Code Source

Travail Pratique Individuel
MiFiSy

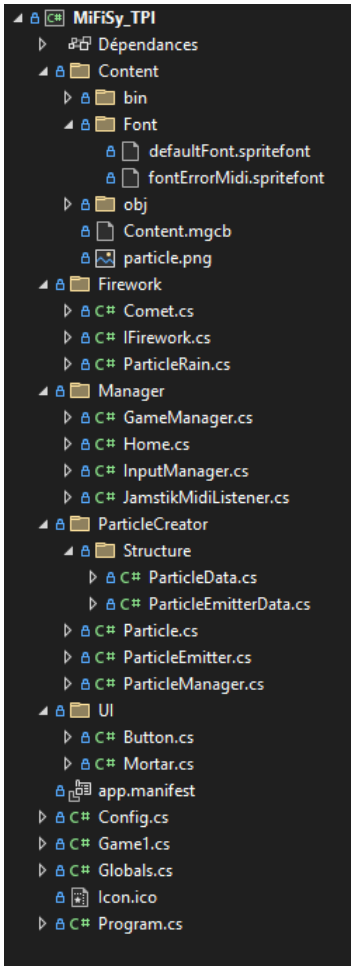
CFPT Informatique

Yoann Meier

15 mai 2024

Table des matières

Structure des fichiers



Config.cs

```
using Microsoft.Xna.Framework;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Class permettant de récupérer en static toutes les données du fichier de configuration
 */
namespace MiFiSy_TPI
{
    internal class Config
    {
        private static XElement _configElement;

        public Config()
        {
            _configElement = XDocument.Load("config.xml").Descendants("Config").FirstOrDefault();
        }

        // Propriétés statiques pour accéder aux valeurs du fichier XML
        public static string AUTHOR_FILE { get => _configElement.Descendants("Author").FirstOrDefault().Value; }
        public static string NAME_SEQUENCE { get => _configElement.Descendants("NameSequence").FirstOrDefault().Value; }
        public static string PATH_MUSIC { get => _configElement.Descendants("PathMusic").FirstOrDefault().Value; }
        public static string PATH_IMG { get => _configElement.Descendants("PathImg").FirstOrDefault().Value; }
        public static string PATH_SAVE_SEQUENCE { get => _configElement.Descendants("PathSaveSequence").FirstOrDefault().Value; }
        public static List<XElement> ALL_MORTAR { get => _configElement.Descendants("Mortar").ToList(); }
        public static Color COLOR_START_COMET { get => Globals.GetColorFromElement(_configElement.Descendants("ColorStartComet")
            .FirstOrDefault()); }
        public static Color COLOR_END_COMET { get => Globals.GetColorFromElement(_configElement.Descendants("ColorEndComet")
            .FirstOrDefault()); }
        public static Color COLOR_PARTICLE_RAIN_START { get => Globals.GetColorFromElement(_configElement.Descendants("ColorStartParticleRain")
            .FirstOrDefault()); }
        public static Color COLOR_PARTICLE_RAIN_END { get => Globals.GetColorFromElement(_configElement.Descendants("ColorEndParticleRain")
            .FirstOrDefault()); }
        public static int PARTICLE_RAIN_SIZE { get => Convert.ToInt32(_configElement.Descendants("ParticleRain").FirstOrDefault().
            Attribute("sizeParticle").Value); }
        public static int PARTICLE_RAIN_NB { get => Convert.ToInt32(_configElement.Descendants("ParticleRain").FirstOrDefault().
            Attribute("nbParticle").Value); }
        public static float PARTICLE_RAIN_LIFESPAN { get => float.Parse(_configElement.Descendants("ParticleRain").FirstOrDefault().
            Attribute("lifeSpan").Value); }
        public static float PARTICLE_RAIN_TIME_SPAWN { get => float.Parse(_configElement.Descendants("ParticleRain").FirstOrDefault().
            Attribute("timeSpawn").Value); }
        public static float PARTICLE_RAIN_SPEED { get => float.Parse(_configElement.Descendants("ParticleRain").FirstOrDefault().
            Attribute("defaultSpeed").Value); }
        public static int COMET_MAIN_SIZE { get => Convert.ToInt32(_configElement.Descendants("Comet").FirstOrDefault().
            Attribute("sizeMainParticle").Value); }
        public static int COMET_OTHER_SIZE { get => Convert.ToInt32(_configElement.Descendants("Comet").FirstOrDefault().
            Attribute("sizeOtherParticle").Value); }
```

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

Globals.cs

```cs
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using MiFiSy_TPI.Firework;
using MiFiSy_TPI.Manager;
using System;
using System.Collections.Generic;
using System.Xml.Linq;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Page contenant des valeurs static nécéssaires dans plusieurs pages
 */
namespace MiFiSy_TPI
{
    internal static class Globals
    {
        public static float TotalSeconds { get; set; }

        /// <summary>
        /// Enum de toutes les pages de l'application
        /// </summary>
        public enum AllPage
        {
            Home,
            Game,
        }

        /// <summary>
        /// Page actuel
        /// </summary>
        public static AllPage ActualPage {  get; set; }

        public static ContentManager Content { get; set; }

        public static SpriteBatch SpriteBatch { get; set; }

        public static SpriteFont DefaultFontButton { get; set; }

        public static GraphicsDevice GraphicsDevice { get; set; }

        public static Random Random { get; set; } = new Random();

        /// <summary>
        /// Largeur de l'écran
        /// </summary>
        public static int ScreenWidth { get; set; }

        /// <summary>
        /// Hauteur de l'écran
        /// </summary>
        public static int ScreenHeight { get; set; }

        /// <summary>
        /// Nom de la musique séléctionné
        /// </summary>
        public static string MusicSelectedName { get; set; }

        public static Home home { get; set; }

        public static GameManager GameManager { get; set; }

        /// <summary>
        /// Liste de feu d'artifice
        /// </summary>
        public static List<IFirework> LstFirework { get; set; }

        public static void Update(GameTime gt)
        {
            TotalSeconds = (float)gt.ElapsedGameTime.TotalSeconds;
```

```
72          }
73
74          /// <summary>
75          /// Retourne un float aléatoire entre min et max
76          /// </summary>
77          /// <param name="min">nombre minimum</param>
78          /// <param name="max">nombre maximum</param>
79          /// <returns>nombre aléatoire</returns>
80          public static float RandomFloat(float min, float max)
81          {
82              return (float)(Random.NextDouble() * (max - min)) + min;
83          }
84
85          /// <summary>
86          /// Retourne un nombre aléatoire entre min et max
87          /// </summary>
88          /// <param name="min">nombre minimum</param>
89          /// <param name="max">nombre maximum</param>
90          /// <returns>nombre aléatoire</returns>
91          public static int RandomInt(int min, int max)
92          {
93              return Random.Next(min, max + 1);
94          }
95
96          /// <summary>
97          /// Méthode pour récupérer la couleur à partir d'un élément XML
98          /// </summary>
99          /// <param name="colorElement">XElement contenant les attributs "r", "g" et ↩
                "b"</param>
100         public static Color GetColorFromElement(XElement colorElement)
101         {
102             if (colorElement.Attribute("r") != null && colorElement.Attribute("g") != null ↩
                    && colorElement.Attribute("b") != null)
103             {
104                 int r = Convert.ToInt32(colorElement.Attribute("r").Value);
105                 int g = Convert.ToInt32(colorElement.Attribute("g").Value);
106                 int b = Convert.ToInt32(colorElement.Attribute("b").Value);
107                 return new Color(r, g, b);
108             }
109             return Color.White;
110         }
111     }
112 }
```

