

Manuel Technique



Travail Pratique Individuel MiFiSy

CFPT Informatique

Yoann Meier

15 mai 2024

Table des matières

Introduction	3
Glossaire	3
Résumé du cahier des charges	4
Description de l'application	4
Framework/librairie	5
Outils/logiciels utilisés	5
Livrables	5
Méthodologie	6
Méthodes en 6 étapes	6
Analyse fonctionnelle	7
Organisation des fichiers	7
Vue Accueil	7
Vue Jeu	8
Mode libre	8
Mode replay	9
Analyse organique	10
Structure des fichiers	10
Diagramme de classe	11
Diagramme de séquence	12
Fichier de configuration	13
Description des classes et des méthodes importantes du projet	14
Config.cs	14
Globals.cs	15
Game1.cs	15
ParticleData.cs	16
ParticleEmitterData.cs	16
Particle.cs	16
ParticleEmitter.cs	17
ParticleManager.cs	17
Button.cs	19
Mortar.cs	20
IFirework.cs	20
ParticleRain.cs	21
Comet.cs	23
JamstikMidiListener.cs	23
InputManager	24
Home.cs	24
GameManager.cs	25
Plan de test et tests	26
Périmètre de test	26
Plan de test	26
Evolution des tests	27

Conclusion	28
Analyse du planning effectif	28
Difficultés rencontrées	28
Amélioration possible	28
Bilan personnel	28
Annexes	29
Planning prévisionnel	29
Planning effectif	30

Introduction

Ce document est un rapport présentant la conception du projet MiFiSy (MIDI Firework Symphony).
Ce projet est réalisé dans le cadre du projet de fin de formation au CFPT Informatique dans la formation *Développement d'applications*.

Il s'agit du *Travail Pratique individuel* (TPI).

Le TPI est un travail de validation de ma formation réalisé en fin de formation qui est nécessaire à l'obtention de mon diplôme : le CFC (*Certificat fédéral de capacité*)

MiFiSy est un projet monogame permettant de créer des feux d'artifice virtuel à l'aide d'une guitare MIDI.

Une musique de fond peut être ajoutée.

La séquence d'effet de feu d'artifice peut être sauvegardée au format XML et rejouée dans l'application.

Mon maître d'apprentissage est M. Garchery et mes experts sont M. Nydegger et Mme Travnjak.

Glossaire

- Mortier
Il s'agit d'un cylindre en carton, en plastique ou en fibre de verre, de diamètre variable, qui sert de guide aux bombes de feu d'artifice portant de faibles charges explosives, mais suffisantes pour être mortelles ([lien de la définition](#)).
- Guitare MIDI
La guitare MIDI est une guitare permettant de récupérer les notes jouées grâce à des capteurs de son intégrés et dont les informations sont envoyées à l'ordinateur par USB en format MIDI.
- Format MIDI
Le format MIDI (Musical Instrument Digital Interface) est un format de fichier destiné à la musique qui permet de communiquer entre un instrument de musique et l'ordinateur. Il permet de récupérer les notes jouées mais également la vitesse à laquelle une note est jouée.
- Pluie de particules (saule pleureur)
Le saule pleureur est un feu d'artifice où un feu d'artifice monte dans le ciel, puis explose en de multiples branches qui vont dans toutes les directions et qui, au bout d'une ou deux secondes, redescendent sous l'effet de gravité et disparaissent ([exemple vidéo](#)).
- Comètes
Un feu d'artifice part d'un mortier et avance en ligne droite, puis il disparaît après un moment ([exemple vidéo](#)).

Résumé du cahier des charges

Description de l'application

- Page d'accueil
 - Liste des musiques disponibles
 - Liste des séquences sauvegardées
 - Bouton pour aller dans le mode libre (musique optionnelle)
 - Bouton pour aller dans le mode replay (choix d'une séquence obligatoire)
- Page de jeu, mode libre
 - 5 mortiers sont disposés uniformément au bas de l'écran avec un angle entre -10 et 10 degrés
 - Bouton de retour à l'accueil
 - Bouton de sauvegarde de la séquence de feu d'artifice au format XML
 - Jouer une corde déclenche un effet de feu d'artifice, la vitesse influe sur la vitesse ou la taille du feu d'artifice
 - Effet de comète :
Une traînée lumineuse propulsée par un des 5 mortiers choisis aléatoirement.
La direction de la traînée est en fonction de l'angle du mortier.
 - Effet de pluie de particules :
Des particules sont générées sur un point en haut de l'écran, celles-ci tombent avec l'effet de la gravité et disparaissent après un certain temps ou lorsqu'elles atteignent le bas de l'écran.
- Page de jeu, mode replay
 - La séquence jouée choisit dans l'accueil est jouée
 - Les mortiers possèdent le même angle que dans la séquence
 - Bouton de retour à l'accueil
 - Les informations de la séquence (auteur, date, nom de la séquence) sont affichées
- Fichier de configuration

Un fichier xml permet de configurer l'application comme la position des mortiers, le dossier de sauvegarde des séquences...

Framework/librairie

Dans ce projet, j'utilise le framework Monogame pour développer mon application. Monogame est un framework C# permettant de faire des jeux.

J'utilise également la librairie NAudio permettant de récupérer les cordes jouées par une guitare MIDI dans le format MIDI.

Outils/logiciels utilisés



Ordinateur Windows 10



C# - Visual Studio 2022



Framework Monogame



GitHub



Google Drive



TexStudio



Suite office

Livrables

- Manuel technique
- Manuel utilisateur
- Journal de bord
- Rapport TPI
- Le projet

Méthodologie

Méthodes en 6 étapes

Pour assurer le bon déroulement de mon projet, j'ai utilisé une méthodologie de travail afin d'être organisé et efficace.

Après avoir examiné différentes méthodes lors de ma formation, j'ai opté pour la méthode en 6 étapes.

Je l'ai choisie, car, parmi les différentes méthodes étudiées, elle s'est révélée être la plus adaptée pour un travail individuel à court terme.

- **S'informer**

Au début du projet, j'ai analysé le cahier des charges afin de comprendre toutes les fonctionnalités à réaliser.

J'ai également demandé des précisions à mon formateur sur certains points que je n'ai pas compris.

- **Planifier**

Une fois le cahier des charges compris, j'ai préparé le planning de mon travail en découpant le travail par tâches avec une durée prévisionnelle.

Voir : [Planning prévisionnel](#)

- **Décider**

Dans la phase de décision, j'ai décidé l'ordre de réalisation de toutes les tâches en fonction de l'importance de celle-ci sur le projet.

- **Réaliser**

Cette étape est cruciale dans mon projet, car elle implique la concrétisation de mon travail en suivant la planification établie précédemment. Grâce à une bonne compréhension des tâches à accomplir, j'ai pu effectuer mon travail dans des conditions optimales.

