

# iBalezator

*Spécifications Techniques de Réalisation*  
Adrien FERREIRA, Alexandra HOSPITAL

2 avril 2015

# Table des matières

<b>1</b>	<b>Le projet</b>	<b>3</b>
1.1	Contexte . . . . .	3
1.2	Environnement de développement . . . . .	3
1.3	Pré-requis . . . . .	3
1.3.1	Jeu sur la guitare . . . . .	3
1.3.2	Lecture de notes sur la portée . . . . .	4
<b>2</b>	<b>Implémentation</b>	<b>5</b>
2.1	Architecture de l'application . . . . .	5
2.2	Fonctionnalités . . . . .	6
2.2.1	GameController . . . . .	6
2.2.2	GameModel . . . . .	6
2.2.3	KeyboardController . . . . .	6
2.2.4	KeyboardView . . . . .	6
2.2.5	NeckController . . . . .	6
2.2.6	NeckView . . . . .	7
2.2.7	Note . . . . .	7
2.2.8	ScoreController . . . . .	7
2.2.9	ScoreView . . . . .	7
2.2.10	SoundController . . . . .	7
2.2.11	StaffController . . . . .	7
2.2.12	StaffView . . . . .	7

# 1 Le projet

**Maitrise d'oeuvre :** Thomas Baspeyras, Fabrice Kordon

**Maitrise d'ouvrage :** Adrien FERREIRA, Alexandra HOSPITAL

## 1.1 Contexte

iBalezator est une application pour terminaux sous iOS 8 qui consiste en un jeu de devinette pour mémoriser la disposition des notes sur le manche d'une guitare. L'application comporte deux modes de jeux :

- 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.

Ce document traite des spécifications techniques de réalisation du projet iBalezator. Il complète le cahier des charges dans lequel les besoins sont spécifiés en détails.

## 1.2 Environnement de développement

Le matériel utilisé est un Mac mini sous 10.9.5. Le développement sera effectué sur la plate-forme de développement Xcode 6.1 en langage Swift, avec possibilité d'inclure des portions de code en Objective-C ou C.

Les tests seront 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.

Bibliothèques (à compléter)

## 1.3 Pré-requis

### 1.3.1 Jeu sur la guitare



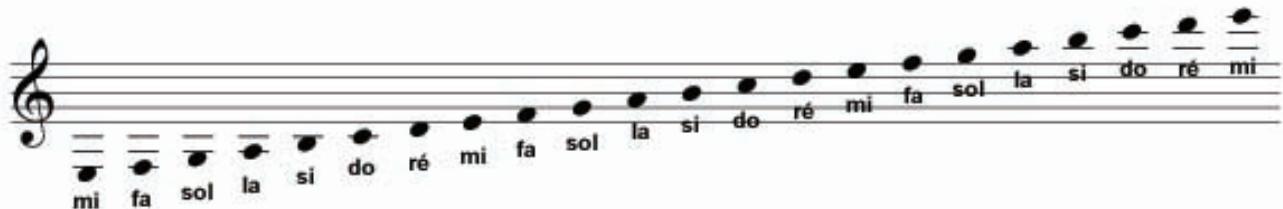
Pour comprendre la description des fonctionnalités de l'application, il est important de comprendre le jeu sur un manche de guitare.

Un manche de guitare est composée de 6 cordes (représentées horizontalement sur l'image ci-dessus) et de 24 frettes (verticallement 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).

### 1.3.2 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 ajoute 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 octaves. Par exemple, du premier mi au deuxième mi, nous avons une octave, c'est-à-dire toutes les notes différentes jusqu'à ce qu'on retombe sur la note de départ. Le premier mi sera donc appelé mi1, le deuxième mi2, et ainsi de suite.

Les notes représentées correspondent aux notes sur le manche de la guitare jusqu'à la 12ème frette. Au-delà de la 12ème frette, et donc au-delà du mi4, les notes sont représentées à l'octave du dessous avec une indication « 8va ». Par exemple, pour représenter le fa4, nous aurons un fa au même niveau que le fa3, avec l'étiquette « 8va » qui indique que ce fa doit être joué une octave au-dessus de celui écrit sur la portée.

