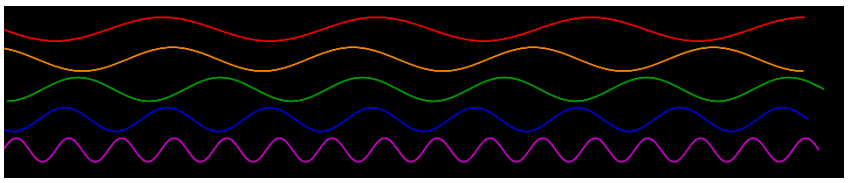
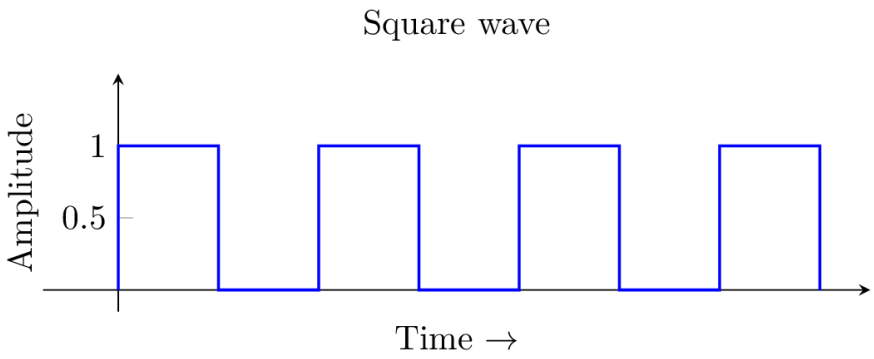
Synthétiseur Monogame :

Intro :

Pour le projet Monogame du premier cours de C#, j'ai fait un synthétiseur simple. Il permet de jouer les sept notes mineur de la cinquième octave. Il est possible de les jouer selon deux sons. Les deux sons de bases possibles sont une vague sinusoïdale :

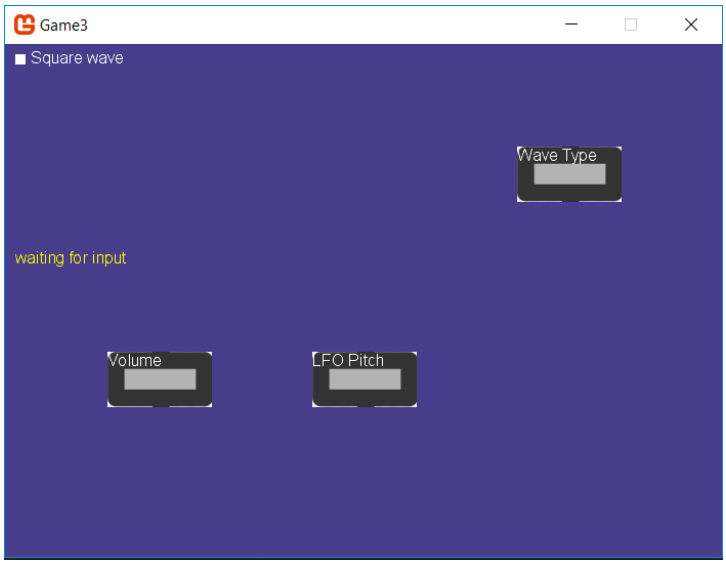


Et une vague '' carré'' :



La premiere émets des frequences fréquemment utiliser pour les basses et la deuxieme pour les sons principaux. C'est en partie la raison de la difference de volume entre le deux.

Le synthétiseur peut moduler le pitch des sons selon une fonction sinus dont l'utilisateur peut modifier l'influence. Lorsque l'utilisateur dépasse 50% de modulation, la couleur du fond change pour le notifier qu'il modifie le son de base de manière significative.



CODE :

- ***1 enum avec au moins 3 éléments :***

J'ai utilisé un enum dans mon code qui contient les clés du clavier, me permettant d'activer des sons lorsque l'utilisateur appui sur les chiffre de un à sept. C'est-à-dire en utilisant D1 à D7

```
List<Keys> myKeys = new List<Keys>()
{
    Keys.D1,
    Keys.D2,
    Keys.D3,
    Keys.D4,
    Keys.D5,
    Keys.D6,
    Keys.D7
};
```

- ***1 instruction d'itération (for, foreach, while):***

J'utilise une itération ``foreach`` pour inséré les vagues sonores dans des instances, qui pourront être modifier a leur tour plus loin dans le code.

```
foreach (SoundEffect sound in soundEffects)
{
    instances.Add(sound.CreateInstance());
}
```