- **Contrôler**

Cette étape est importante, car elle consiste à contrôler le bon avancement du projet en s'assurant que les objectifs du travail sont atteints. À chaque tâche terminée, j'ai effectué des tests pour vérifier le bon fonctionnement de ces tâches.

Voir : ??

- **Évaluer**

Pour finir, j'ai évalué tous les résultats obtenus et j'ai également réfléchi aux différents moyens d'améliorer mon travail, que ce soit pour ajouter de nouvelles fonctionnalités ou améliorer mon code.

Voir : [Amélioration possible](#)

Analyse fonctionnelle

Organisation des fichiers

Le fichier config.xml doit être dans le même répertoire que le fichier exécutable du projet.

Le dossier de sauvegarde et celui contenant les musiques peut se trouver n'importe où sur le disque, mais le chemin doit être modifié dans config.xml.

Vue Accueil

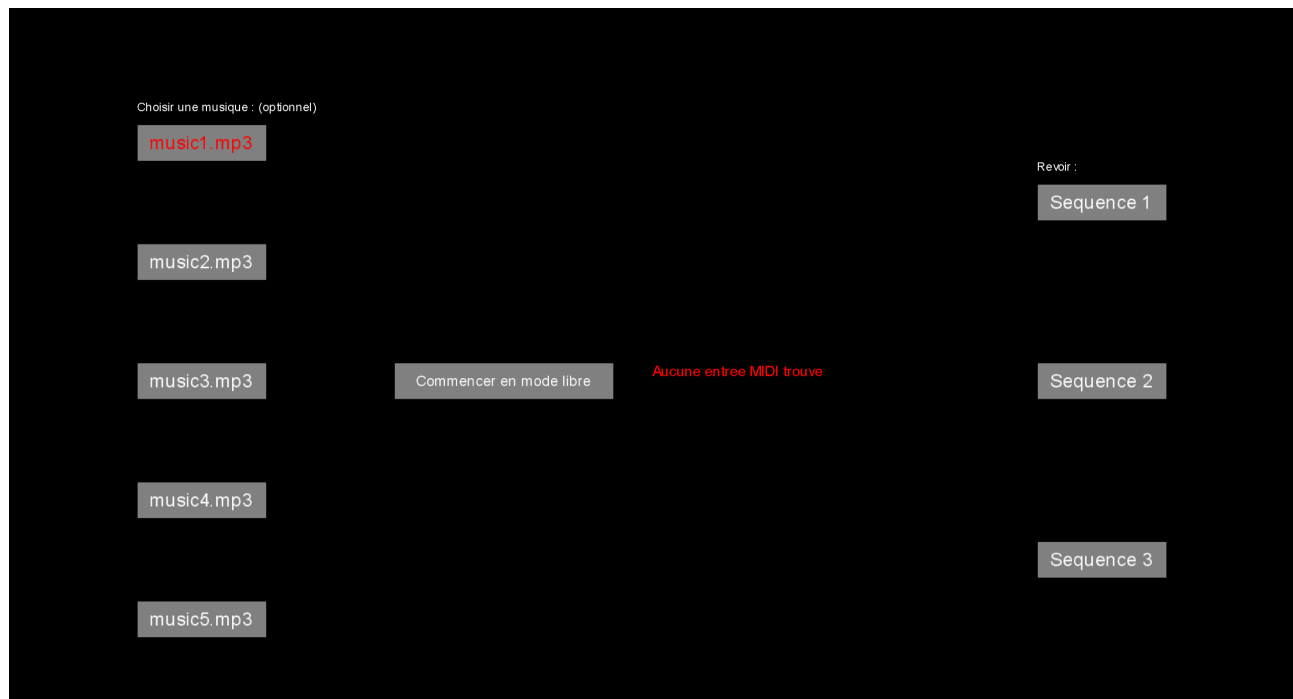


FIGURE 1 – page de démarrage de l'application

Dans cette page, un message d'erreur apparaît au milieu de l'écran si aucune entrée MIDI n'est détectée : **"Aucune entrée MIDI trouvée"**.

Si l'utilisateur souhaite une musique de fond pendant le mode libre, il peut appuyer sur l'une des musiques affichée à gauche. Chaque musique provient d'un dossier qui peut être défini dans le fichier de configuration. Lorsqu'une musique est sélectionnée, sa couleur deviendra rouge (music1.mp3 sur l'exemple).

L'utilisateur peut appuyer sur le bouton **"Commencer en mode libre"**, ce qui le redirigera vers la page de jeu en mode libre.

À droite de l'écran, les différentes séquences précédemment enregistrées sont affichées.

Lors d'un clique sur l'une d'entre elle, l'utilisateur est redirigé sur la page de jeu en mode replay.