Game1.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Media;
using MiFiSy_TPI.Firework;
using MiFiSy_TPI.Manager;
using MiFiSy_TPI.ParticleCreator;
using System.Collections.Generic;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Page principale de l'application
 */
namespace MiFiSy_TPI
{
    public class Game1 : Game
    {
        private GraphicsDeviceManager _graphics;
        private SpriteBatch _spriteBatch;
        private JamstikMidiListener _jamstikMidiListener;

        public Game1()
        {
            _graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            IsMouseVisible = true;
        }

        protected override void Initialize()
        {
            // taille de l'application en fonction de la taille de l'écran
            _graphics.PreferredBackBufferWidth = ↩
                GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Width;
            _graphics.PreferredBackBufferHeight = ↩
                GraphicsAdapter.DefaultAdapter.CurrentDisplayMode.Height;
            _graphics.IsFullScreen = false;
            _graphics.ApplyChanges();

            Globals.ScreenWidth = _graphics.PreferredBackBufferWidth;
            Globals.ScreenHeight = _graphics.PreferredBackBufferHeight;
            Globals.Content = Content;
            Globals.GraphicsDevice = GraphicsDevice;
            Globals.ActualPage = Globals.AllPage.Home;
            Globals.LstFirework = new List<IFirework>();
            Globals.MusicSelectedName = "";
            Globals.DefaultFontButton = Content.Load<SpriteFont>("Font/defaultFont");

            // Permet de mettre en boucle les musiques
            MediaPlayer.IsRepeating = true;

            new Config();
            Globals.home = new Home();
            base.Initialize();
        }

        protected override void LoadContent()
        {
            _spriteBatch = new SpriteBatch(GraphicsDevice);
            Globals.SpriteBatch = _spriteBatch;
            _jamstikMidiListener = new ↩
                JamstikMidiListener(Content.Load<SpriteFont>("Font/fontErrorMidi"));
        }

        protected override void Update(GameTime gameTime)
        {
            Globals.Update(gameTime);
            InputManager.Update();
            switch (Globals.ActualPage)
            {
                // Page d'accueil
                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

```cs
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Struct contenant les variables d'une particule (Viens de : https://www.youtube.com/watch?v=-4_kj_gyWRY)
 */
namespace MiFiSy_TPI.ParticleCreator.Structure
{
    internal struct ParticleData
    {
        public Texture2D texture = Globals.Content.Load<Texture2D>("particle");
        public float lifespan = 2f;
        public Color colorStart = Color.Yellow;
        public Color colorEnd = Color.Red;
        public float opacityStart = 1f;
        public float opacityEnd = 0f;
        public float sizeStart = 32f;
        public float sizeEnd = 4f;
        public float speed = 100f;
        public float angle = 0f;

        public ParticleData()
        {
        }
    }
}
```

ParticleEmitterData.cs

```cs
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Struct contenant les variables d'un emitteur de particules (Viens de : https://www.youtube.com/watch?v=-4_kj_gyWRY)
 */
namespace MiFiSy_TPI.ParticleCreator.Structure
{
    internal struct ParticleEmitterData
    {
        public ParticleData particleData = new ParticleData();
        public float angle = 0f;
        public float angleVariance = 0f;
        public float lifespanMin = 0.1f;
        public float lifespanMax = 2f;
        public float speedMin = 10f;
        public float speedMax = 100f;
        public float interval = 1f;
        public int emitCount = 1;
        public bool decreasedLifespan = false;
        public float nbDecreasedLifespan = 0.05f;
        public bool randomPosX = false;
        public float intervalPos = 0.01f;
        public ParticleEmitterData()
        {
        }
    }
}
```

Particle.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using MiFiSy_TPI.ParticleCreator.Structure;
using System;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Class d'une particule (Viens de : https://www.youtube.com/watch?v=-4_kj_gyWRY)
 */
namespace MiFiSy_TPI.ParticleCreator
{
    internal class Particle
    {
        private ParticleData _data;
        private Vector2 _position;
        private float _lifespanLeft;
        private float _lifespanAmount;
        private Color _color;
        private float _opacity;
        public bool isFinished = false;
        private float _scale;
        private Vector2 _origin;
        private Vector2 _direction;

        public Vector2 Position { get => _position; set => _position = value; }
        internal ParticleData Data { get => _data; set => _data = value; }

        public Particle(Vector2 pos, ParticleData data)
        {
            _data = data;
            _lifespanLeft = data.lifespan;
            _lifespanAmount = 1f;
            _position = pos;
            _color = data.colorStart;
            _opacity = data.opacityStart;
            _origin = new Vector2(_data.texture.Width / 2, _data.texture.Height / 2);

