

PSAR

RAPPORT

iBalezator



Adrien FERREIRA
Alexandra HOSPITAL

Encadrant : Fabrice KORDON
Client : Thomas BASPEYRAS

Table des matières

1	Introduction	3
1.1	Sujet	3
1.2	Problématique	3
2	Présentation du produit	4
2.1	Fonctionnalités	4
2.1.1	Mode manche/clavier	4
2.1.2	Mode portée/manche	5
2.1.3	Score et aide	6
2.2	Manuel d'utilisation	6
2.2.1	Mode manche/clavier	6
2.2.2	Mode portée/manche	7
2.3	Recettes : bilan des tests de validation	7
2.3.1	Mode manche/clavier	7
2.3.2	Mode portée/manche	8
2.3.3	Changement de mode	8
3	Réalisation et implémentation	9
3.1	Particularité du solfège	9
3.2	Mise en relation d'une note sur le manche avec une note sur la portée	10
3.3	Mise en relation de coordonnées d'image de manche avec une Note	10
3.4	Son	11
3.5	Temporisation	11
3.6	Architecture finale	11
3.7	Adaptation de l'application aux différentes tailles d'écran des terminaux	12
4	Organisation	14
4.1	Équipe de développement, environnement de développement	14
4.2	Organisation dans le temps	14
4.3	Documents produits	14
5	Conclusion	15
5.1	L'application iBalezator	15
5.2	Améliorations possibles de l'application	15
6	Annexe	17
6.1	Lecture de notes sur le manche	17
6.2	Lecture de notes sur la portée	17
6.3	Comptes rendus des réunions	18

1 Introduction

Ce document constitue le rapport du projet iBalezator.

Maitrise d'oeuvre : Thomas Baspeyras, Fabrice Kordon

Maitrise d'ouvrage : Adrien Ferreira, Alexandra Hospital

Le produit du projet étant directement lié à la musique, quelques notions importantes pour comprendre son intégralité ont été repertoriées en annexe.

1.1 Sujet

iBalezator est une application pour petit terminaux sous iOS 8 (de 3,5 à 4,7 pouces) qui consiste en un jeu de devinette pour mémoriser la disposition des notes sur le manche d'une guitare. Elle s'inspire du site Internet Le Balezator <http://www.orbite.info/balezator/>, créé par Thomas Baspeyras. L'application comporte deux modes de jeu :

- Le mode manche/clavier, dans lequel l'utilisateur lit une question sur le manche et répond sur un clavier
- Le mode portée/manche, dans lequel l'utilisateur lit une question sur la portée et répond sur le manche

L'application s'adresse à des guitaristes débutants connaissant le principe de jeu sur le manche d'une guitare et sachant lire une partition musicale. Elle propose une aide à la mémorisation des notes sur le manche de la guitare et à la lecture d'une partition.

1.2 Problématique

La problématique de ce projet est de pouvoir adapter un mode de jeu présent sur le site web Le Balezator – le mode manche/clavier – sur petits terminaux en restant ludique et ergonomique ; ainsi que d'implémenter un nouveau mode de jeu – le mode portée/manche.

2 Présentation du produit

2.1 Fonctionnalités

2.1.1 Mode manche/clavier

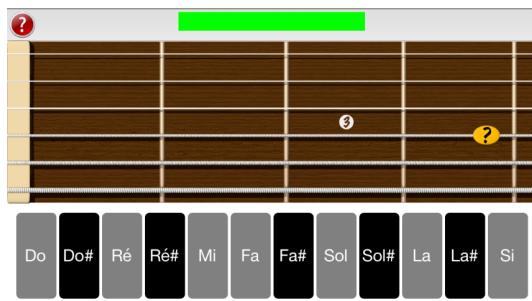


FIGURE 2.1 – Le mode manche/clavier

L’application démarre en mode manche/clavier.

Ce mode est composé :

- D’une barre d’outils comprenant une barre de score et une aide
- D’un manche de guitare qu’il est possible de faire glisser vers la gauche pour travailler sur les parties plus aiguës
- D’un clavier composé des 12 notes de la gamme : Do, Do#, Ré, Ré#, Mi, Fa, Fa#, Sol, Sol#, La, La#, Si

Une note est tirée aléatoirement et apparaît sous forme de question sur le manche de la guitare symbolisée par une pastille jaune (figure 2.2) indiquant à l’utilisateur la note à trouver. Le son de la note en question est émis. À l’aide du clavier présenté dans la partie basse de l’écran, l’utilisateur choisit sa réponse, la note qu’il pense lire.

Si l’utilisateur saisit une mauvaise réponse, la fausse note apparaît sous forme de pastille rouge (figure 2.4) avec le son correspondant à cette fausse note. S’il y a plusieurs possibilités d’affichage de la mauvaise note sur le manche, l’application favorisera une position proche de la note à trouver. L’application invite l’utilisateur à réessayer jusqu’à ce qu’il trouve la bonne réponse, avec le son de la note à trouver. Par exemple, sur la figure ci-dessus, la note à trouver est un Fa#. Tant que l’utilisateur n’aura pas répondu la touche Fa# du clavier, l’application reposera cette même question.

Si l’utilisateur saisit une bonne réponse, la pastille sur le manche s’allumera en vert (figure 2.3) pendant une seconde, le son de la bonne note sera émis, et l’application passera à la question suivante.

L’utilisateur a la possibilité de faire défiler le manche pour choisir la partie sur laquelle il souhaite travailler. Les questions posées concerneront toujours la partie du manche visible à l’écran.



FIGURE 2.2 – Question



FIGURE 2.3 – Bonne réponse



FIGURE 2.4 – Mauvaise réponse

L’utilisateur change de mode de jeu pour passer en mode portée/manche.