Vue Jeu

Mode libre

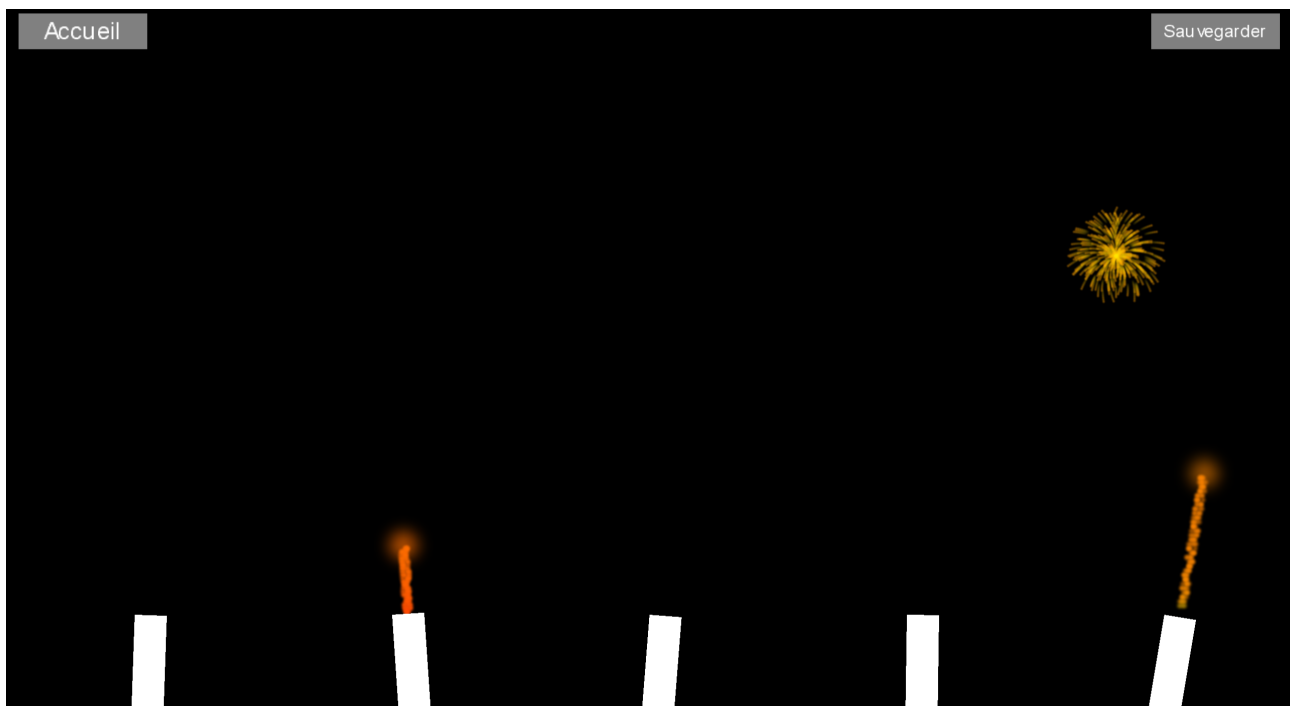


FIGURE 2 – Mode de jeu libre

Dans cette page, un bouton "**Accueil**" permet de retourner à l'accueil, ce qui remet à 0 la séquence de feu d'artifice.

Plusieurs mortiers, définis dans le fichier de configuration (position x et y, taille en hauteur et en largeur et l'angle d'inclinaison) sont affichés.

Si aucun mortier n'est défini dans le fichier de configuration, 5 mortiers sont disposés uniformément en bas de l'écran avec la même taille et un angle entre -10 et 10 degrés.

Si l'utilisateur (connecté à une guitare MIDI) joue la première corde (tout en bas), la comète sera lancée aléatoirement de l'un des mortier représenté par un rectangle blanc, avec la même direction.

La vitesse de déplacement dépend de la vélocité à laquelle la corde est jouée.

Si l'utilisateur joue la deuxième corde, la pluie de particules est créée sur un point aléatoire du haut de l'écran. La durée de vie du feu d'artifice dépend de la vélocité à laquelle la corde est jouée.

Le bouton en haut à droite permet de sauvegarder la séquence de feu d'artifice créée au format XML.

Un message de confirmation de sauvegarde apparaît brièvement sur l'écran.

Mode replay



FIGURE 3 – Mode de jeu replay

Dans cette page, une séquence sauvegardée est jouée, la musique, l'image de fond et la position des mortiers est identique, les informations de la séquence sont écrits en haut de l'écran.

A la fin de la séquence, un message ("**Sauvegarde effectue**") apparaît pour notifier la fin de l'enregistrement à l'utilisateur.

Le bouton "**Accueil**" permet de revenir à l'accueil.

Analyse organique

Structure des fichiers

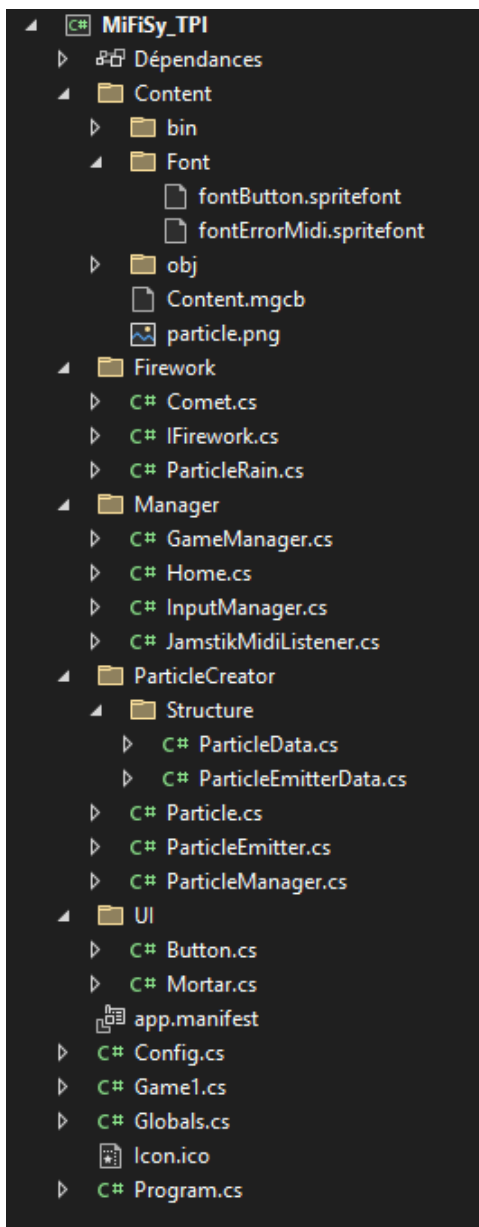


Diagramme de séquence

Ce diagramme de séquence montre le fonctionnement global de l'application, il ne rentre pas dans les détails comme la musique et l'image de fond pour rester simple.