            SetAngleAndDirection();
        }

        /// <summary>
        /// Calcul la direction de la particule avec l'angle
        /// </summary>
        public void SetAngleAndDirection()
        {
            if (_data.speed != 0)
            {
                // Converti l'angle en radians
                _data.angle = MathHelper.ToRadians(_data.angle);
                // Calcul la direction grace à l'angle
                _direction = new Vector2((float)Math.Sin(_data.angle), -(float)Math.Cos(_data.angle));
            }
            else
            {
                _direction = Vector2.Zero;
            }
        }

        public void Update()
        {
            _lifespanLeft -= Globals.TotalSeconds;
            if (_lifespanLeft <= 0f)
            {
                isFinished = true;
                return;
            }

            // Calcule le temps de vie restant
```

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

ParticleEmitter.cs

```cs
using Microsoft.Xna.Framework;
using MiFiSy_TPI.ParticleCreator.Structure;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Class d'une particule (Viens de : https://www.youtube.com/watch?v=-4_kj_gyWRY)
 */
namespace MiFiSy_TPI.ParticleCreator
{
    internal class ParticleEmitter
    {
        private ParticleEmitterData data;
        private float _intervalLeft;
        private Vector2 _emitPosition;
        public bool destroy;

        internal ParticleEmitterData Data { get => data; set => data = value; }

        public ParticleEmitter(Vector2 emitPosition, ParticleEmitterData data)
        {
            _emitPosition = emitPosition;
            this.data = data;
            _intervalLeft = data.interval;
            destroy = false;
        }

        /// <summary>
        /// Émet une nouvelle particule avec une position
        /// </summary>
        /// <param name="pos">La position à partir de laquelle émettre la particule</param>
        public void Emit(Vector2 pos)
        {
            ParticleData d = data.particleData;
            // Random lifespan, speed, angle
            d.lifespan = Globals.RandomFloat(data.lifespanMin, data.lifespanMax);
            d.speed = Globals.RandomFloat(data.speedMin, data.speedMax);
            d.angle = Globals.RandomFloat(data.angle - data.angleVariance, data.angle + data.angleVariance);

            if (data.randomPosX)
            {
                // Position random X
                float xPosition = pos.X * Globals.ScreenWidth;
                float randomX = Globals.RandomFloat(xPosition - data.intervalPos * Globals.ScreenWidth, xPosition + data.intervalPos * Globals.ScreenWidth) / Globals.ScreenWidth;
                pos.X = randomX;
            }
            Particle p = new Particle(pos, d);
            ParticleManager.AddParticle(p);
        }

        public void Update()
        {
            _intervalLeft -= Globals.TotalSeconds;
            if (_intervalLeft <= 0f)
            {
                // Réinitialise le temps restant
                _intervalLeft += data.interval;
                // Emet les nouvelles particules
                for (int i = 0; i < data.emitCount; i++)
                {
                    Emit(_emitPosition);
                }

                // Diminue le lifespan des prochaines particules
                if (data.decreasedLifespan)
                {
```

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

ParticleManager.cs

```cs
using System;
using System.Collections.Generic;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Class d'une particule (Viens de : https://www.youtube.com/watch?v=-4_kj_gyWRY)
 */
namespace MiFiSy_TPI.ParticleCreator
{
    internal class ParticleManager
    {
        private static List<Particle> _particles = new List<Particle>();
        private static List<ParticleEmitter> _particleEmitters = new List<ParticleEmitter>();

        /// <summary>
        /// Ajoute une particule dans la liste
        /// </summary>
        public static void AddParticle(Particle p)
        {
            _particles.Add(p);
        }

        /// <summary>
        /// Ajoute un émetteur de particules
        /// </summary>
        public static void AddParticleEmitter(ParticleEmitter e)
        {
            _particleEmitters.Add(e);
        }

        public static void Update()
        {
            // Supprime les particules et émetteur finis
            _particles.RemoveAll(p => p.isFinished);
            _particleEmitters.RemoveAll(p => p.destroy);

            try
            {
                _particles.ForEach(p => p.Update());
                _particleEmitters.ForEach(e => e.Update());
            }
            catch (InvalidOperationException) { /* Il arrive parfois qu'une particule ou émetteur soit ajouté pendant la mise à jour */ }
        }

        /// <summary>
        /// Supprime une paricule
        /// </summary>
        public static void RemoveParticle(Particle p)
        {
            _particles.Remove(p);
        }

        /// <summary>
        /// Supprime un émetteur de paricules
        /// </summary>
        public static void RemoveParticleEmitter(ParticleEmitter p)
        {
            _particleEmitters.Remove(p);
        }

        /// <summary>
        /// Affiche les particules
        /// </summary>
        public static void Draw()
        {
            try
            {
```

```
69                  _particles.ForEach(p => p.Draw());
70              }
71              catch (InvalidOperationException) { /* Il arrive parfois qu'une particule ou é↩
                    metteur soit ajouté pendant l'affichage */ }
72          }
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

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Media;
using MiFiSy_TPI.Manager;
using MiFiSy_TPI.ParticleCreator;
using System;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Class d'un bouton
 */
namespace MiFiSy_TPI.UI
{
    internal class Button
    {
        private Vector2 _position;
        private Vector2 _textPosition;
        private Texture2D _texture;
        private string _text;
        private float _widthRectangle;
        private float _heightRectangle;
        private Color _backgroundColor;
        private Color _textColor;
        private float _padding;
        private float _scale;
        private string _action;
        private bool _isPressed;

        public Rectangle Rectangle { get => new Rectangle((int)(_position.X * Globals.ScreenWidth), (int)(_position.Y * Globals.ScreenHeight), _texture.Width, _texture.Height); }
        public bool IsPressed { get => _isPressed; set => _isPressed = value; }
        public string Text { get => _text; set => _text = value; }
        public Color TextColor { get => _textColor; set => _textColor = value; }

        public Button(Vector2 position, float width, float height, string text, Color backgroundColor, Color textColor, string action, float padding = 0.2f)
        {
            _position = position;
            _widthRectangle = width;
            _heightRectangle = height;
            _text = text;
            _backgroundColor = backgroundColor;
            _textColor = textColor;
            _padding = padding;
            _action = action;
            _scale = 1;
            IsPressed = false;