2.1.2 Mode portée/manche

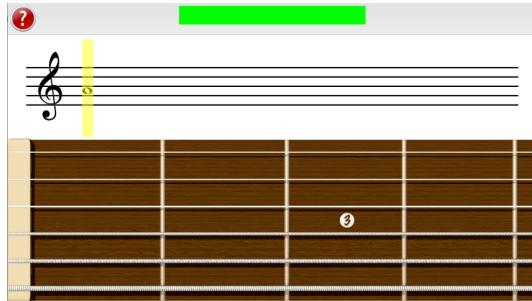


FIGURE 2.5 – Le mode portée/manche

Le mode portée/manche est composé :

- D'une barre d'outils comprenant une barre de score et une aide
- D'une portée en clé de sol
- D'un manche de guitare qu'il est possible de faire glisser vers la gauche pour travailler sur les parties plus aiguës

L'ensemble de notes à afficher est tiré aléatoirement. La première note à trouver est affichée sous forme de question symbolisée par une surbrillance jaune (figure 2.1.2). Le son de la note à trouver est émis. L'utilisateur lit les notes qui apparaissent sur la portée. À l'aide du manche de guitare dans la partie basse de l'écran, l'utilisateur choisit sa réponse en tapant sur la position souhaitée.

Si l'utilisateur saisit une mauvaise réponse, la surbrillance de la note deviendra rouge et la note qu'il a effectivement touchée sur le manche apparaîtra sur la partition (figure 2.6), avec le son correspondant à cette fausse note. L'utilisateur est invité à réessayer jusqu'à ce qu'il trouve la bonne réponse. La barre de score est mise à jour.

Si l'utilisateur saisit une bonne réponse, la surbrillance devient verte (figure 2.7) pendant une seconde, le son correspondant à cette bonne réponse est émis et l'application passe à la question suivante : la note apparaît sur la portée en surbrillance jaune.

L'utilisateur a la possibilité de faire défiler le manche pour choisir la partie qu'il souhaite travailler. Les questions posées concerneront toujours la partie du manche visible à l'écran.

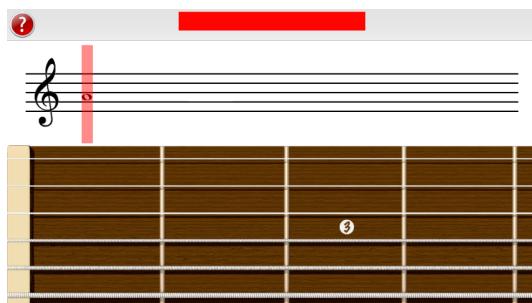


FIGURE 2.6 – Mauvaise réponse

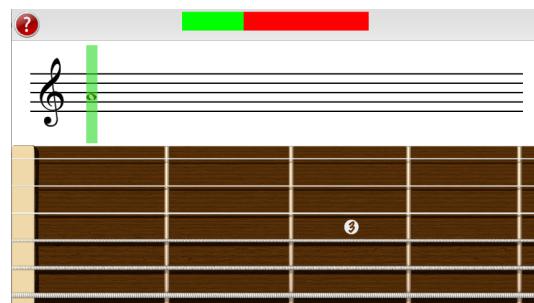


FIGURE 2.7 – Bonne réponse

La particularité de la portée est que quand les notes sont hautes (pour notre application : à partir du Mi 3 lignes au dessus de la portée, inclus), on utilise en solfège une notation « 8va » qui permet d'écrire les notes une octave plus bas que leur place réelle (cf : paragraphe 3.2 pour plus de détails techniques). Ainsi, on évite d'ajouter des lignes au dessus de la portée, ce qui rendrait peu claire la lecture. L'application prend en compte cette notation pour les notes qui sont plus aiguës que le Ré#, 2 lignes au dessus de la portée. À partir du Mi suivant, les notes sont écrites une octave en dessous.

L'utilisateur change de mode de jeu pour passer en mode manche/clavier.

2.1.3 Score et aide

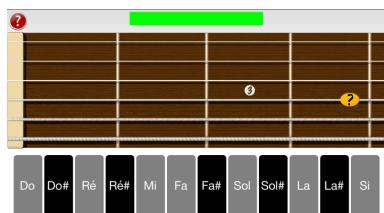
Une barre de score commune aux deux modes est présente dans la barre d'outils, dans la partie supérieure de l'écran. Elle fait état de la progression de l'utilisateur en affichant une jauge en proportion de bonnes réponses (en vert) et de mauvaises réponses (en rouge).

Une vue d'aide a été prévue sur l'application. Aujourd'hui, elle n'est pas complétée.

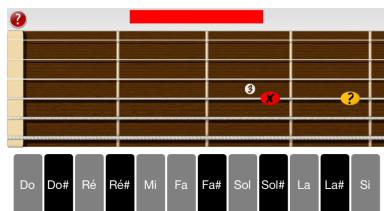
2.2 Manuel d'utilisation

L'application démarre en mode manche/clavier. Pour changer de mode, il faut effectuer un glissé vertical sur l'écran.

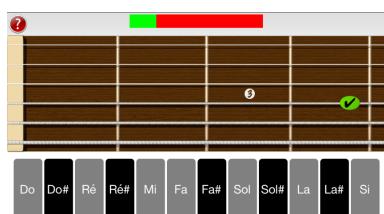
2.2.1 Mode manche/clavier



Au démarrage de l'application, le système commence en mode manche/clavier. Une question est posée, représentée par un marqueur jaune. Le son de la note à trouver est joué. Pour répondre, appuyer sur une des touches du clavier.