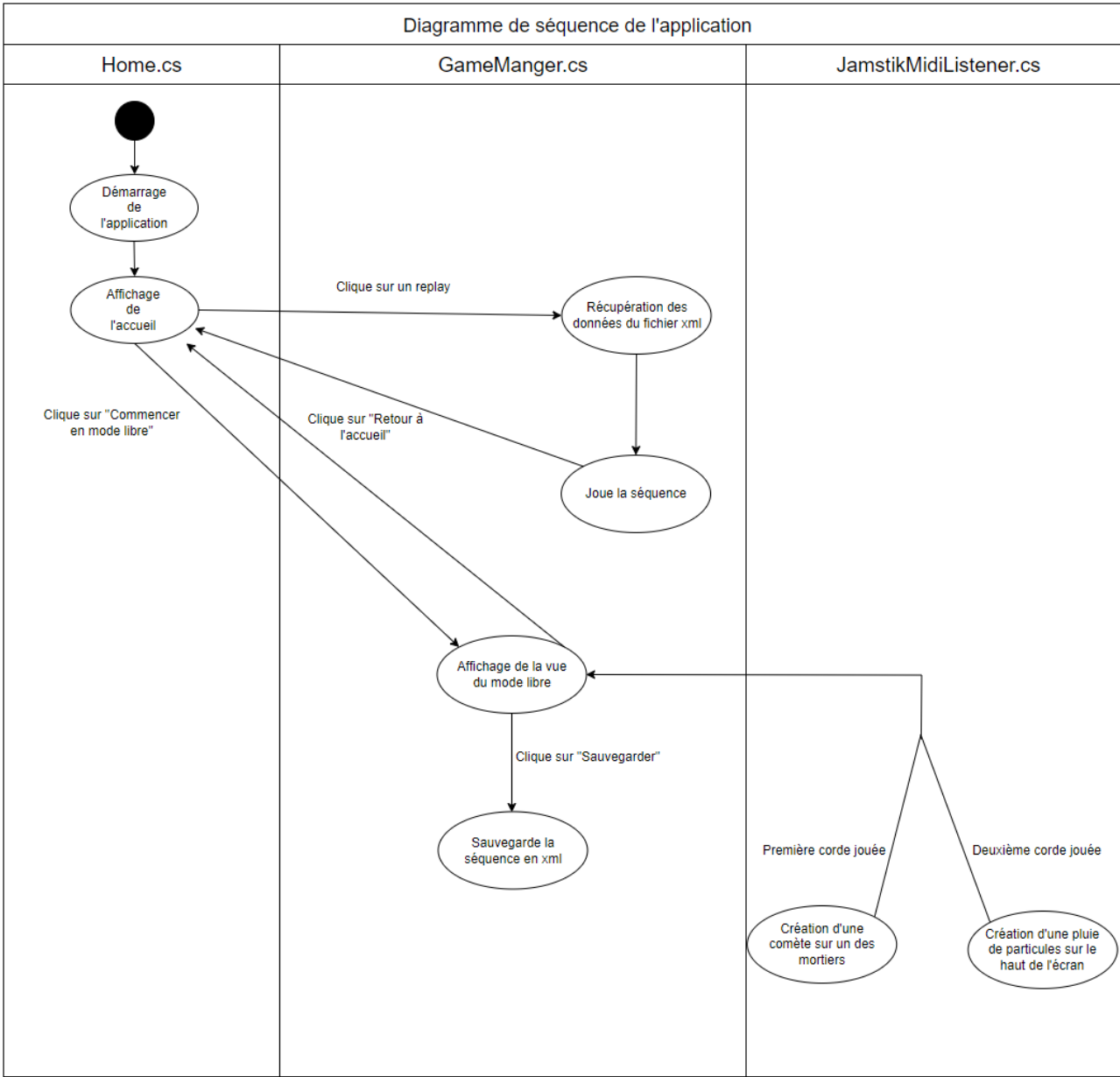


FIGURE 5 – Diagramme de séquence

Fichier de configuration

Le fichier config.xml permet de paramétrer des valeurs importantes de l'application.
Il doit être dans le même dossier que l'exécutable pour que le projet fonctionne.
Voilà à quoi il ressemble :

```

1 <Config>
2   <Author>Yoann</Author>
3   <NameSequence>Sequence de feu d artifice</NameSequence>
4   <PathMusic>music</PathMusic>
5   <PathImg></PathImg>
6   <PathSaveSequence>save</PathSaveSequence>
7   <Mortar angle="10" height="0,15" width="0,025" positionX="0,1" positionY="0,85"/>
8   <Mortar angle="10" height="0,15" width="0,025" positionX="0,3" positionY="0,85"/>
9   <Mortar angle="10" height="0,15" width="0,025" positionX="0,5" positionY="0,85"/>
10  <Mortar angle="10" height="0,15" width="0,025" positionX="0,7" positionY="0,85"/>
11  <Mortar angle="10" height="0,15" width="0,025" positionX="0,9" positionY="0,85"/>
12  <ColorStartParticleRain r="255" g="69" b="0"/>
13  <ColorStartComet r="255" g="69" b="0"/>
14  <ColorEndParticleRain r="255" g="255" b="0"/>
15  <ColorEndComet r="255" g="255" b="0"/>
16  <ParticleRain sizeParticle="10" nbParticle="360" lifeSpan="0,06" timeSpawn="0,04" ↔
    defaultSpeed="90"/>
17  <Comet sizeMainParticle="120" sizeOtherParticle="20" defaultSpeed="7" ↔
    defaultLifespan="1,5" />
18 </Config>

```

- **Author**
(texte) Nom de l'auteur qui sera stocké dans les prochaines sauvegardes de feux d'artifice.
- **NameSequence**
(texte) Nom de la séquence qui sera stocké dans les prochaines sauvegardes de feux d'artifice.
- **PathMusic**
(texte) Chemin d'accès du dossier contenant les musiques qui apparaîtront dans l'accueil (10 seront affichés au maximum).
- **PathImg**
(texte) Chemin d'accès de l'image qui apparaîtra dans le jeu en mode libre.
- **PathSaveSequence**
(texte) Chemin où les sauvegarde sont récupérés et où les prochaines seront ajoutées (10 seront affichés au maximum).
- **Mortar**
Si aucun mortier n'est précisé, 5 seront posé dans le mode libre.
Sinon, les mortiers spécifiés ici seront affichés comme indiquer ici dans le jeu en mode libre.
Les cinq paramètres sont :
 - (nombre réel) L'angle
 - (nombre réel) La hauteur
 - (nombre réel) La largeur
 - (nombre réel) La position sur l'axe X et Y
- **ColorStartParticleRain**
Couleur de départ des particules des pluie de particules dans le jeu en mode libre.
Les attributs (nombre entier) corresponde à RGB pour crée la couleur.
- **ColorStartComet**
Couleur de fin des particules des comètes dans le jeu en mode libre.
- **ColorEParticleRain**
Couleur de fin des particules des pluie de particules dans le jeu en mode libre.
- **ColorEndComet**
Couleur de départ des particules des comètes dans le jeu en mode libre.

- **ParticleRain**

Paramètre de la pluie particule dans le jeu en mode libre.

- (**nombre entier**) sizeParticle : la taille des particules générés
- (**nombre entier**) nbParticle : le nombre de particules en mouvement qui seront générés équitablement dans toutes les directions
- (**nombre à virgule**) lifespan : la durée de vie par défaut de la particule. Cette valeur est multiplié par la vitesse de la corde de la guitare
- (**nombre à virgule**) timeSpawn : le temps entre les apparitions des particules immobile sur celles en mouvements
- (**nombre à virgule**) defaultSpeed : la vitesse des particules

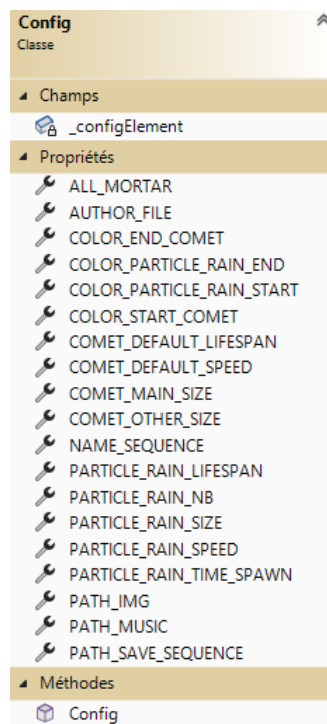
- **Comète**

Paramètre de la comète dans le jeu en mode libre.