            SetTexture();
            SetTextPositionAndScale();
        }

        /// <summary>
        /// Crée la texture du rectangle du bouton avec ses dimensions et sa couleur
        /// </summary>
        public void SetTexture()
        {
            int width = (int)(_widthRectangle * Globals.ScreenWidth);
            int height = (int)(_heightRectangle * Globals.ScreenHeight);
            _texture = new Texture2D(Globals.GraphicsDevice, width, height);
            Color[] colorData = new Color[width * height];
            for (int i = 0; i < colorData.Length; ++i)
            {
                colorData[i] = _backgroundColor;
            }
            _texture.SetData(colorData);
        }

        /// <summary>
```

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

Mortar.cs

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using System;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Class d'un mortier
 */
namespace MiFiSy_TPI.UI
{
    internal class Mortar
    {
        private Vector2 _position;
        private Color _color;
        private float _width;
        private float _height;
        private Texture2D _texture;
        private float _angle;

        public Rectangle Rectangle { get => new Rectangle((int)(Position.X * Globals.ScreenWidth), (int)(Position.Y * Globals.ScreenHeight), _texture.Width, _texture.Height); }
        public Vector2 Position { get => _position; set => _position = value; }
        public float Width { get => _width; set => _width = value; }
        public float Angle { get => _angle; set => _angle = value; }
        public float Height { get => _height; set => _height = value; }

        public Mortar(Vector2 position, float width, float height, float angle, Color color)
        {
            Position = position;
            _color = color;
            Width = width;
            Height = height;
            Angle = Globals.RandomFloat(-angle, angle);

            // Converti l'angle en radians
            if (Angle >= 0)
            {
                Angle = MathHelper.ToRadians(Angle);
            }
            else
            {
                Angle = -MathHelper.ToRadians(Math.Abs(Angle));
            }
            SetTexture();
        }

        /// <summary>
        /// Crée la texture du rectangle du bouton avec ses dimensions et sa couleur
        /// </summary>
        public void SetTexture()
        {
            int width = (int)(Width * Globals.ScreenWidth);
            int height = (int)(Height * Globals.ScreenHeight);
            _texture = new Texture2D(Globals.GraphicsDevice, width, height);
            Color[] colorData = new Color[width * height];
            for (int i = 0; i < colorData.Length; ++i)
            {
                colorData[i] = _color;
            }
            _texture.SetData(colorData);
        }

        /// <summary>
        /// Affiche le mortier
        /// </summary>
        public void Draw()
        {
            Globals.SpriteBatch.Draw(_texture, Rectangle, null, _color, Angle, Vector2.Zero, SpriteEffects.None, 0);
```

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

IFirework.cs

```cs
using Microsoft.Xna.Framework;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Interface des particules pour n'avoir qu'une seule liste
 */
namespace MiFiSy_TPI.Firework
{
    public interface IFirework
    {
        /// <summary>
        /// position du feu d'artifice au départ
        /// </summary>
        Vector2 StartPosition { get; set; }

        /// <summary>
        /// durée de vie du feu d'artifice
        /// </summary>
        float Lifespan { get; set; }

        /// <summary>
        /// Temps après le début du mode libre où ce feu d'artifice est créé
        /// </summary>
        float LaunchTime { get; set; }

        /// <summary>
        /// Vitesse de départ
        /// </summary>
        float StartSpeed { get; set; }

        /// <summary>
        /// Méthode update pour supprimer les anciennes particules
        /// </summary>
        void Update();
    }
}
```