- ***1 tableau :***

En ce qui concerne le tableau, quelques-uns ont été utiliser, mais le plus significatif c'est le tableau d'instance. Il permet de naviguer à travers chacune des instances crées pour chacun des sons a l'aide de son index ``i``.

```
// joue un son
for (int i = 0; i < myKeys.Count; i++)
{

    Keys currentKey = myKeys[i];
    if (Keyboard.GetState().IsKeyDown(currentKey))
    {
        ispressed = true;
        if (sine == false)
        {
            instances[i + 7].IsLooped = true;
            instances[i + 7].Play();
        }
        else
        {
            instances[i].IsLooped = true;
            instances[i].Play();
        }

    }
```

- ***1 appel de méthode avec 2 arguments nommés :***

  La fonction draw est appeler de la classe Sprite avec 2 arguments nommés, c'est-à-dire spritebatch et vector. Elle permet de faire apparaitre un petit rectangle a une position définie.

```
public void draw(SpriteBatch spriteBatch, Vector2 vector)
{
    spriteBatch.Draw(texture, rectangle, color: Color.Red);

}
```

- ***1 définition de méthode surchargé (overloaded) :***

  La méthode surchargée que j'utilise sert a indiquer le type de vague sonore que l'utilisateur à choisi. Quand la vague change, la fonction surchargé est utilisée ou revient à la fonction de base

```
    public void draw(SpriteBatch spriteBatch )
    {
        spriteBatch.Draw(texture, rectangle, color: Color.White);

    }
    public void draw(SpriteBatch spriteBatch, Vector2 vector)
    {
        spriteBatch.Draw(texture, rectangle, color: Color.Red);

    }
```

- ***1 collection utilisée (List, HashSet, Dictionary) :***

J'ai utilisé une List dans laquelle je place les clés de l'enum ''keys''. Elle sert a appelé un élément qu'elle contient lors d'opérations ou lorsque le code lui fait référence.

```
List<Keys> myKeys = new List<Keys>()
{
    Keys.D1,
    Keys.D2,
    Keys.D3,
    Keys.D4,
    Keys.D5,
    Keys.D6,
    Keys.D7
};
```

- ***JEU :***

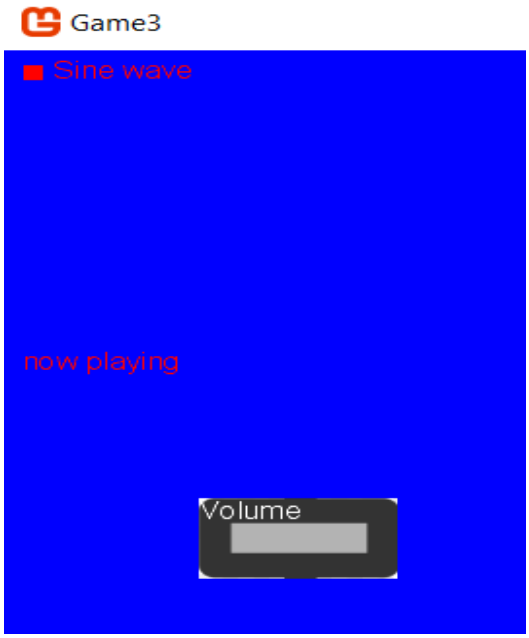
Le jeu utilise les touches : 1, 2, 3, 5, 6, 7. Elles servent à jouer les notes mineures de l'octave numéro 5. La raison pour laquelle les majeurs n'ont pas été incluent est que le but premier d'un synthétiseur est de créer un son a l'aide d'une onde sonore et ensuite de la modifier à l'aide d'une modulation.

- ***Interface***

L'interface du synthétiseur est très simple avec trois boutons : Volume, LFO pitch et Wave Type.

