     comet.StartAngle.ToString().Replace(".", ",")),
241                 new XAttribute(ATTRIBUTE_SPEED, ←
242                     comet.StartSpeed.ToString().Replace(".", ",")),
243                 new XAttribute(ATTRIBUTE_LIFESPAN, ←
244                     comet.Lifespan.ToString().Replace(".", ","))
245             )
246         );
247         fireworkSequence.Add(cometElement);
248     }
249     else if (firework is ParticleRain rain)
250     {
251         XElement rainElement = CreateCommonFireworkElement(rain, ←
252             ATTRIBUTE_TYPE_PARTICLE_RAIN);
253         rainElement.Add(
254             new XElement(ELEMENT_SIZE, Config.PARTICLE_RAIN_SIZE),
255             new XElement(ELEMENT_START,
256                 new XAttribute(ATTRIBUTE_POSITION_X, ←
257                     rain.StartPosition.X.ToString().Replace(".", ",")),
258                 new XAttribute(ATTRIBUTE_POSITION_Y, ←
259                     rain.StartPosition.Y.ToString().Replace(".", ",")),
260                 new XAttribute(ATTRIBUTE_SPEED, ←
261                     rain.StartSpeed.ToString().Replace(".", ",")),
262                 new XAttribute(ATTRIBUTE_LIFESPAN, ←
263                     rain.Lifespan.ToString().Replace(".", ",")),
264                 new XAttribute(ATTRIBUTE_NB_PARTICLE, Config.PARTICLE_RAIN_NB)
265             )
266         );
267         fireworkSequence.Add(rainElement);
268     }
269 }
270
271 // Sauvegarde le fichier
272 document.Save($"{Config.PATH_SAVE_SEQUENCE}{currentDate.ToString("yyyy-MM-dd ←
273     HH_mm_ss")}.xml");
274 Globals.LstFirework.Clear();
275 _timerLaunch = 0f;
276 }
277
278 /// <summary>
279 /// Crée les éléments communs au feu d'artifice comme la couleur et le temps de ←
280 lancement
281 /// </summary>
282 /// <param name="firework">feu d'artifice lancé, pour récupérer le launchTime</param>
283 /// <param name="type">type de feu d'artifice : Comète, pluie de particule</param>
284 private static XElement CreateCommonFireworkElement(IFirework firework, string type)
285 {
286     XElement baseParticle = new XElement(ELEMENT_FIREWORK,
287         new XAttribute(ATTRIBUTE_TYPE, type),
288         new XAttribute(ATTRIBUTE_LAUNCH_TIME, ←
289             firework.LaunchTime.ToString().Replace(".", ","))
290     );
291     if (type == ATTRIBUTE_TYPE_COMET)
292     {
293         baseParticle.Add(
294             new XElement(ELEMENT_COLOR_START,
295                 new XAttribute(ATTRIBUTE_R_COLOR, Config.COLOR_START_COMET.R),
296                 new XAttribute(ATTRIBUTE_G_COLOR, Config.COLOR_START_COMET.G),
297                 new XAttribute(ATTRIBUTE_B_COLOR, Config.COLOR_START_COMET.B)
298             ),
299             new XElement(ELEMENT_COLOR_END,
300                 new XAttribute(ATTRIBUTE_R_COLOR, Config.COLOR_END_COMET.R),
301                 new XAttribute(ATTRIBUTE_G_COLOR, Config.COLOR_END_COMET.G),
302                 new XAttribute(ATTRIBUTE_B_COLOR, Config.COLOR_END_COMET.B)
303             )
304         );
305     }
306 }

```