Si la réponse est mauvaise, le marqueur le notifie (marqueur rouge). Le son de la fausse note est joué. Répondre de nouveau jusqu'à ce que la réponse soit acceptée.

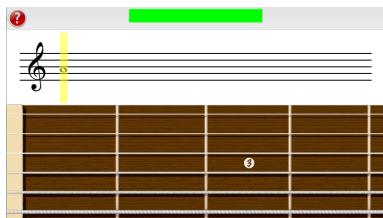


Si la réponse est bonne, le marqueur le notifie (marqueur vert), le son de la note est joué, puis une nouvelle question est posée. Répondre comme précédemment.

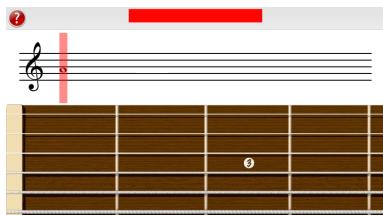
N.B : Si le manche est changé de place, une nouvelle question sera générée. Les questions concernent toujours la partie visible du manche. Répondre comme expliqué précédemment.

2.2.2 Mode portée/manche

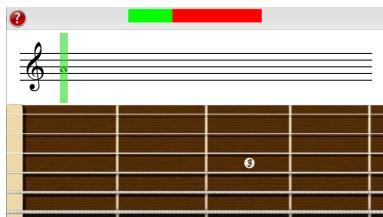
Pour changer de mode, il faut effectuer un glissé vertical sur le manche.



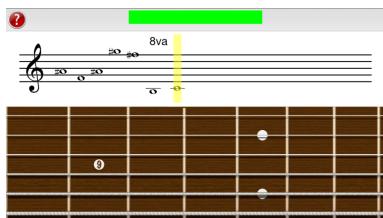
Après avoir changé de mode, une question est posée sur la portée : la note apparaît surlignée en jaune. Le son de la note à trouver est joué. Pour répondre, appuyer sur une position sur le manche.



Si la réponse est mauvaise, la surbrillance le notifie (rouge). La fausse note apparaît sur la portée et le son correspondant à cette fausse note est joué. Répondre de nouveau jusqu'à ce que la réponse soit acceptée.



Si la réponse est bonne, la surbrillance le notifie (vert) pendant une seconde, le son de la note est joué, puis une nouvelle question est posée.



Dans le cas où le manche a été déplacé et que la frette 12 apparaît à l'écran, il est possible d'avoir la notation « octava » affichée. À partir de cette notation, lire les notes une octave au dessus.

2.3 Recettes : bilan des tests de validation

Une série de tests a été réalisée sur l'application par l'équipe de développement, en jouant au jeu de l'iBalezator.

Les interfaces graphique de chacun des modes ont été réalisées d'après les maquettes fournies par notre client, Thomas Baspeyras. Chaque élément du jeu (manche, clavier, portée, barre de score) correspond à un pourcentage de la zone de l'écran. Les touches du clavier sont disposées en lignes et colorées à la façon d'un piano (c'est-à-dire que les notes dièses sont plus foncées que les notes « normales ») et dimensionnées de façon à ce que l'utilisateur ait un touché confortable sur ces touches.

De même, le manche de guitare doit être assez grand pour que le touché sur une position soit précis. Il doit y avoir au moins 4 cases à l'écran du côté sillet du manche, même sur un petit écran (i.e : 3,5 pouces).

2.3.1 Mode manche/clavier

L'application est capable de créer un jeu et de garder un enchaînement logique des événements. Le système tire au sort une note dans la partie visible du manche, affiche le marqueur de question, joue la note puis attend la réponse de l'utilisateur. Cela signifie que l'application est capable de mettre en relation une note avec sa position sur le manche, et le son lui correspondant.

Le système est capable de détecter une réponse entrée par l'utilisateur sur le clavier, de l'analyser et de la qualifier de bonne ou mauvaise réponse. Il le notifie au moyen de l'affichage du marqueur vert ou rouge, il joue le son correspondant à la note entrée par l'utilisateur.

Si le manche est changé de place, une nouvelle question est générée, toujours dans la partie visible du manche.

La mise à jour du score est également gérée.

Bilan : le mode manche/clavier est conforme aux besoins détaillés dans le cahier des charges.

2.3.2 Mode portée/manche

L'application est capable de créer un jeu et de garder un enchaînement logique des événements.

Le système tire au sort une note dans la partie visible du manche, la dessine sur la portée, puis attend la réponse de l'utilisateur. L'application est capable de mettre en relation une note et sa représentation graphique sur la portée, notamment lorsque cette note est « diésée » ou non. Les notes Mi et Si ne seront jamais affichées avec un dièse.

Le système est capable de détecter une réponse entrée par l'utilisateur sur le manche, de l'analyser et de la qualifier de bonne ou mauvaise réponse. Il le notifie au moyen de l'affichage de la fausse note sur la portée, du surlignage rouge de la note et du son de la fausse note joué au moment du tap de l'utilisateur. Si le manche est changé de place, une nouvelle question est générée, toujours dans la partie visible du manche.

Dans le cas où les notes sont trop aiguës pour être affichées sur la portée ou sur les 2 lignes au dessus de la portée, l'application est capable d'afficher un « 8va » à partir duquel toutes les notes seront écrites une octave en dessous de leur position réelle.

Lorsque la portée est remplie et que la dernière note a été validée comme bonne réponse, la portée est complètement vidée pour laisser place à un nouvel ensemble de notes tirées aléatoirement, comme au lancement de ce mode.

Bilan : le mode portée/manche est conforme aux besoins détaillés dans le cahier des charges.