## 2 Implémentation

### 2.1 Architecture de l'application

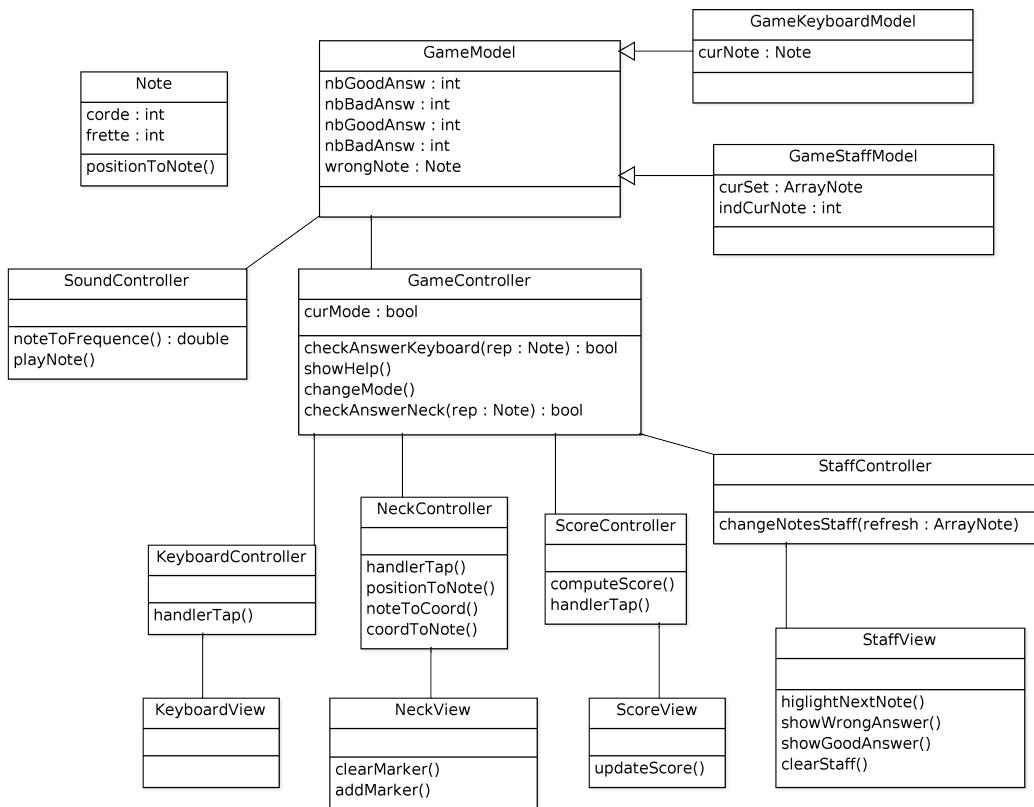


FIGURE 2.1 – Diagramme de classes

Paragraphe explicatif

## 2.2 Fonctionnalités

### 2.2.1 GameController

Cette classe est responsable de la logique du jeu. C'est elle qui vérifie les réponses données par l'utilisateur, mettre à jour son score, déléguer l'actualisation des vues au contrôleurs adéquats. Ce contrôleur contient également un attribut qui précise le mode courant du jeu : clavier/manche ou portée/manche dans curMode.

— changeMode : cette méthode réagence l'écran avec les vues nécessaires. La variable curMode est mise à jour.

— showHelp : cette méthode vide l'écran pour n'afficher que l'écran d'aide (certainement une WebView).

— CheckAnswerKeyboard prend en paramètre la note sélectionnée par l'utilisateur sur le clavier. Il fait correspondre la note actuellement présentée sur le manche en interrogeant le modèle (getCurNote).

— CheckAnswerNeck prend en paramètre la note sélectionnée par l'utilisateur sur le manche. Il fait correspondre la note actuellement présentée sur la portée en interrogeant le modèle.