ParticleRain.cs

```cs
using Microsoft.Xna.Framework;
using MiFiSy_TPI.ParticleCreator;
using MiFiSy_TPI.ParticleCreator.Structure;
using System.Collections.Generic;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Class d'une pluie de particules, version 2
 */
namespace MiFiSy_TPI.Firework
{
    public class ParticleRain : IFirework
    {
        private List<Particle> _lstMainParticles;
        private float _lifespan;
        private float _timerLife;
        private float _timerSpawn;
        private float _launchTime;
        private Vector2 _startPosition;
        private float _startSpeed;
        private float _nbParticle;
        private Color _colorStart;
        private Color _colorEnd;
        private float _size;

        public float Lifespan { get => _lifespan; set => _lifespan = value; }
        public float LaunchTime { get => _launchTime; set => _launchTime = value; }
        public Vector2 StartPosition { get => _startPosition; set => _startPosition = value; }
        public float StartSpeed { get => _startSpeed; set => _startSpeed = value; }

        /// <summary>
        /// Constructeur de la classe utilisée dans le jeu libre : créer la pluie de particules en fonction de paramètre du fichier de configuration
        /// </summary>
        /// <param name="speed">vitesse de départ du feu d'artifice</param>
        /// <param name="lifespan">durée de vie du feu d'artifice</param>
        /// <param name="launchTime">Le temps à lequel l'effet a été créé, seulement utilisé pour la sauvegarde</param>
        /// <param name="distanceFromBorder">distance pour ne pas créer la particule en dehors ou sur le bord de l'écran</param>
        public ParticleRain(float speed, float lifespan, float launchTime, float distanceFromBorder = 100)
        {
            LaunchTime = launchTime;
            Lifespan = lifespan;
            StartSpeed = speed;
            _nbParticle = Config.PARTICLE_RAIN_NB;
            _colorStart = Config.COLOR_PARTICLE_RAIN_START;
            _colorEnd = Config.COLOR_PARTICLE_RAIN_END;
            _size = Config.PARTICLE_RAIN_SIZE;

            _timerLife = 0;
            _timerSpawn = 0;
            // Position aléatoire du feu d'artifice sur la partie haute de l'écran
            StartPosition = new Vector2(Globals.RandomFloat(distanceFromBorder, Globals.ScreenWidth - distanceFromBorder) / Globals.ScreenWidth, Globals.RandomFloat(distanceFromBorder, Globals.ScreenHeight / 2) / Globals.ScreenHeight);
            _lstMainParticles = new List<Particle>();

            for (int i = 0; i < _nbParticle; i++)
            {
                float angle = 360 / _nbParticle * i;
                // Vitesse aléatoire entre 0 et le maximum
                float newSpeed = Globals.RandomFloat(0, speed);
                ParticleData particleData = new ParticleData()
                {
                    angle = angle,
                    speed = newSpeed,
```

```
                    colorStart = _colorStart,
                    colorEnd = _colorEnd,
                    sizeStart = _size,
                    sizeEnd = _size,
                    lifespan = Lifespan,
                };
                Particle p = new Particle(StartPosition, particleData);
                _lstMainParticles.Add(p);
                ParticleManager.AddParticle(p);
            }
        }
        /// <summary>
        /// Constructeur de la classe utilisée dans le jeu replay : créer la pluie de particules en fonction de paramètre du fichier qui est rejoué
        /// </summary>
        /// <param name="position">position de départ</param>
        /// <param name="speed">vitesse de départ du feu d'artifice</param>
        /// <param name="lifespan">durée de vie du feu d'artifice</param>
        /// <param name="colorStart">couleur de départ</param>
        /// <param name="colorEnd">couleur de fin</param>
        /// <param name="size">taille des particules</param>
        /// <param name="nbParticle">nombre de particules a générer</param>
        public ParticleRain(Vector2 position, float speed, float lifespan, Color colorStart, Color colorEnd, float size, float nbParticle)
        {
            LaunchTime = 0f;
            Lifespan = lifespan;
            StartSpeed = speed;
            _nbParticle = nbParticle;
            _colorStart = colorStart;
            _colorEnd = colorEnd;
            _size = size;

            _timerLife = 0;
            _timerSpawn = 0;
            StartPosition = position;

            _lstMainParticles = new List<Particle>();

            for (int i = 0; i < _nbParticle; i++)
            {
                float angle = 360 / _nbParticle * i;
                // Vitesse aléatoire entre 0 et le maximum
                float newSpeed = Globals.RandomFloat(0, speed);
                ParticleData particleData = new ParticleData()
                {
                    angle = angle,
                    speed = newSpeed,
                    colorStart = _colorStart,
                    colorEnd = _colorEnd,
                    sizeStart = _size,
                    sizeEnd = _size,
                    lifespan = lifespan,
                };
                Particle p = new Particle(StartPosition, particleData);
                _lstMainParticles.Add(p);
                ParticleManager.AddParticle(p);
            }
        }

        /// <summary>
        /// Méthode qui fait apparaître de nouvelles particules si la durée de vie n'est pas atteinte, sinon supprime les particules
        /// </summary>
        public void Update()
        {
            _timerLife += Globals.TotalSeconds;
            _timerSpawn += Globals.TotalSeconds;
            // Supprime en fin de vie
            if (_timerLife >= Lifespan)
            {
                _lstMainParticles.Clear();
            }

            if (_lstMainParticles.Count != 0)
            {
                if (_timerSpawn >= Config.PARTICLE_RAIN_TIME_SPAWN)
                {
                    // Ajoute une particule immobile sur chaque particule en mouvement
                    for (int i = 0; i < _nbParticle; i++)
                    {
                        ParticleData particleData = new ParticleData()
                        {
                            angle = MathHelper.ToDegrees(_lstMainParticles[i].Data.angle),
                            speed = 0,
                            colorStart = _colorStart,
                            colorEnd = _colorEnd,
```