2.3.3 Changement de mode

Le changement de mode s'effectue par un glissé vertical sur l'écran.

3 Réalisation et implémentation

3.1 Particularité du solfège

En musique occidentale, une Octave est divisée en 12 demi-tons. Sur un piano, un demi-ton est matérialisé par deux touches adjacentes. Un ton est composé de deux demi-tons. Une gamme est composée de tons et demi-tons. La gamme majeure est composée de 2 tons puis un demi-ton puis trois tons et un demi-ton. Si l'on part d'un Do et que l'on suit un tel schéma, sur un piano, cela correspond aux touches blanches. Une Octave est divisée selon la structure d'une gamme majeure. Chaque touche blanche correspond à un degré de la gamme. Deux degrés sont donc séparés soit par un ton soit par un demi-ton. Un intervalle est le nombre de demi-tons qui sépare deux degrés. La tessiture d'un instrument c'est la totalité des notes qu'il est capable de jouer. Dans le cadre de ce projet, cette particularité de la théorie musicale apporte son lot de problématiques notamment : le placement des notes sur une portée, le calcul d'intervalle, la structure de l'accordage de la guitare.

Nous avons mis au point un algorithme qui permet de consommer successivement et hétérogénierement des demi-tons et résoudre les problèmes cités ci-dessus. La structure interne d'une Octave est irrégulière mais elle est cyclique. Cette cyclicité nous a permis d'utiliser les modulus sur les indices pour nous déplacer sur toute la tessiture de la guitare avec une seule structure de données.

La gestion des dièses est omise, elle consiste presque exclusivement à vérifier à la fin de la boucle que `resteDemiTons` est égal à zéro. Partant de la note la plus grâve que la guitare peut jouer (paramètre `a`), on peut ainsi savoir où placer `b` sur la portée. Ceci en multipliant un décalage fixe par rapport aux coordonnées constantes connue de `a` sur la portée.

```
/*
 * Fonction permettant de trouver le nombre d'interlignes entre deux notes sur la portée.
 * Retourne le nombre d'interlignes séparant a et b
 * a doit être plus grâve que b.
 * a : Note de départ
 * b : Note d'arrivée
 */
Fonction nombreDeDemiTonsEntreAB(a : Note, b : Note) : Entier
DébutFonction
    // deux tons, un demi-ton, trois tons, un demi-ton
    Constante tonalitéMajeure = [2, 2, 1, 2, 2, 2, 1];

    Variable resteDemiTons : Entier; //restant à consommer
    Variable nbInt : Entier; // Nombre d'interlignes
    Variable cmpt : Entier; // itérateur sur tonalitéMajeure

    resteDemiTons = a.nbDemiTonsAvec(b);
    cmpt = 0;
    nbInt = 0;

    TantQue resteDemiTons > 0
        Faire
            //consommation des demi-tons en fonction de la structure de la gamme
            resteDemiTons = resteDemiTons - tonalitéMajeure[cmpt modulo tonalitéMajeure.taille];
            cmpt++;
```

```

nbInt++;
FinTantQue

    retourner nbInt;
FinFonction

```

3.2 Mise en relation d'une note sur le manche avec une note sur la portée

Une des directives du cahier des charges est d'afficher le symbole 8va quand une note est très aigue. Ce symbole indique à l'utilisateur qu'il doit lire la note affichée une octave plus aigue que ce qui est affiché sur la portée. Une autre directive du cahier des charges est qu'une fois qu'une note est octaviée, celles qui suivent doivent l'être également. Ceci dit, certaines notes ne peuvent pas être lues en notation octaviée. Sur une même position visible du manche, on aura donc des notes qui peuvent être octaviées et d'autres non. Hors nous utilisons un tirage aléatoire pour choisir les notes à afficher. Voilà pourquoi nous avons dû biaiser l'aléatoire. Nous tirons des notes aléatoire et si une apparaît comme nécessitant d'être octaviée, nous déterminons un sous ensemble de notes qui peuvent être octaviées parmi toutes celles sur la partie visisible du manche. Puis nous relançons l'aléatoire sur ce nouveau sous-ensemble.

Sur un manche de guitare il est possible de jouer une même note/fréquence à différents endroits du manche (appelées notes enharmoniques). Ces différentes notes peuvent être sur une même partie visible du manche. Cette particularité conduit à plusieurs problèmes. Par exemple, en mode manche/clavier, quand une note est affichée et que l'utilisateur saisit une mauvaise réponse au clavier, il est impossible de savoir quelle note il avait en tête. Voilà pourquoi nous avons utilisé un algorithme qui à une note donnée sur le manche trouve la note éronnée la plus proche. Ce même phénomène survient en mode portée/manche. Quand l'utilisateur lit une note sur la portée il a potentiellement plusieurs positions sur lesquelles il peut la jouer. Dans ce cas là, il faut donc trouver et accepter toutes les réponses possibles.

Comme nous l'avons vu dans la section ??? le positionnement d'une note sur la portée dépend d'une structure hétérogène. Or, nous avons décidé de dessiner la portée grâce aux fonctionnalités d'iOS. Des lignes supplémentaires sont rajoutées en haut (respectivement bas) de la portée quand une note est trop aigue (respectivement grêve) pour rentrer sur les cinq lignes de base. Hors, ces lignes supplémentaires ne sont à dessiner que sous les notes concernées. Voilà pourquoi, en conjonction de l'algorithme présenté en section ???, nous avons dû déterminer la quantité de lignes supplémentaires à ajouter pour chaque note du tirage. Ceci au moment de l'affichage de chaque note, individuellement.