```

291         else if (type == ATTRIBUTE_TYPE_PARTICLE_RAIN)
292         {
293             baseParticle.Add(
294                 new XElement(ELEMENT_COLOR_START,
295                     new XAttribute(ATTRIBUTE_R_COLOR, Config.COLOR_PARTICLE_RAIN_START.R),
296                     new XAttribute(ATTRIBUTE_G_COLOR, Config.COLOR_PARTICLE_RAIN_START.G),
297                     new XAttribute(ATTRIBUTE_B_COLOR, Config.COLOR_PARTICLE_RAIN_START.B)
298                 ),
299                 new XElement(ELEMENT_COLOR_END,
300                     new XAttribute(ATTRIBUTE_R_COLOR, Config.COLOR_PARTICLE_RAIN_END.R),
301                     new XAttribute(ATTRIBUTE_G_COLOR, Config.COLOR_PARTICLE_RAIN_END.G),
302                     new XAttribute(ATTRIBUTE_B_COLOR, Config.COLOR_PARTICLE_RAIN_END.B)
303                 )
304             );
305         }
306         return baseParticle;
307     }
308
309     /// <summary>
310     /// Crée une comète
311     /// </summary>
312     /// <param name="velocity">La vitesse change en fonction de la vélocité</param>
313     public void CreateComete(int velocity)
314     {
315         int nbMortar = Globals.RandomInt(0, Config.ALL_MORTAR.Count - 1);
316         Vector2 emitPos = _lstMortar[nbMortar].Position;
317         emitPos.X += _lstMortar[nbMortar].Width / 2;
318         Globals.LstFirework.Add(new Comet(emitPos, _lstMortar[nbMortar].Angle, ←
            Config.COMET_DEFAULT_SPEED * velocity, Config.COMET_DEFAULT_LIFESPAN, ←
            _timerLaunch));
319     }
320
321     /// <summary>
322     /// Crée une pluie de particule
323     /// </summary>
324     /// <param name="velocity">La durée de vie change en fonction de la vélocité</param>
325     public void CreateParticleRain(int velocity)
326     {
327         Globals.LstFirework.Add(new ParticleRain(Config.PARTICLE_RAIN_SPEED, ←
            Config.PARTICLE_RAIN_LIFESPAN * velocity, _timerLaunch));
328     }
329
330     public void Update()
331     {
332         _timerLaunch += Globals.TotalSeconds;
333
334         _menuButton.Update();
335         try
336         {
337             Globals.LstFirework.ForEach(x => x.Update());
338         }
339         catch (InvalidOperationException) { /* Il arrive parfois qu'un feu d'artifice ←
            soit ajouté pendant la mise à jour */ }
340
341         if (Mode)
342         {
343             _saveButton.Update();
344
345             if (_saveButton.IsPressed)
346             {
347                 SaveSequence();
348                 showMessageSave = true;
349             }
350
351             // permet d'afficher le message de confirmation de sauvegarde pendant un ←
            certain temps
352             if (showMessageSave)
353             {
354                 _timerSave += Globals.TotalSeconds;
355
356                 if (_timerSave >= TIME_MESSAGE_SAVE)
357                 {
358                     _timerSave = 0;
359                     showMessageSave = false;
360                 }
361             }
362
363             // permet d'utiliser les touches du clavier au cas où la guitare ne ←
            fonctionne pas lors de la présentation
364             if (InputManager.IsKeyParticleRainPressed)
365             {
366                 Globals.LstFirework.Add(new ParticleRain(80, 3f, _timerLaunch));
367             }
368             if (InputManager.IsKeyCometPressed)
369             {
370                 int nbMortar = Globals.RandomInt(0, Config.ALL_MORTAR.Count - 1);
371                 Vector2 emitPos = _lstMortar[nbMortar].Position;

```