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

Comet.cs

```
using Microsoft.Xna.Framework;
using MiFiSy_TPI.ParticleCreator;
using MiFiSy_TPI.ParticleCreator.Structure;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Classe de création du feu d'artifice de la comète
 */
namespace MiFiSy_TPI.Firework
{
    public class Comet : IFirework
    {
        private Particle _mainParticle;
        private ParticleEmitter _emitter;
        private float _lifespan;
        private float _timerLife;
        private float _launchTime;
        private Vector2 _startPosition;
        private float _startAngle;
        private float _startSpeed;

        public float LaunchTime { get => _launchTime; set => _launchTime = value; }
        public Vector2 StartPosition { get => _startPosition; set => _startPosition = value; }
        internal Particle MainParticle { get => _mainParticle; set => _mainParticle = value; }
        public float StartAngle { get => _startAngle; set => _startAngle = value; }
        public float StartSpeed { get => _startSpeed; set => _startSpeed = value; }
        public float Lifespan { get => _lifespan; set => _lifespan = value; }


        /// <summary>
        /// Constructeur de la classe utilisée dans le jeu libre : créer la comète en fonction de paramètre du fichier de configuration
        /// </summary>
        /// <param name="position">Position de départ de la comète</param>
        /// <param name="angle">angle de la comète</param>
        /// <param name="speed">vitesse de la comète</param>
        /// <param name="lifespan">durée de vie de la comète</param>
        /// <param name="launchTime">Le temps à lequel l'effet a été créé, seulement utilisé pour la sauvegarde</param>
        public Comet(Vector2 position, float angle, float speed, float lifespan, float launchTime)
        {
            LaunchTime = launchTime;
            StartPosition = position;
            StartAngle = MathHelper.ToDegrees(angle);
            StartSpeed = speed;
            Lifespan = lifespan;
            _timerLife = 0;

            // Créer la particule principale, la tête
            ParticleData particleData = new ParticleData()
            {
                angle = StartAngle,
                speed = StartSpeed,
                lifespan = Lifespan,
                colorStart = Config.COLOR_START_COMET,
                colorEnd = Config.COLOR_END_COMET,
                sizeStart = Config.COMET_MAIN_SIZE,
                sizeEnd = Config.COMET_MAIN_SIZE,
            };
            _mainParticle = new Particle(position, particleData);
            ParticleManager.AddParticle(_mainParticle);

            // créer l'émetteur qui suit la tête, la queue
            ParticleEmitterData ped = new ParticleEmitterData()
            {
                interval = 0.01f,
                emitCount = 5,
```

```
                lifespanMin = Lifespan,
                lifespanMax = Lifespan,
                angle = StartAngle,
                randomPosX = true,
                intervalPos = 0.003f,
                decreasedLifespan = true,
                nbDecreasedLifespan = 0.05f,
                speedMin = StartSpeed,
                speedMax = StartSpeed,
                particleData = new ParticleData()
                {
                    colorStart = Config.COLOR_START_COMET,
                    colorEnd = Config.COLOR_END_COMET,
                    sizeStart = Config.COMET_OTHER_SIZE,
                    sizeEnd = Config.COMET_OTHER_SIZE,
                }
            };
            _emitter = new ParticleEmitter(_mainParticle.Position, ped);
            ParticleManager.AddParticleEmitter(_emitter);
        }

        /// <summary>
        /// Constructeur de la classe utilisée dans le jeu replay : créer la comète en ↩
            fonction de paramètre du fichier qui est rejoué
        /// </summary>
        /// <param name="position">Position de départ de la comète</param>
        /// <param name="angle">angle de la comète</param>
        /// <param name="speed">vitesse de la comète</param>
        /// <param name="lifespan">durée de vie de la comète</param>
        /// <param name="colorStart">couleur de départ de la comète</param>
        /// <param name="colorEnd">couleur de fin de la comète</param>
        /// <param name="mainSize">taille des particules de la tête</param>
        /// <param name="otherSize">taille des particules de la queue</param>
        public Comet(Vector2 position, float angle, float speed, float lifespan, Color ↩
            colorStart, Color colorEnd, float mainSize, float otherSize)
        {
            LaunchTime = 0f;
            StartPosition = position;
            StartAngle = angle;
            StartSpeed = speed;
            Lifespan = lifespan;
            _timerLife = 0;

            // Créer la particule principale, la tête
            ParticleData particleData = new ParticleData()
            {
                angle = StartAngle,
                speed = StartSpeed,
                lifespan = Lifespan,
                colorStart = colorStart,
                colorEnd = colorEnd,
                sizeStart = mainSize,
                sizeEnd = mainSize,
            };
            _mainParticle = new Particle(position, particleData);
            ParticleManager.AddParticle(_mainParticle);

            // créer l'émetteur qui suit la tête, la queue
            ParticleEmitterData ped = new ParticleEmitterData()
            {
                interval = 0.01f,
                emitCount = 5,
                lifespanMin = Lifespan,
                lifespanMax = Lifespan,
                angle = StartAngle,
                randomPosX = true,
                intervalPos = 0.003f,
                decreasedLifespan = true,
                nbDecreasedLifespan = 0.05f,
                speedMin = StartSpeed,
                speedMax = StartSpeed,
                particleData = new ParticleData()
                {
                    colorStart = colorStart,
                    colorEnd = colorEnd,
                    sizeStart = otherSize,
                    sizeEnd = otherSize,
                }
            };
            _emitter = new ParticleEmitter(_mainParticle.Position, ped);
            ParticleManager.AddParticleEmitter(_emitter);
        }

        public void Update()
        {
            _timerLife += Globals.TotalSeconds;
            if (_timerLife >= Lifespan)
```

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

JamstikMidiListener.cs

```cs
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using NAudio.Midi;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Classe pour gérer les entrés MIDI
 */
namespace MiFiSy_TPI.Manager
{
    internal class JamstikMidiListener
    {
        private MidiIn _midi;
        private bool _isConnected;
        private SpriteFont _font;

        /// <summary>
        /// Constructeur de la classe, connexion à la guitare
        /// </summary>
        /// <param name="font">font pour le message d'erreur si la guitare n'est pas trouvée</param>
        public JamstikMidiListener(SpriteFont font)
        {
            _font = font;
            _isConnected = false;

            if (MidiIn.NumberOfDevices != 0)
            {
                for (int i = 0; i < MidiIn.NumberOfDevices; i++)
                {
                    MidiInCapabilities capabilities = MidiIn.DeviceInfo(i);
                    // Connexion au Jamstik
                    if (capabilities.ProductName == "Jamstik")
                    {
                        _midi = new MidiIn(i);
                        _midi.MessageReceived += MidiIn_MessageReceived;
                        _midi.Start();
                        _isConnected = true;
                        break;
                    }
                }
            }
        }

        /// <summary>
        /// Méthode appelé évènements MIDI, traite seulement les évènements de notes, on ou off
        /// </summary>
        private void MidiIn_MessageReceived(object sender, MidiInMessageEventArgs e)
        {
            // Si on est dans le jeu en mode libre
            if (Globals.ActualPage == Globals.AllPage.Game && Globals.GameManager.Mode)
            {
                MidiEvent midiEvent = MidiEvent.FromRawMessage(e.RawMessage);
                if (midiEvent is NoteEvent noteEvent)
                {
                    // Lorsqu'une note est jouée
                    if (noteEvent.CommandCode == MidiCommandCode.NoteOn)
                    {
                        // Corde 1 jouée
                        if (noteEvent.Channel - 1 == 1)
                        {
                            Globals.GameManager.CreateComete(noteEvent.Velocity);
                        }
                        // Corde 2 jouée
                        else if (noteEvent.Channel - 1 == 2)
                        {
                            Globals.GameManager.CreateParticleRain(noteEvent.Velocity);
                        }
                    }
```

```
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

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Input;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Classe pour gérer la souris
 */
namespace MiFiSy_TPI.Manager
{
    internal class InputManager
    {
        private static MouseState _lastMouseState;
        private static bool _hasSpaceKeyUp;
        private static bool _hasEnterKeyUp;

        public static bool HasClicked { get; private set; }
        public static Vector2 MousePosition { get; private set; }
        public static bool IsKeyParticleRainPressed { get; private set; }
        public static bool IsKeyCometPressed { get; private set; }

        public static void Update()
        {
            // Souris
            var mouseState = Mouse.GetState();

            HasClicked = mouseState.LeftButton == ButtonState.Pressed && ↩
                _lastMouseState.LeftButton == ButtonState.Released;
            MousePosition = mouseState.Position.ToVector2();
            _lastMouseState = mouseState;

            // Espace
            if (Keyboard.GetState().IsKeyUp(Keys.Space))
            {
                _hasSpaceKeyUp = true;
            }
            if (Keyboard.GetState().IsKeyDown(Keys.Space) && _hasSpaceKeyUp)
            {
                IsKeyParticleRainPressed = true;
                _hasSpaceKeyUp = false;
            }
            else
            {
                IsKeyParticleRainPressed = false;
            }

            // Enter
            if (Keyboard.GetState().IsKeyUp(Keys.Enter))
            {
                _hasEnterKeyUp = true;
            }
            if (Keyboard.GetState().IsKeyDown(Keys.Enter) && _hasEnterKeyUp)
            {
                IsKeyCometPressed = true;
                _hasEnterKeyUp = false;
            }
            else
            {
                IsKeyCometPressed = false;
            }
        }
    }
}
```