3.3 Mise en relation de coordonnées d'image de manche avec une Note

Le tirage des notes/questions auxquelles l'utilisateur est invité répondre doit toujours être en fonction de la partie du manche affichée à l'écran. Seule les frettes affichées dans leur totalité sont éligibles à l'aléatoire. Le manche de la guitare n'étant finalement qu'une image que l'on peut faire défiler horizontalement, il a donc fallu convertir de simples coordonnées en frettes. IOS nous donne accès à l'offset de l'image et les dimensions de l'écran. Suite à cela nous avons stocké les coordonnées des frettes et déterminé lesquelles étaient affichées dans leur totalité.

Nous avons aussi stocké les coordonnées des cordes. De ce fait nous avons pu opérer à des croisements car quand l'utilisateur tape sur l'image du manche, IOS nous donne la coordonnée du tap. À un tap sur l'écran il est donc possible de retrouver la note que l'utilisateur a voulu sélectionner.

3.4 Son

Les notes affichées à l'écran doivent aussi être jouées sur les hauts-parleurs/écouteurs du terminal. Le client avait clairement demandé dès le départ qu'il ne voulait pas que l'on stocke les 36 fichiers audio correspondant aux notes qu'il est possible de jouer. En fonction de la qualité des enregistrements ces fichiers peuvent être volumineux et augmenter le poids de l'application et nuire à son succès.

AudioKit est une bibliothèque de génération de fréquences sonores pour IOS. Elle est écrite en *Objective-C* mais il est possible de lancer des fonctions écrites dans ce langage depuis *Swift* grâce à une interface. Cette bibliothèque est intéressante car elle permet de jouer des notes mais avec des simulations d'instruments. En jouant sur les caractéristiques de vie du son elle va générer des fréquences qui ressemblent à une guitare, une mandoline, un vibraphone, entre autres. Nous avons créé un contrôleur capable de prendre en paramètre des objets Note de notre modèle et ainsi masquer les complexités de la bibliothèque dans l'implémentation de la logique du jeu. Cette bibliothèque n'utilise aucun fichier audio et est très légère.

Cependant elle est relativement jeune et il est difficile de trouver des articles, des exemples, de l'aide. Il apparaît qu'elle n'est pas Thread-safe ce qui a posé des problèmes avec l'utilisation de Timers et aucun documentation sur ce point n'est disponible. Aucune documentation non-plus sur la manière de compiler et exporter une application stand-alone qui embarque cette bibliothèque.

3.5 Temporisation

Une des demandes du cahier des charges était de temporiser l'enchaînement des questions. Ainsi, si l'utilisateur saisit une réponse valide, on affiche un indicateur, on attend 1 seconde puis on passe à la question suivante. Pendant ce laps d'attente il est important de désactiver certaines fonctionnalités de l'interface. Par exemple, en mode clavier/manche, si l'utilisateur sélectionne d'autres notes sur le clavier pendant la temporisation, ces réponses doivent être ignorées. Le fonctionnement par défaut d'IOS est de tous les traiter après la fin du timer. Nous avons donc dû gérer les désactivations de fonctionnalités en fonction du contexte.

3.6 Architecture finale

L'architecture finale de l'application est restée, dans sa majeure partie, fidèle au diagramme de classe élaboré en phase d'analyse.

Quelques modifications ont cependant été apportées. Le modèle, tout d'abord, compte une classe de plus : MainModel. Cette classe est donc pour stocker les données communes aux deux modes de jeu. Nous pensions au début que chaque mode de jeu tiendrait à jour son propre score. Après concertation avec le client, ce dernier a jugé plus pertinent que le score soit commun aux deux modes de jeu. Le mode de jeu courant a été également stocké dans cette classe plutôt que dans le contrôleur principal. Elle possède également en attribut deux instances correspondant aux deux modèles des deux modes de jeu.

L'affichage d'une fenêtre d'aide n'avait pas été spécifié dans le cahier des charges. Cependant, comme son ajout était simple et rapide, nous l'avons ajouté. Les classes HelpController et HelpView permettent la gestion d'un tel affichage. La classe HelpController n'est présente que par soucis de cohérence mais elle ne fait que relayer les ordres du GameController.

Les vues de l'application, conformément au patron MVC, ne s'occupent que des affichages sans traitements. La vue StaffView embarque ceci-dit une méthode privée qui calcule et affiche les lignes supplémentaires nécessaires au placement de la note. Nous avons placé ce traitement dans la vue car c'est la vue qui connaît les dimensions de la portée et si la note à afficher est à l'intérieur des cinq lignes de base. Le

StaffController demande d'afficher la note à la vue à telle position et elle place les lignes supplémentaires si besoin. Contrairement aux dièzes, 8va et marqueurs de réponse, les lignes supplémentaires n'ont pas de logique métier, elle sont simplement une aide à la lecture pour l'utilisateur.

La fonction `noteToFrequency()` aurait pu être placée dans la classe Note. Cependant, nous l'avons laissé dans la classe SoundController dans le but d'isoler tous les aspects sonores dans cette classe.