- (**nombre entier**) sizeMainParticle : la taille de la particule principale de la comète
- (**nombre entier**) sizeOtherParticle : la taille des particules qui suivent la particule principale
- (**nombre à virgule**) defaultSpeed : la vitesse de départ qui est multiplié par la vitesse de la corde de la guitare
- (**nombre à virgule**) defaultLifespan : la durée de vie de la comète

Description des classes et des méthodes importantes du projet

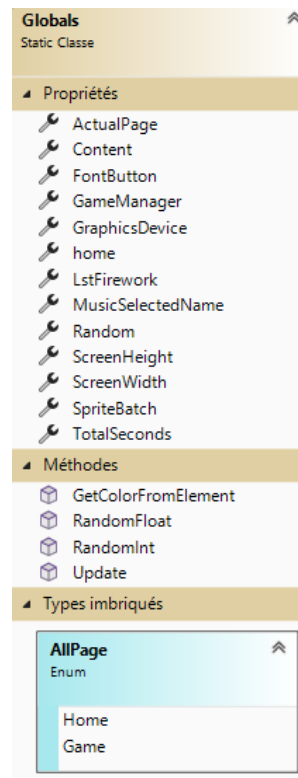
Config.cs



Cette classe sert à récupérer les données du fichier de configuration.

Elle contient des propriété static permettant seulement de récupérer une information du fichier, comme l'auteur des feux d'artifices par exemples.

Elle contient aussi un constructeur pour ouvrir et récupérer les données du fichier, ce qui évite d'ouvrir le fichier à chaque fois que l'on veut récupérer une de ses valeurs.

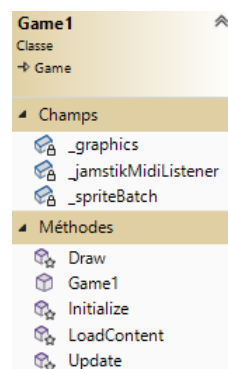


Globals.cs

Cette classe permet de récupérer des valeurs static qui sont nécessaires dans énormément de classes comme la hauteur et la largeur de l'écran, le SpriteBatch, une classe qui permet d'afficher des éléments, image ou texte, à l'écran et dont je ne possède qu'une seule instance (Détail sur le site de monogame : [SpriteBatch](#))

Cette page contient aussi des propriétés static très peu utilisées, mais qui sont utilisées dans des classes très séparés et différentes comme par exemple de r instanciée la classe de l'accueil depuis la classe Bouton, lors du retour à l'accueil depuis le jeu.

Game1.cs



C'est la classe principale du projet qui est créé par défaut lorsqu'on crée un projet.

Elle hérite de la classe de Monogame [Game](#) et contient plusieurs méthodes essentielles au fonctionnement du projet comme **Initialize** et **LoadContent** au lancement du projet et surtout **Update**, une méthode qui est appelée de nombreuses fois chaque secondes et qui permet de mettre à jour les informations.

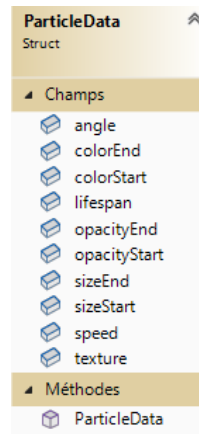
Finalement, il y a **Draw** qui permet d'afficher les éléments du jeu et qui lui aussi est appelé très souvent.

Plus d'informations [ici](#).

Dans mon projet, toutes mes classes qui doivent mettre à jour ou afficher des éléments sont appelés dans l'Update et le Draw de Game1.cs.

Une enum permet de mettre à jour et d'afficher les éléments correspondants à la page actuelle : l'accueil ou le jeu.

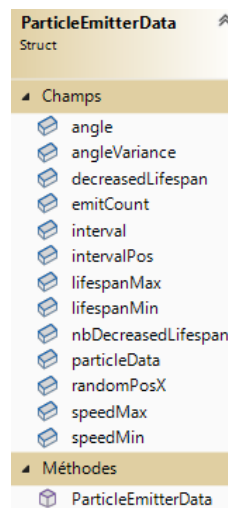
ParticleData.cs



C'est une **struct** qui stock des informations concernant une particule : la texture, la couleur, la taille, la vitesse, l'angle...

Cela provient d'un [tutoriel](#) pour créer un système de particule.

ParticleEmitterData.cs



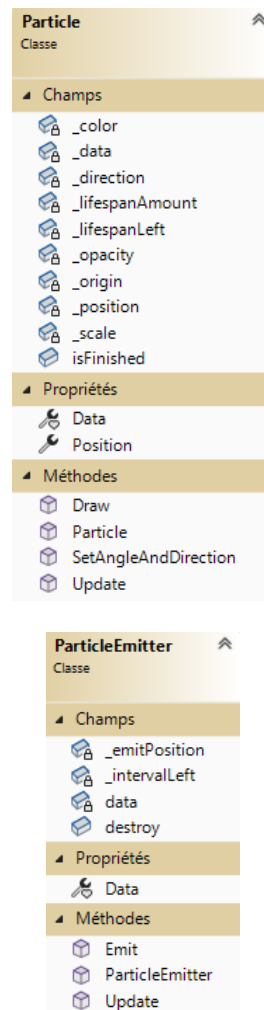
C'est une **struct** qui stock des informations permettant d'émettre des particules avec un ParticleData, l'intervall d'émission, le nombre à émettre, un angle ou une vitesse aléatoire, etc.

Proviens également du tutoriel, mais que j'ai légèrement modifié.

Particle.cs

Cette classe permet de créer une particule, de lui donner une durée de vie, de se déplacer, de changer de couleur en fonction de la durée de vie.

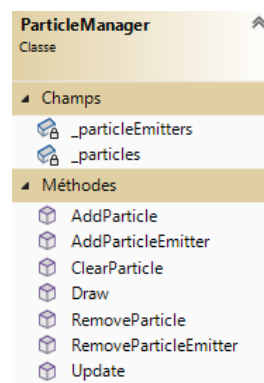
Cette classe utilise le struct [ParticleData.cs](#) et vient du même tutoriel.



ParticleEmitter.cs

Cette classe émet des particules selon les informations du **ParticleEmitterData.cs** qu'il possède. Les particules sont émises à une position défini dans les paramètres du constructeur. Cette classe provient également du tutoriel de système de particules.