Home.cs

```cs
using Microsoft.Xna.Framework;
using MiFiSy_TPI.UI;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml.Linq;
/*
 * Auteur : Yoann Meier
 * Date : 15/05/2024
 * Projet : Projet TPI, application de simulation de feux d'artifices en 2D
 * Description de la page : Classe de l'accueil
 */
namespace MiFiSy_TPI.Manager
{
    public class Home
    {
        private Button _btnPlay;
        private Dictionary<string, Button> _lstReplay;
        private List<Button> _lstBtnMusic;

        /// <summary>
        /// Nombre maximum de fichier afficher
        /// </summary>
        private const int NB_FILE_MAX = 10;

        /// <summary>
        /// Constructeur de la classe, récupère les musiques, les séquences sauvegardées des dossiers défini dans le fichier de configuration
        /// </summary>
        public Home()
        {
            _btnPlay = new Button(new Vector2(0.3f, 0.5f), 0.17f, 0.05f, "Commencer en mode libre", Color.Gray, Color.White, "play");
            _lstReplay = new Dictionary<string, Button>();
            _lstBtnMusic = new List<Button>();

            if (Directory.Exists(Config.PATH_SAVE_SEQUENCE))
            {
                string[] allReplay = Directory.GetFiles(Config.PATH_SAVE_SEQUENCE);
                if (allReplay.Length > NB_FILE_MAX)
                {
                    Array.Resize(ref allReplay, NB_FILE_MAX);
                }

                for (int i = 0; i < allReplay.Length; i++)
                {
                    string fileType = allReplay[i].Split(".")[1];
                    if (fileType == "xml")
                    {
                        string nameSequence = XDocument.Load(allReplay[i]).Descendants("FireworkSequence").FirstOrDefault().Attribute("name").Value;
                        _lstReplay.Add(allReplay[i], new Button(new Vector2(0.8f, Globals.ScreenHeight / (float)(allReplay.Length + 1) * (i + 1) / Globals.ScreenHeight), 0.1f, 0.05f, nameSequence, Color.Gray, Color.White, "playReplay"));
                    }
                }
            }

            if (Directory.Exists(Config.PATH_MUSIC))
            {
                string[] allMusic = Directory.GetFiles(Config.PATH_MUSIC);
                if (allMusic.Length > NB_FILE_MAX)
                {
                    Array.Resize(ref allMusic, NB_FILE_MAX);
                }

                for (int i = 0; i < allMusic.Length; i++)
                {
```

```
66                      string fileType = allMusic[i].Split(".")[1];
67                      if (fileType == "mp3" || fileType == "wav")
68                      {
69                          string fileName = allMusic[i].Split('/')[1];
70                          _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"));
71                      }
72                  }
73              }
74          }
75
76          public void Update()
77          {
78              _btnPlay.Update();
79              foreach (Button btnMusic in _lstBtnMusic)
80              {
81                  btnMusic.Update();
82                  if (btnMusic.IsPressed)
83                  {
84                      bool changeOk = true;
85                      // Si aucune musique n'est séléctionné
86                      if (Globals.MusicSelectedName != "")
87                      {
88                          Button btnSelected = _lstBtnMusic.Find(x => x.Text ==
                                Globals.MusicSelectedName);
89                          btnSelected.TextColor = Color.White;
90                          Globals.MusicSelectedName = "";
91                          // Si on a appuyé sur le même bouton, on ne le met pas en rouge, on
                                veut qu'il devienne blanc
92                          if (btnSelected.Text == btnMusic.Text)
93                          {
94                              changeOk = false;
95                          }
96                      }
97
98                      // Change la couleur en rouge si c'est une nouvelle musique qui est
                            séléctionné
99                      if (changeOk)
100                     {
101                         Globals.MusicSelectedName = btnMusic.Text;
102                         btnMusic.TextColor = Color.Red;
103                     }
104                 }
105             }
106
107             for (int i = 0; i < _lstReplay.Count; i++)
108             {
109                 Button btn = _lstReplay.ElementAt(i).Value;
110                 btn.Update();
111                 if (btn.IsPressed)
112                 {
113                     Globals.GameManager = new GameManager(false, "",
                            _lstReplay.ElementAt(i).Key);
114                     Globals.ActualPage = Globals.AllPage.Game;
115                 }
116             }
117         }
118
119         /// <summary>
120         /// Affiche tous les éléments de l'accueil
121         /// </summary>
122         public void Draw()
123         {
124             _btnPlay.Draw();
125             if (_lstBtnMusic.Count != 0)
126             {
127                 Globals.SpriteBatch.DrawString(Globals.DefaultFontButton, "Choisir une
                        musique : (optionnel)", new Vector2(_lstBtnMusic[0].Rectangle.X,
                        _lstBtnMusic[0].Rectangle.Y - 50), Color.White);
128             }
129             _lstBtnMusic.ForEach(x => x.Draw());
130             if (_lstReplay.Count != 0)
131             {
132                 Globals.SpriteBatch.DrawString(Globals.DefaultFontButton, "Revoir :", new
                        Vector2(_lstReplay.ElementAt(0).Value.Rectangle.X,
                        _lstReplay.ElementAt(0).Value.Rectangle.Y - 50), Color.White);
133             }
134             foreach (Button btn in _lstReplay.Values)
135             {
136                 btn.Draw();
137             }
138         }
139     }
140 }
```