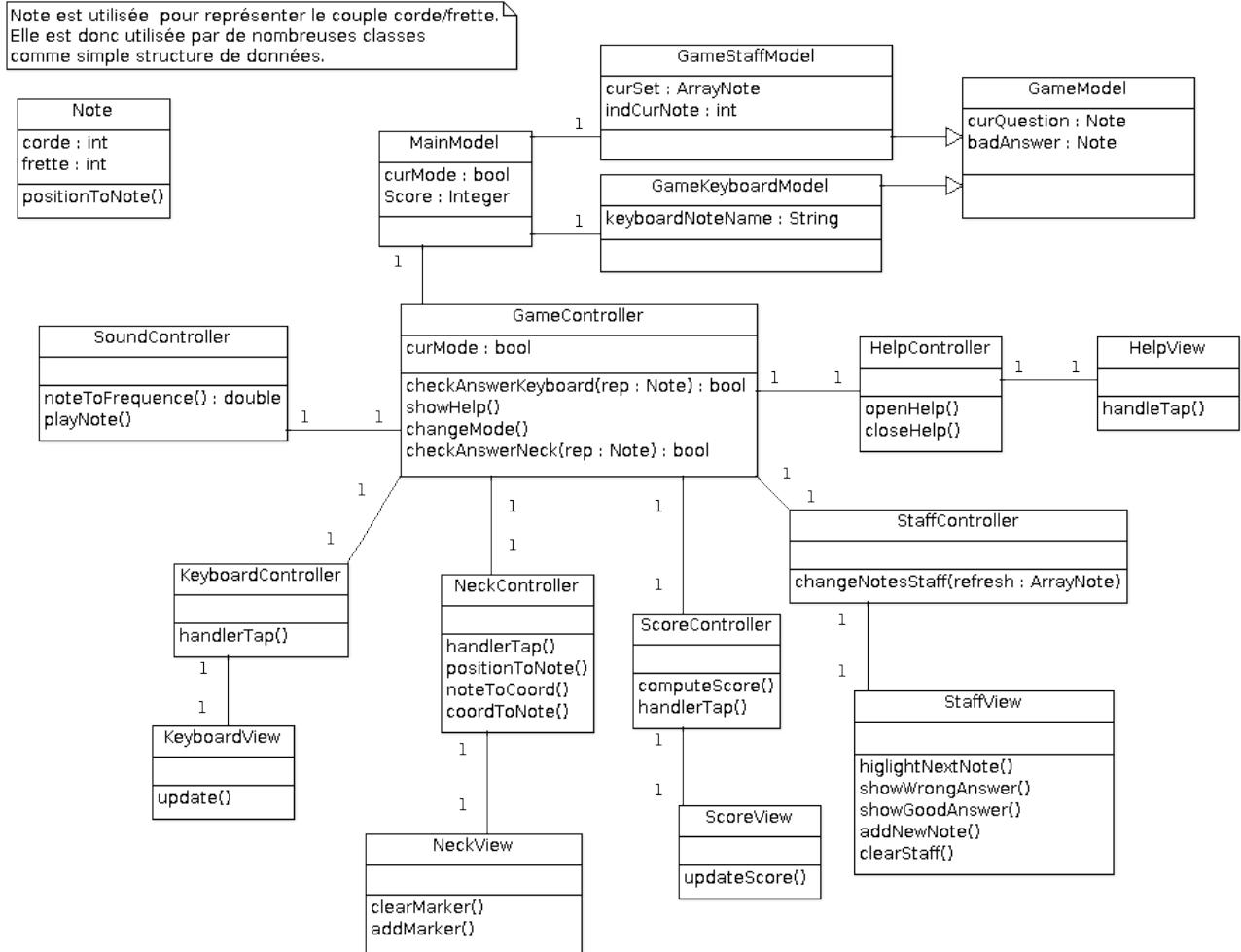


FIGURE 3.1 – Diagramme de classe de l'architecture finale

3.7 Adaptation de l'application aux différentes tailles d'écran des terminaux

Une des directives du cahier des charges était que l'on puisse toujours voir quatre frettes du manche, même en haut du manche. Le client nous a fourni une image de manche qu'il voulait sur l'application. Aux vues des proportions de l'image, il n'était pas possible de faire apparaître quatre frettes. En accord avec le client, nous avons donc décidé de tasser l'image et donc perdre les proportions de l'image originale. Cette atypicité n'a pas posé de problème majeur, il a suffit de changer le tableau de coordonnées des frettes et codres.

L'affichage de la portée doit s'adapter à la taille de l'écran du terminal. Une première approche consiste à stocker les images des 36 notes possibles dessinées sur une portée et les afficher côte-à-côte en fonction du tirage. Cette méthode présente l'inconvénient de nécessiter le stockage d'un grand nombre d'images,

nous avons donc écarté cette solution. Une deuxième méthode consiste à n'avoir qu'une image de portée et ensuite dessiner les notes sur ce fond. Plutôt que d'utiliser cette technique qui pourrait éventuellement faire apparaître la portée comme tassée en fonction de sa définition originale et les dimensions du terminal, nous avons décidé de dessiner la portée. Les avantages de cette méthode est que l'on ne stocke aucune image, l'application sera donc d'autant plus légère. De plus, le dessin étant vectoriel, c'est l'assurance d'avoir des traits nets quelque-soit l'écran. Enfin, il devient possible d'adapter l'écartement, la position et la hauteur des traits de la portée en fonction des capacités d'affichage du terminal. Nous devons donc calculer le nombre et la position des lignes supplémentaires au besoin comme indiqué dans la section ???.

La tessiture est l'ensemble des notes qui peuvent être joué par un instrument donné. La guitare est capable de jouer trente-six notes différentes qui doivent toutes pouvoir être affichées sur la portée. La portée doit donc avoir une taille conséquente pour pouvoir toutes les accueillir. Les écrans auxquels nous avons à faire étant petits, nous avons dû trouver la taille optimale pour le lecture sur la portée et utiliser le symbole **8va** en conséquence. Ainsi, nous avons trouvé que le seuil optimal avant l'affichage du symbole **8va** est Ré3. Ceci à pour effet d'abaisser la portée vers le bas et de faciliter la lecture des notes en tête de manche qui correspondra certainement à la position la plus fréquente chez les guitaristes débutants.