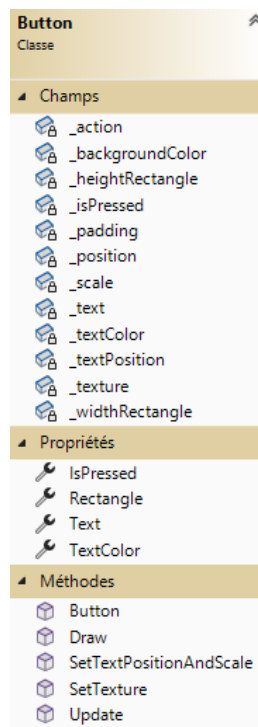
ParticleManager.cs



Cette classe gère les particules avec simplement une liste de particules et une liste de `particleEmitter`, avec des méthodes static pour ajouter et supprimer ainsi qu'une méthode "Update" et "Draw" pour afficher et mettre à jour les particules et les émetteurs.

Viens également du tutoriel.

Button.cs



Cette classe permet simplement de créer un bouton un champ string "action" permet de savoir quelle action le clique réalise avec par exemple "goBack" pour revenir à l'accueil ou encore "play" pour aller depuis l'accueil dans le jeu en mode libre.

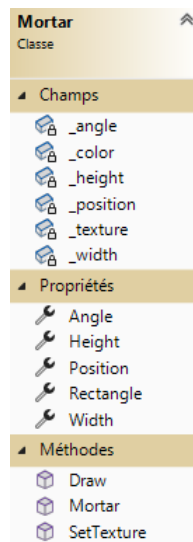
La fonction "SetTextPositionAndScale" permet de calculer la taille et la position du texte dans le rectangle en fonction du nombre de caractères récupérés avec la fonction "MeasureString" de la classe SpriteFont pour récupérer la taille du texte en X et en Y.

SpriteFont est une classe pour charger une police de texte et l'utiliser pour afficher du texte ([documenta-
tion](#)).

```

69     /// <summary>
70     /// Calcule la position et la taille du text par rapport à la largeur du rectangle ↔
71     /// qu'il contient
72     /// </summary>
73     public void SetTextPositionAndScale()
74     {
75         if (Globals.FontButton.MeasureString(_text).X != 0)
76         {
77             // Calcul du facteur d'échelle pour le texte
78             float scaleX = (Rectangle.Width * (1 - _padding)) / ↔
79                 Globals.FontButton.MeasureString(_text).X;
80             float scaleY = (Rectangle.Height * (1 - _padding)) / ↔
81                 Globals.FontButton.MeasureString(_text).Y;
82             _scale = Math.Min(scaleX, scaleY);
83         }
84         else
85         {
86             _scale = 1;
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     }

```

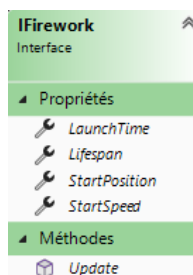


Mortar.cs

Cette classe permet simplement d'afficher un rectangle avec une rotation pour simuler un mortier qui tire des feux d'artifices.

J'ai créé cette classe pour la sauvegarde, car la position, la taille et l'angle des mortiers doit être identique dans le mode replay.

IFirework.cs

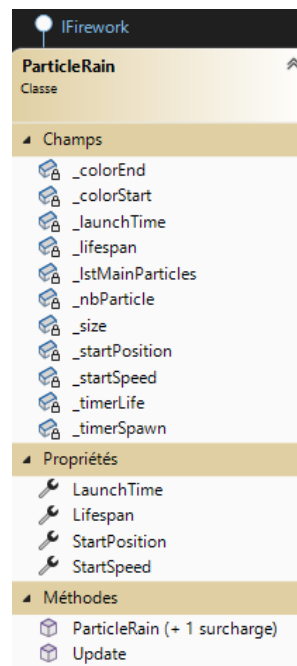


C'est une interface qui possède des propriétés obligatoires à chaque feu d'artifice : une position, une durée de vie, le moment dans le temps où il a été lancé (pour refaire la séquence au bon moment dans le replay) et une vitesse.

Mais le plus important est la fonction "Update" qui permet de mettre à jour l'effet de particule.

Le but de cette interface est de n'avoir qu'une seule liste de feu d'artifice et non une seule liste par type de feu d'artifice, ce qui évite la redondance de code et simplifie le travail.

ParticleRain.cs



Cette classe permet de créer le feu d'artifice de la pluie de particules, elle implémente **IFirework.cs**. Je vais expliquer comment cela fonctionne.

Tout d'abord, voici mon constructeur dans le mode libre (celui pour le mode replay est presque identique) :

```

34     /// <summary>
35     /// Constructeur de la classe utilisée dans le jeu libre : créer la pluie de ←
36     /// particule en fonction de paramètre du fichier de configuration
37     /// </summary>
38     /// <param name="speed">vitesse de départ du feu d'artifice</param>
39     /// <param name="lifespan">durée de vie du feu d'artifice</param>
40     /// <param name="launchTime">Le temps à laquelle l'effet a été créée, seulement ←
41     /// utilisé pour la sauvegarde</param>
42     /// <param name="distanceFromBorder">distance pour ne pas créer la particule en ←
43     /// dehors ou sur le bord de l'écran</param>
44     public ParticleRain(float speed, float lifespan, float launchTime, float ←
45     distanceFromBorder = 100)
46     {
47         LaunchTime = launchTime;
48         Lifespan = lifespan;
49         StartSpeed = speed;
50         _nbParticle = Config.PARTICLE_RAIN_NB;
51         _colorStart = Config.COLOR_PARTICLE_RAIN_START;
52         _colorEnd = Config.COLOR_PARTICLE_RAIN_END;
53         _size = Config.PARTICLE_RAIN_SIZE;
54
55         _timerLife = 0;
56         _timerSpawn = 0;
57         // Position aléatoire du feu d'artifice sur la partie haute de l'écran
58         StartPosition = new Vector2(Globals.RandomFloat(distanceFromBorder, ←
59         Globals.ScreenWidth - distanceFromBorder) / Globals.ScreenWidth, ←
60         Globals.RandomFloat(distanceFromBorder, Globals.ScreenHeight / 2) / ←
61         Globals.ScreenHeight);
62         _lstMainParticles = new List<Particle>();
63
64         for (int i = 0; i < _nbParticle; i++)
65         {
66             float angle = 360 / _nbParticle * i;
67             // Vitesse aléatoire entre 0 et le maximum
68             float newSpeed = Globals.RandomFloat(0, speed);
69             ParticleData particleData = new ParticleData()
70             {
71                 angle = angle,
72                 speed = newSpeed,
73                 colorStart = _colorStart,
74                 colorEnd = _colorEnd,
75                 sizeStart = _size,
76                 sizeEnd = _size,
77                 lifespan = Lifespan,

```

```

71         };
72         Particle p = new Particle(StartPosition, particleData);
73         _lstMainParticles.Add(p);
74         ParticleManager.AddParticle(p);
75     }
76 }

```

Dans ce code, j'initialise les variables et choisis une position aléatoire.