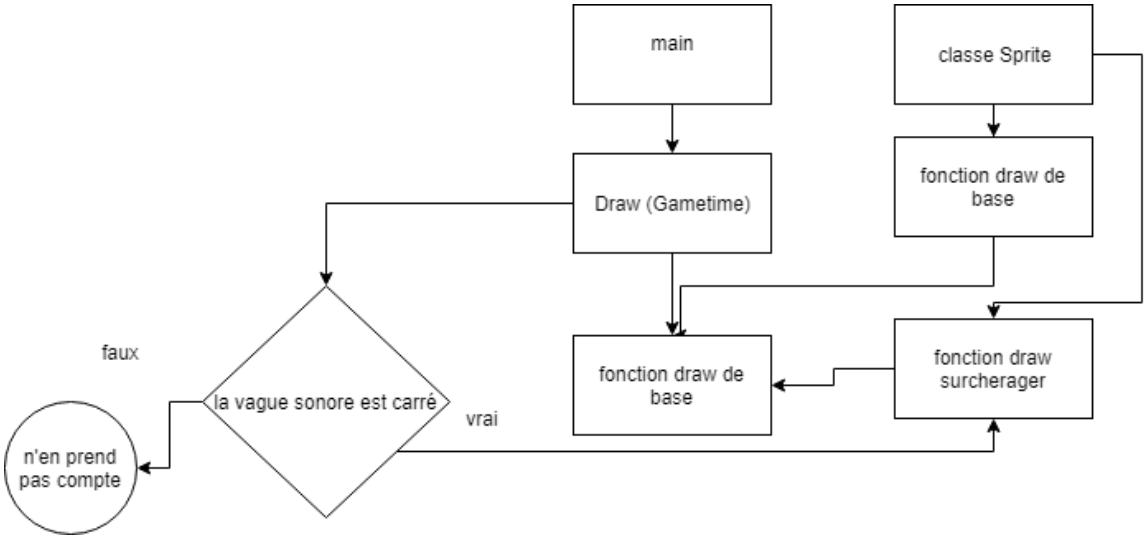
1) **Volume** : contrôle le pourcentage de volume que monogame envoie à l'instance en cours, contrôle le volume.
2) **LFO Pitch** : contrôle le pourcentage de modulation qui affect le son, moduler par une fonction sinus.
3) **Wave Type** : Contrôle le type de son entre Sine, d'avantage basses fréquences, ou Square, qui est plus utile pour les sons principaux.
4) **Waiting for input** : indique qu'aucune note est présentement jouer.
5) **Square Wave** : indique le type de vague sonore sélectionner est de type carré.

6) Sine Wave : indique que la vague sonore sélectionner est de modulation sinusoïdale.
7) Now playing : indique qu'un son est actif.



Les classes

Les classes n'ont pas une hiérarchie entre-elles, elles opèrent chacune sur des parties distinctes du synthétiseur. Cependant, à l'intérieur d'une même classe il y a une hiérarchie, tel que défini par la méthode surcharger. La fonction de surcharge prime, lorsqu'elle est appelée, sur la fonction de base.



Difficulté reliée à un algorithme :

J'ai eu de la difficulté avec les collisions du bouton avec la souris, puisque tout d'abord j'avais essayé d'utiliser la classe Button de la calculatrice, cependant j'avais besoin de la personnaliser pour suivre la souris en position vertical. J'ai alors utiliser la classe Button pour la texture et pour entreposer quelque variables, mais les collision on été déterminer par l'équation suivante : où mousestate.Position.X représente la position en X de la souris. La position des Y de la souris est représenter par mousestate.Position.Y. `Xs[0]`représente la `position en X du bouton 1.` `Ys[0]` représente la position en Y du bouton 1. La hauteur logique du bouton est définit par barknob.Height et la largeur logique par barknob.Width. La largeur est divisée en 2 puisqu'elle part du centre, il en est de même pour la hauteur logique. Les bordures maximales et minimales du déplacement du bouton sont de 300 et 100 sur les Y. Les X sont non pertinents pour les déplacements puisque le pourcentage d'influence du bouton est basé sur sa position relative a 100 sur 300 en Y seulement. Par exemple : 200 serais 50% d'effet.

```
mousestate.Position.X > Xs[0] &&
mousestate.Position.Y < Ys[0] + barknob.Height &&
mousestate.Position.X < Xs[0] + barknob.Width/2 &&
mousestate.Position.Y > Ys[0] - barknob.Height / 2 &&
mousestate.Position.Y < 300 &&
mousestate.Position.Y > 100
```

Difficulté reliée au langage :

J'ai principalement eu de la misère à savoir ce que je ne savais pas… je m'explique, je ne savais pas comment approcher les problèmes, ce qui pouvait les rendre très dur à régler. Cependant, a force d'essayer des options, j'ai compris à quel point les classes, les collections, les instructions; fonctionnent entre elles et rendent la programmation plus simple. Par simple je veux pas dire facile, mais plus efficace, lisible et compréhensible. Je dirais que la plus grande difficulté reliée au langage est à quel point il est vaste et de vraiment connaitre toutes les possibilités à notre disposition.

Sources (dont je me rappel) les images sont libre de droits:

https://stackoverflow.com/

https://docs.microsoft.com/fr-ca/dotnet/csharp/programming-guide/

Collège Lasalle (Felix) : documentation de cours

Et le premier vidéo de monogame que j'ai regardé :
https://www.youtube.com/watch?v=N6r87rGDFV8