4 Organisation

4.1 Équipe de développement, environnement de développement

L'équipe de développement est composée d'Adrien Ferreira et d'Alexandra Hospital.

Le matériel utilisé est un Mac mini sous MacOS 10.9.5. Les tests sont effectués sur un iPod touch MD720NF/A sous iOS 8.1.3. Le déploiement sera fait du Mac vers l'iPod via le logiciel Xcode.

Le développement est effectué sur la plate-forme de développement Xcode 6.1 en langage Swift. Nous utilisons les différentes bibliothèques incluses par défaut pour le Swift.

La bibliothèque de son utilisée est audiokit (<http://audiokit.io/features/>) implémentée en Objective-C.

4.2 Organisation dans le temps

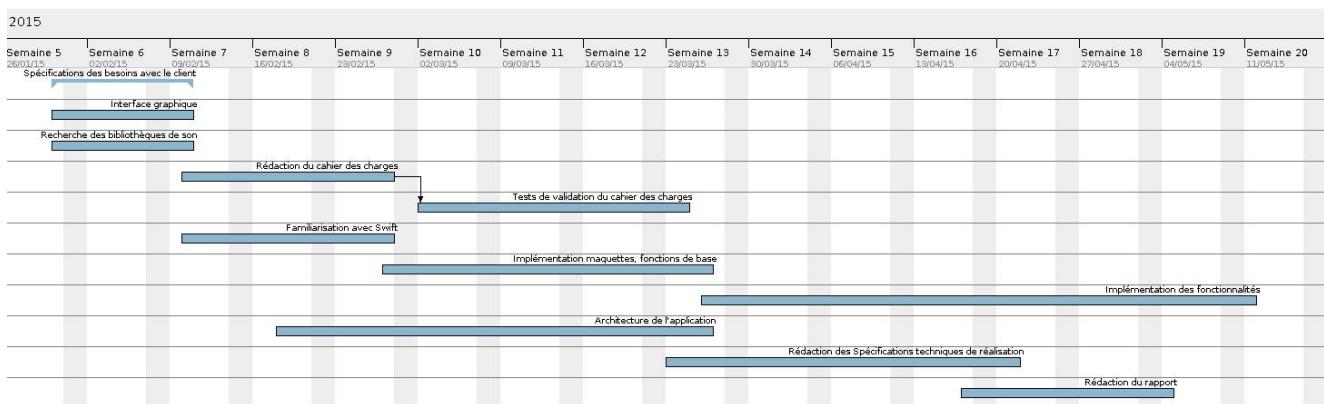


FIGURE 4.1 – Notre organisation dans le temps

Durant toute la durée du projet, des réunions avec l'encadrant et le client ont été planifiées pour tester l'application et faire un bilan de sa implantation. Des comptes rendus ont été rédigés et ajoutés à l'annexe de ce rapport.

4.3 Documents produits

Les différents documents produits lors de ce projet sont :

- Le cahier des charges, spécifiant tous les besoins de l'application pour le client
- Les Spécifications Techniques de Réalisation (STR), décrivant l'analyse de l'architecture de l'application et de son implémentation avant la phase de développement
- Les comptes rendus joints dans l'annexe
- Le présent rapport

5 Conclusion

5.1 L'application iBalezator

Durant ce projet, nous avons développé l'application iBalezator, adaptation pour terminaux iOS du Balezator en ligne. Les deux modes de jeux décrits dans le cahier des charges – manche/clavier et portée/manche – ont été implémentés.

5.2 Améliorations possibles de l'application

Les événements réceptionnés par les vues sont directement relayés au contrôleur homonyme sans traitement. Les contrôleurs associés à chaque vue sont présents pour masquer la partie graphique de l'application aux classes gérant logique métier. Pour cela, le contrôleur du manche convertit les coordonnées de tap sur le manche en Note intelligible par le reste de l'architecture. Si l'on souhaite dans le futur changer l'image du manche ou proposer le mode gaucher, le reste de l'application n'en est pas affecté.

Il peut être utile de proposer à l'utilisateur de changer l'accordage de sa guitare. Pour effectuer nos calculs de Note sur le manche, nous passons par un tableau qui stocke les intervalles entre les cordes de la guitare. Dans notre modèle, une note n'est qu'un couple <Frette,Corde> matérialisé par la classe Note. Si l'on voulait changer l'accordage, il suffirait de donner un autre tableau à la classe Note pour paramétriser les fonctions de conversion.

Les vues sont calculées en proportion d'écran, la portée est dessinée à l'exécution et le manche n'est qu'une image couplée à un tableau de coordonnées pour faire correspondre un tap à une note. Si l'on voulait adapter l'application à d'autre type de terminaux, d'autres tailles d'écran cela serait, a priori, possible sans trop de difficultés. Les vues s'adapteront aux proportions de l'écran. Pour l'affichage de la portée, seul un coefficient multiplicateur est à appliquer aux coordonnées des tracés. Enfin, pour l'affichage du manche, en fonction des dimensions de l'image utilisée et du terminal il faudra changer le tableau de coordonnées d'image des frettes et corde.

Le manche de la guitare est pour le moment une image. Il serait possible, comme pour la portée, de la dessiner au moment de l'exécution. Un manche de guitare possède des proportions calculables. Une formule mathématique permet d'obtenir la distance nécessaire entre chaque frette. Cette formule est valable pour n'importe quel instrument à corde. Ceci pourrait donc nous permettre d'afficher n'importe quelle manche d'instrument à corde et donc de ne pas uniquement proposer la guitare à 24 frettes. Il faudrait cependant s'assurer que le dessin de l'image du manche n'est pas trop gourmand en ressources et n'affecte pas la fluidité de l'application. Ceci n'a pas été le cas lorsque nous avons décidé de dessiner la portée plutôt que d'afficher une image.