Ensuite, je crée les particules qui seront en mouvement. Le nombre, la couleur et la taille sont définis dans le fichier de configuration.

La vitesse est aléatoire et les particules sont réparties sur 360 degrés.

Voici ma méthode "Update" qui est appelée plusieurs fois par secondes :

```

126     /// <summary>
127     /// Méthode qui fais apparaître de nouvelles particules si la durée de vie n'est ←
128     /// pas atteinte, sinon supprime les particules
129     /// </summary>
130     public void Update()
131     {
132         _timerLife += Globals.TotalSeconds;
133         _timerSpawn += Globals.TotalSeconds;
134         // Supprime en fin de vie
135         if (_timerLife >= Lifespan)
136         {
137             _lstMainParticles.Clear();
138         }
139         if (_lstMainParticles.Count != 0)
140         {
141             if (_timerSpawn >= Config.PARTICLE_RAIN_TIME_SPAWN)
142             {
143                 // Ajoute une particule immobile sur chaque particule en mouvement
144                 for (int i = 0; i < _nbParticle; i++)
145                 {
146                     ParticleData particleData = new ParticleData()
147                     {
148                         angle = MathHelper.ToDegrees(_lstMainParticles[i].Data.angle),
149                         speed = 0,
150                         colorStart = _colorStart,
151                         colorEnd = _colorEnd,
152                         sizeStart = _size,
153                         sizeEnd = _size,
154                         lifespan = Lifespan - _timerLife,
155                     };
156                     Particle p = new Particle(_lstMainParticles[i].Position, ←
157                                             particleData);
158                     ParticleManager.AddParticle(p);
159                 }
160                 _timerSpawn = 0;
161             }
162             // Si un tiers du temps total est passé, les particules en mouvement tombent
163             if (_timerLife >= Lifespan / 3)
164             {
165                 foreach (Particle item in _lstMainParticles)
166                 {
167                     ParticleData data = item.Data;
168                     int angleAdd = MathHelper.ToDegrees(data.angle) < 180 ? 1 : -1;
169                     data.angle = MathHelper.ToDegrees(data.angle) + angleAdd;
170                     item.Data = data;
171                     item.SetAngleAndDirection();
172                 }
173             }
174         }
175     }

```

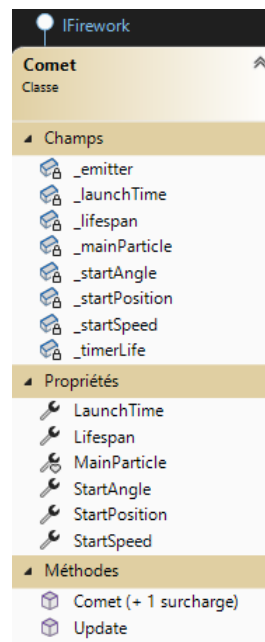
Tout d'abord, j'augmente les compteurs et je vide les listes de particules si la durée de vie est terminée.

Ensuite, je vais faire apparaître une particule immobile à la même position que chaque particule en mouvement avec un délai défini dans le fichier de configuration et une durée de vie en fonction du temps passé pour que la fin du feu d'artifice soit en même temps.

Finalement, si le tiers de la durée de vie est atteint, je vais déplacer l'angle des particules en mouvement d'un degré pour simuler la gravité.

Les prochaines particules seront donc décalées d'un degré par rapport au précédent et cela crée donc un effet de chute des particules et de mouvements.

Comet.cs

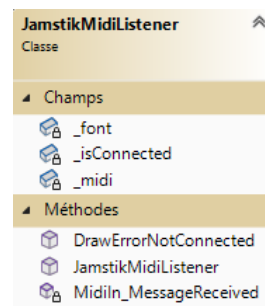


Cette classe crée une comète.

Il y a une particule principale qui part et un émetteur qui suit la particule principale.

La position et l'angle viennent d'un des mortiers, le choix et l'aléatoire sont dans le [GameManager.cs](#).

JamstikMidiListener.cs

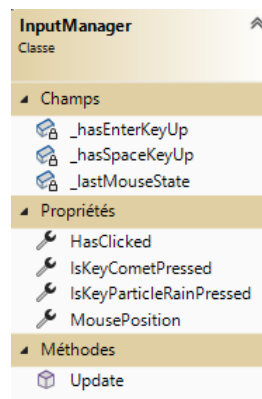


Cette classe gère la connexion avec la guitare MIDI.

Dans le constructeur, on parcourt les connexions et se connecter à "Jamstick", le nom de la connexion de la guitare. Ensuite, à chaque événement déclencher, une méthode reçoit ces événements, regarde si c'est une note, si elle vient de sonner, récupère la corde et appelle une fonction pour créer le feu d'artifice lié à cette corde.

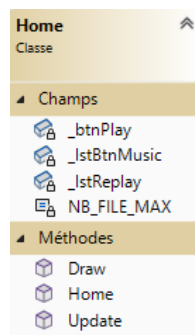
Si aucune connexion n'est trouvée, une fonction affiche un message d'erreur dans l'accueil.

InputManager



Cette classe permet simplement de gérer les cliques avec la souris pour les boutons et pouvoir récupérer l'état du clique partout avec une propriété static.

Home.cs



Cette classe contient l'accueil de l'application.

La classe Home gère l'interface utilisateur de l'accueil de l'application, choisir une musique, aller en mode libre et charger des séquences enregistrées.

Les musiques sont dans une liste de boutons. Je peux ensuite récupérer la musique dans le jeu libre avec le nom du bouton.

Alors que pour les enregistrements, j'utilise un dictionnaire avec comme clé un string qui contient le nom du fichier (et qui est donc forcément unique) et comme valeur un bouton avec comme texte le nom de la séquence récupéré dans le fichier de la séquence.

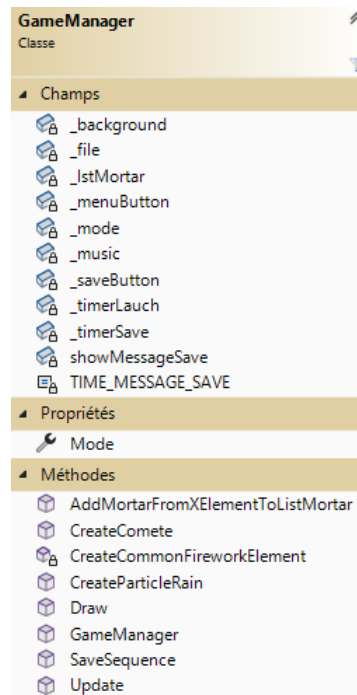
J'utilise ce dictionnaire pour pouvoir transmettre le nom du fichier dans le **GameManager.cs** et récupérer le contenu.

En effet, le nom de la séquence peut être le même pour plusieurs séquences, car il est défini lors de la sauvegarde et récupéré du fichier de configuration.