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

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

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

```
                    )
                );
            }

            // Ajoute les feux d'artifices
            foreach (IFirework firework in Globals.LstFirework)
            {
                if (firework is Comet comet)
                {
                    XElement cometElement = CreateCommonFireworkElement(comet, ↩
                        ATTRIBUTE_TYPE_COMET);
                    cometElement.Add(
                        new XElement(ELEMENT_SIZE,
                            new XAttribute(ATTRIBUTE_MAIN_SIZE, Config.COMET_MAIN_SIZE),
                            new XAttribute(ATTRIBUTE_OTHER_SIZE, Config.COMET_OTHER_SIZE)
                        ),
                        new XElement(ELEMENT_START,
                            new XAttribute(ATTRIBUTE_POSITION_X, ↩
                                comet.StartPosition.X.ToString().Replace(".", ",")),
                            new XAttribute(ATTRIBUTE_POSITION_Y, ↩
                                comet.StartPosition.Y.ToString().Replace(".", ",")),
                            new XAttribute(ATTRIBUTE_ANGLE, ↩
                                comet.StartAngle.ToString().Replace(".", ",")),
                            new XAttribute(ATTRIBUTE_SPEED, ↩
                                comet.StartSpeed.ToString().Replace(".", ",")),
                            new XAttribute(ATTRIBUTE_LIFESPAN, ↩
                                comet.Lifespan.ToString().Replace(".", ","))
                        )
                    );
                    fireworkSequence.Add(cometElement);
                }
                else if (firework is ParticleRain rain)
                {
                    XElement rainElement = CreateCommonFireworkElement(rain, ↩
                        ATTRIBUTE_TYPE_PARTICLE_RAIN);
                    rainElement.Add(
                        new XElement(ELEMENT_SIZE, Config.PARTICLE_RAIN_SIZE),
                        new XElement(ELEMENT_START,
                            new XAttribute(ATTRIBUTE_POSITION_X, ↩
                                rain.StartPosition.X.ToString().Replace(".", ",")),
                            new XAttribute(ATTRIBUTE_POSITION_Y, ↩
                                rain.StartPosition.Y.ToString().Replace(".", ",")),
                            new XAttribute(ATTRIBUTE_SPEED, ↩
                                rain.StartSpeed.ToString().Replace(".", ",")),
                            new XAttribute(ATTRIBUTE_LIFESPAN, ↩
                                rain.Lifespan.ToString().Replace(".", ",")),
                            new XAttribute(ATTRIBUTE_NB_PARTICLE, Config.PARTICLE_RAIN_NB)
                        )
                    );
                    fireworkSequence.Add(rainElement);
                }
            }
            // Sauvegarde le fichier
            document.Save($"{Config.PATH_SAVE_SEQUENCE}{currentDate.ToString("yyyy-MM-dd ↩
                HH_mm_ss")}.xml");
            Globals.LstFirework.Clear();
            _timerLauch = 0f;
        }

        /// <summary>
        /// Crée les éléments communs au feu d'artifice comme la couleur et le temps de ↩
            lancement
        /// </summary>
        /// <param name="firework">feu d'artifice lancé, pour récupérer le launchTime</param>
        /// <param name="type">type de feu d'artifice : Comète, pluie de particule</param>
        private static XElement CreateCommonFireworkElement(IFirework firework, string type)
        {
            XElement baseParticle = new XElement(ELEMENT_FIREWORK,
                new XAttribute(ATTRIBUTE_TYPE, type),
                new XAttribute(ATTRIBUTE_LAUNCH_TIME, ↩
                    firework.LaunchTime.ToString().Replace(".", ","))
           );
            if (type == ATTRIBUTE_TYPE_COMET)
            {
                baseParticle.Add(
                    new XElement(ELEMENT_COLOR_START,
                        new XAttribute(ATTRIBUTE_R_COLOR, Config.COLOR_START_COMET.R),
                        new XAttribute(ATTRIBUTE_G_COLOR, Config.COLOR_START_COMET.G),
                        new XAttribute(ATTRIBUTE_B_COLOR, Config.COLOR_START_COMET.B)
                    ),
                     new XElement(ELEMENT_COLOR_END,
                         new XAttribute(ATTRIBUTE_R_COLOR, Config.COLOR_END_COMET.R),
                         new XAttribute(ATTRIBUTE_G_COLOR, Config.COLOR_END_COMET.G),
                         new XAttribute(ATTRIBUTE_B_COLOR, Config.COLOR_END_COMET.B)
                     )
                );
            }
```

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

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

```
446                 {
447                     Globals.SpriteBatch.Draw(_background, new Rectangle(0, 0, ↩
                            Globals.ScreenWidth, Globals.ScreenHeight), Color.White);
448                 }
449 
450                 // Affiche les données du replay
451                 Globals.SpriteBatch.DrawString(Globals.DefaultFontButton, $"Nom de la ↩
                        sequence : {_file.Attribute(ATTRIBUTE_NAME).Value}", new Vector2(0.75f * ↩
                        Globals.ScreenWidth, 0.05f * Globals.ScreenHeight), Color.White);
452                 Globals.SpriteBatch.DrawString(Globals.DefaultFontButton, $"Auteur : ↩
                        {_file.Attribute(ATTRIBUTE_AUTHOR).Value}", new Vector2(0.75f * ↩
                        Globals.ScreenWidth, 0.1f * Globals.ScreenHeight), Color.White);
453                 Globals.SpriteBatch.DrawString(Globals.DefaultFontButton, $"Date : ↩
                        {_file.Attribute(ATTRIBUTE_CREATION_DATE).Value}", new Vector2(0.75f * ↩
                        Globals.ScreenWidth, 0.15f * Globals.ScreenHeight), Color.White);
454 
455                 // Affiche un message de fin de replay
456                 if (_timerLauch >= float.Parse(_file.Attribute(ATTRIBUTE_TIME_END).Value))
457                 {
458                     Globals.SpriteBatch.DrawString(Globals.DefaultFontButton, "Fin du ↩
                            replay", new Vector2(0.5f * Globals.ScreenWidth, 0.5f * ↩
                            Globals.ScreenHeight), Color.Red);
459                 }
460             }
461 
462             _menuButton.Draw();
463             _lstMortar.ForEach(m => m.Draw());
464         }
465     }
466 }
```