La bibliothèque de son que nous utilisons est puissante, elle possède de nombreux réglages modifiables à l'exécution. Ses nombreuses simulations d'instruments nous permettraient d'adapter le son des notes aux futurs éventuels instruments proposés par l'application. On peut également prévoir une interface permettant à l'utilisateur de modifier ses propres réglages. La génération de son étant encapacée dans la classe SoundController, il suffirait de lui passer les paramètres utilisateurs pour qu'il puisse jouer la note en fonction. Les paramètres de forme d'onde, de réverbération, de bruit, ... seront laissés à l'utilisateur

tandis que la fréquence (qui caractérise une note) sera elle gérée par l'application.

Notre application répond au patron de conception MVC. Ainsi, la totalité des données pérennes de l'application sont stockées dans ces classes. On y retrouve entre autres le score, le mode courant, la question/réponse courante, etc. Après sérialisation de ces classes, nous pourrons les stocker dans la mémoire du téléphone et les recharger au lancement. Ces variables seront donc rechargées et l'utilisateur pourra continuer sa partie.

Il serait également possible de proposer une fenêtre de partage de score sur les réseaux sociaux. Un contrôleur servant d'interface pour cette API sera certainement utile. Après avoir récupéré le ratio de bonnes réponses de l'utilisateur, ce contrôleur enverra ces données au serveur du réseau social.

La portée actuelle affiche une armure de clé en *Do* majeur. Il est intéressant pour les guitaristes d'être à l'aise dans toutes les tonalités. Comme présenté dans la section ???, la structure de la gamme majeure est stockée dans un tableau que l'on parcourt séquentiellement et cycliquement (modulos). Pour ne pas rentrer trop dans les détails solphégiques, il faut juste retenir que dans notre algorithme, s'il reste un demi-ton à consommer à la fin de la boucle, la note est diésée. Changer la tonalité affichée sur la portée consistera simplement, non-pas à partir de la case zéro du tableau mais à changer en fonction de la tonalité. Par exemple, commencer pour *Ré* majeur à l'indice 1, *Fa* majeur à l'indice 3, *Si* majeur à l'indice 5, etc. Les tonalités mineurs seront également gérées en ne se déplaçant non-pas sur le tableau de la structure majeure mais sur celui de la structure mineure.

6 Annexe

6.1 Lecture de notes sur le manche



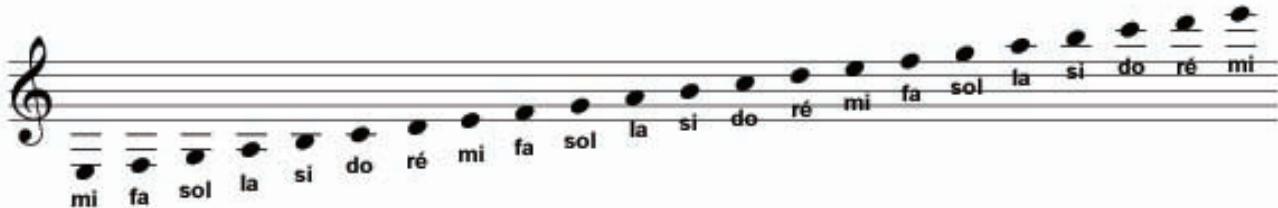
Un manche de guitare est composée de 6 cordes (représentées horizontalement sur l'image ci-dessus) et de 24 frettes (verticalement sur l'image). Les cordes sont, de haut en bas : mi (aigu), si, sol, ré, la, mi (grave). Ce sont les notes jouées pour les cordes dites « à vide », c'est-à-dire sans pincer aucune corde sur les cases du manche.

Les notes d'une gamme sont do, do#, ré, ré#, mi, fa, fa#, sol, sol#, la, la#, si. Elles sont séparées par un intervalle appelé « demi-ton ». Par exemple entre do et do#, nous avons un demi-ton. Les frettes sont placées tous les demi-tons sur le manche de la guitare. De cette façon entre deux frettes, nous avons toujours un demi-ton. Par exemple pour la corde de mi, la première frette nous donne un fa, la deuxième un fa#, la troisième un sol, et ainsi de suite.

Pour lire une note sur le manche de la guitare, on doit donc se placer sur la corde en question, puis l'augmenter de demi-ton en demi-ton jusqu'à arriver à la note voulue, à la frette désirée. Par exemple, pour la corde de mi aigu, sur la troisième frette, on obtient un sol (mi augmenté de trois demi-tons dans la gamme).

Compte tenu de la disposition des cordes et des frettes sur une guitare, il est possible d'obtenir la même note (= la même fréquence) à plusieurs endroits différents du manche.

6.2 Lecture de notes sur la portée



Une portée est constituée de 5 lignes. Les notes peuvent être disposées sur une ligne ou entre deux lignes. Les notes sur la partie basse de la portée sont graves, et celles de la partie haute sont aigues. Lorsque les notes sont plus graves ou plus aigues que celles placées dans la portée, on trace des lignes supplémentaires. Sur l'image ci-dessus, des lignes ont été ajoutées du mi au do graves et du la au mi aigus.

En mode portée/manche, les notes possibles affichées sur la portée seront celles ci-dessus. La portée contient 3 otaves.

Les notes représentées correspondent aux notes sur le manche de la guitare jusqu'à la 12ème frette, corde 1. Au delà de la 12ème frette corde 1, et donc au delà du mi inclus, les notes sont représentées à l'octave du dessous avec une indication « 8va ».

6.3 Comptes rendus des réunions