```

372         emitPos.X += _lstMortar[nbMortar].Width / 2;
373         Globals.LstFirework.Add(new Comet(emitPos, _lstMortar[nbMortar].Angle, ←
        400, 1.5f, _timerLaunch));
374     }
375 }
376 else
377 {
378     // Rejoue toute la séquence
379     foreach (XElement firework in _file.Descendants(ELEMENT_FIREWORK))
380     {
381         if (float.Parse(firework.Attribute(ATTRIBUTE_LAUNCH_TIME).Value) == ←
            _timerLaunch)
382         {
383             // Récupère les informations communs aux feux d'artifices
384             Color colorStart = Globals.GetColorFromElement(firework.
                Descendants(ELEMENT_COLOR_START).FirstOrDefault());
385             Color colorEnd = Globals.GetColorFromElement(firework.
                Descendants(ELEMENT_COLOR_END).FirstOrDefault());
386             float positionX = float.Parse(firework.Descendants(ELEMENT_START).
                FirstOrDefault().Attribute(ATTRIBUTE_POSITION_X).Value);
387             float positionY = float.Parse(firework.Descendants(ELEMENT_START).
                FirstOrDefault().Attribute(ATTRIBUTE_POSITION_Y).Value);
388             float speed = float.Parse(firework.Descendants(ELEMENT_START).
                FirstOrDefault().Attribute(ATTRIBUTE_SPEED).Value);
389             float lifespan = float.Parse(firework.Descendants(ELEMENT_START).
                FirstOrDefault().Attribute(ATTRIBUTE_LIFESPAN).Value);
390
391             string fireworkType = firework.Attribute(ATTRIBUTE_TYPE).Value;
392             if (fireworkType == ATTRIBUTE_TYPE_COMET)
393             {
394                 // Crée une comète
395                 float sizeMain = float.Parse(firework.Descendants(ELEMENT_SIZE).
                    FirstOrDefault().Attribute(ATTRIBUTE_MAIN_SIZE).Value);
396                 float sizeOther = float.Parse(firework.Descendants(ELEMENT_SIZE).
                    FirstOrDefault().Attribute(ATTRIBUTE_OTHER_SIZE).Value);
397                 float angle = float.Parse(firework.Descendants(ELEMENT_START).
                    FirstOrDefault().Attribute(ATTRIBUTE_ANGLE).Value);
398                 Globals.LstFirework.Add(new Comet(new Vector2(positionX, ←
                    positionY), angle, speed, lifespan, colorStart, colorEnd, ←
                    sizeMain, sizeOther));
399             }
400             else if (fireworkType == ATTRIBUTE_TYPE_PARTICLE_RAIN)
401             {
402                 // Créer une pluie de particule
403                 float size = float.Parse(firework.Descendants(ELEMENT_SIZE).
                    FirstOrDefault().Value);
404                 float nbParticle = ←
                    float.Parse(firework.Descendants(ELEMENT_START).
                    FirstOrDefault().Attribute(ATTRIBUTE_NB_PARTICLE).Value);
405                 Globals.LstFirework.Add(new ParticleRain(new Vector2(positionX, ←
                    positionY), speed, lifespan, colorStart, colorEnd, size, ←
                    nbParticle));
406             }
407         }
408     }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 /// <summary>
424 /// Méthode d'affichage du jeu, libre et replay
425 /// </summary>
426 public void Draw()
427 {
428     if (Mode)
429     {
430         // Affiche l'image de fond si elle a été spécifiée dans le fichier de ←
            configuration
431         if (_background != null)
432         {
433             Globals.SpriteBatch.Draw(_background, new Rectangle(0, 0, ←
                Globals.ScreenWidth, Globals.ScreenHeight), Color.White);
434         }
435         _saveButton.Draw();
436
437         // Affiche le message de confirmation de sauvegarde
438         if (showMessageSave)
439         {
440             Globals.SpriteBatch.DrawString(Globals.FontButton, "Sauvegarde ←
                effectue", new Vector2(0.5f * Globals.ScreenWidth, 0.5f * ←
                Globals.ScreenHeight), Color.Red);
441         }
442     }
443     else
444     {
445         // Affiche l'image de fond si elle a été spécifiée dans le fichier de la ←
            séquence
446         if (_background != null)

```



```
447         {
448             Globals.SpriteBatch.Draw(_background, new Rectangle(0, 0, ↵
449                                     Globals.ScreenWidth, Globals.ScreenHeight), Color.White);
450         }
451         // Affiche les données du replay
452         Globals.SpriteBatch.DrawString(Globals.FontButton, $"Nom de la sequence : ↵
453                                     {_file.Attribute(ATTRIBUTE_NAME).Value}", new Vector2(0.75f * ↵
454                                     Globals.ScreenWidth, 0.05f * Globals.ScreenHeight), Color.White);
455         Globals.SpriteBatch.DrawString(Globals.FontButton, $"Auteur : ↵
456                                     {_file.Attribute(ATTRIBUTE_AUTHOR).Value}", new Vector2(0.75f * ↵
457                                     Globals.ScreenWidth, 0.1f * Globals.ScreenHeight), Color.White);
458         Globals.SpriteBatch.DrawString(Globals.FontButton, $"Date : ↵
459                                     {_file.Attribute(ATTRIBUTE_CREATION_DATE).Value}", new Vector2(0.75f * ↵
460                                     Globals.ScreenWidth, 0.15f * Globals.ScreenHeight), Color.White);
461
462         // Affiche un message de fin de replay
463         if (_timerLauch >= float.Parse(_file.Attribute(ATTRIBUTE_TIME_END).Value))
464         {
465             Globals.SpriteBatch.DrawString(Globals.FontButton, "Fin du replay", new ↵
466                                     Vector2(0.5f * Globals.ScreenWidth, 0.5f * Globals.ScreenHeight), ↵
467                                     Color.Red);
468         }
469     }
470     _menuButton.Draw();
471     _lstMortar.ForEach(m => m.Draw());
472 }
473 }
```