Seules les 10 premières musiques et les 10 premières séquences sont chargées.

Un message d'erreur est affiché au milieu de l'écran si la guitare MIDI n'a pas été trouvée au lancement.

GameManager.cs



Cette classe gère le jeu, le mode libre comme le mode replay.

Cette classe fait donc l'affichage des deux modes et charge les musiques et l'image de fond si spécifiée dans le fichier de configuration pour le mode libre ou dans la sauvegarde de la séquence pour le mode replay.

Cette classe contient les méthodes de création de feu d'artifice utilisées dans [JamstikMidiListener.cs](#) : "**CreateComete**" et "**CreateParticleRain**" qui ajoute le nouveau feu d'artifice dans une liste d'IFirework en prenant en compte une vitesse transmise en paramètre.

La liste sert à la sauvegarde.

- **Méthode "SaveSequence"**

Cette méthode sauvegarde donc la séquence dans le mode libre.

Tout d'abord, elle crée les informations de base de la séquence : son nom, l'auteur, la durée totale de la séquence, la date actuelle et possiblement une image ou une musique.

Ensuite, on ajoute les mortiers avec leurs positions, leurs angles, leurs hauteurs et leurs largeurs.

Finalement, je parcours la liste des feux d'artifices et les ajoute dans mon xml avec comme données principale le type (comète ou pluie de particules), le temps de lancement du feu d'artifice (un chronomètre se lance à l'arrivée dans le mode libre), leurs positions, etc.

Pour les données de type "float", je remplace les points par des virgules car les points créent une erreur : ""

- **Reproduction de la séquence**

Pour reproduire la séquence, un chronomètre se lance et reproduit les feux d'artifice lorsque le chronomètre est identique à la valeur stockée dans chaque feu d'artifice du fichier xml.

Plan de test et tests

Périmètre de test

Pour MiFiSy, je vais créer un plan de test visant à garantir le bon fonctionnement de l'application du point de vue de l'utilisateur.

Ce plan inclura des tests fonctionnels pour évaluer à la fois la performance et la convivialité de l'interface utilisateur.

Ensuite, je consignerai tous les tests réalisés ainsi que leurs résultats dans un tableau pour assurer la qualité de l'application et suivre l'évolution du projet.

Plan de test

N°	Description du test	Résultat attendu
1	Lors d'un clique sur une musique d'ambiance	La couleur du nom de la musique change et la musique est sélectionnée
2	Lors d'un clique sur une musique d'ambiance puis sur le mode libre dans l'accueil	La musique sélectionnée est jouée en boucle dans le mode libre
3	Lors d'un clique sur le mode libre dans l'accueil sans cliquer sur une musique	Le mode libre se lance sans musique
4	Lors d'un clique sur un replay dans l'accueil	Le replay se lance, la musique et les effets sont identiques
5	Lorsque le replay est terminé	Un message l'indiquant apparaît à l'écran
6	Lors du clique sur 'Accueil' dans le mode libre ou replay	Retour à la page d'accueil
7	Lors du clique sur 'Sauvegarder' dans le mode libre	Toute la séquence créée est sauvegardée dans un fichier xml et un message "Sauvegarde réussie" apparaît brièvement au milieu de l'écran
8	Dans le mode libre, lorsque la première corde de la guitare est jouée	L'effet de comète est créé sur un des mortiers aléatoirement
9	Dans le mode libre, lorsque la deuxième corde de la guitare est jouée	L'effet de pluie de particules est créé aléatoirement sur le haut de l'écran
10	Si la guitare n'est pas trouvée lors du démarrage de l'application	Un message d'erreur apparaît au milieu de l'écran
11	Dans l'accueil, s'il y a plus de 10 musiques	Seuls les 10 premiers s'affichent dans l'ordre alphabétique
12	Dans l'accueil, s'il y a plus de 10 fichiers à rejouer	Seuls les 10 premiers s'affichent dans l'ordre alphabétique
13	Lors du chargement d'un fichier extérieur au programme (image ou musique), si le fichier n'existe pas	L'application continue de fonctionner et aucun fichier n'est chargé
14	Lors du chargement d'un fichier extérieur au programme, si le fichier ne correspond pas au type demandé	L'application continue de fonctionner et aucun fichier n'est chargé

Evolution des tests

N Test	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10	J11
1	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	
2	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	
3	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	
4	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	
5	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	
6	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	
7	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	
8	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	
9	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	
10	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	
11	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	
12	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	
13	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	
14	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	

Conclusion

Analyse du planning effectif

Difficultés rencontrées

Amélioration possible

Bilan personnel

Annexes

Planning prévisionnel

Tâches à réaliser	Temps nécessaire	24.04	25.04	29.04	30.04	02.05	06.05	07.05	08.05	13.05	14.05	15.05	Total
Lecture de l'énoncé	01:30	01:30											01:30
Planification	02:00	02:00											02:00
Mise en place du dépôt git	00:10	00:10											00:10
Mise en place du trello	00:30	00:30											00:30
Rendue du planning prévisionnel													
Projet													
Mise en place du système de particules	00:40		00:40										00:40
Récupération des notes en format MIDI	00:30		00:30										00:30
Vue de la page de jeu (mode libre et replay)	02:00		02:00										02:00
Création de l'effet de comète	03:00		03:00										03:00
Création de l'effet de pluie de particules	06:30			06:30									06:30
Affichage d'effet de particule en fonction des entrées MIDI	03:00				03:00								03:00
Fin du développement de création de feu d'artifice													
Sauvegarde de la séquence jouée (mode libre uniquement)	10:00				02:00	05:00	03:00						10:00
Trouver des musiques d'ambiance	01:00					01:00							01:00
Musique choisie joué en boucle dans la page jeu	01:00						01:00						01:00
Page d'accueil													
Affichage de la page d'accueil	02:00						02:00						02:00
Mode replay													
Affichage d'une séquence enregistrée	06:20							06:20					06:20
Fin du développement													
Temps retard / bonus	07:10								04:30	02:40			07:10
Bonus													
Création de l'effet de feux d'artifice d'étoiles simples													00:00
Création de l'effet de feux d'artifice à explosion simple													00:00
Création de l'effet de feux d'artifice de fontaines													00:00
Général													
Rédaction des tests	12:00	02:30			00:40				01:10	03:00	03:40	01:00	12:00
Rédaction de la documentation	25:00	01:00	01:30	01:10	02:00	01:40	01:40	01:20	02:00	02:00	04:00	06:40	25:00
Rendue du projet													
Journal de bord	03:40	00:20	00:20	00:20	00:20	00:20	00:20	00:20	00:20	00:20	00:20	00:20	03:40
	88:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	08:00	88:00
					Jalon								

Planning effectif