En fonction du résultat de la fonction Check, le GameController met à jour le score de l'utilisateur dans le GameModel. Si la réponse est mauvaise le GameController stocke également la note erronée dans le GameModel (wrongAnswer).

### 2.2.2 GameModel

Les données seront stockées dans ce modèle, lui-même divisé en deux classes (par héritage) :

— GameKeyboardModel : contient pour seul attribut la note courante à deviner.

— GameStaffModel : contient un ensemble de notes présentées sur la portée, et l'indice dans cet ensemble de la note courante à deviner.

### 2.2.3 KeyboardController

Cette classe a pour vocation de réceptionner les événements en provenance de KeyboardView et de lancer les fonctions d'actualisation sur cette même vue. handlerTap reçoit en paramètre le nom de la note tapée au clavier et délègue la vérification de la question posée au GameController. Ce handler peut éventuellement être déplacé dans le GameController pour des raisons de praticité.

### 2.2.4 KeyboardView

C'est cette vue qui représente le clavier sur lequel l'utilisateur saisit sa réponse en mode Manche/Clavier. Cette vue doit intercepter le tap de l'utilisateur effectué sur un clavier (UIButton ou SegmentedControl). Les événements en provenance de cette vue sont dirigés vers l'objet KeyboardController. A priori aucun traitement à destination de cette vue n'est attendu. Cette vue doit représenter 35% de la taille de l'écran.

### 2.2.5 NeckController

Cette classe a pour vocation de réceptionner les événements en provenance de NeckView et de lancer les fonctions d'actualisation sur cette même vue. Elle contient des méthodes de conversion :

- coordToPosition : lorsque l'utilisateur tape sur le manche, les coordonnées correspondant à son tap sont données à une fonction qui a pour but de la transformer en position sur le manche. La position sur le manche est un couple <frette, corde>.
- Cette fonction traduit une position (couple <frette, corde>) sur le manche en note. Chaque frette représentant un demi-ton, on augmente la note de la corde donnée d'autant de demi-tons que le numéro de la frette, modulo les douze notes différentes.
- noteToCoord : Une note, selon le couple <frette, corde> est traduite en coordonnées sur le manche
- getClosestFret :

## 2.2.6 NeckView

Cette classe est une UIScrollView pour permettre à l'utilisateur de faire défiler le manche. Cette UIScrollView contiendra elle-même une UIImageView qui affiche effectivement l'image du manche. Les événements en provenance de cette vue seront dirigés vers l'objet NeckController. Cette vue permet d'afficher des pastilles de couleur sur les différentes notes du manche grâce à des méthodes d'ajout et de suppression : addMarker, deleteMarker. Ces méthodes ne doivent manipuler que des coordonnées du tap de l'utilisateur sur la vue.

Cette classe contient des méthodes d'ajout, de suppression et de mise à jour d'une pastille sur le manche.

Cette vue représente 55% de la taille de l'écran.

## 2.2.7 Note

Pour pouvoir positionner une note de manière unique sur le manche, il faut avoir le couple <corde, frette> on ne peut se contenter d'un nom de note, car cette note est disponible à plusieurs endroits sur le manche. Le but de cette classe est donc de mémoriser et convertir ce couple en une autre information utile (i.e : nom de note simple, note en valeur absolue, ...). positionToNote : cette méthode convertit le couple <corde, frette> en un nom de note (ex : do, sol, ré#).

## 2.2.8 ScoreController

Cette classe a pour vocation de lancer les fonctions d'actualisation sur ViewController. changeNotesStaff : cette fonction interroge le GameModel pour connaître les notes à afficher sur la portée. Il peut ainsi invoquer les méthodes : highlightNextNote, showWrongAnswer, showGoodAnswer et addNewNote.

## 2.2.9 ScoreView

Cette vue présente la barre de progression de l'utilisateur. Elle affiche également un bouton (UIButton) pour que l'utilisateur puisse changer de mode.

Les événements en provenance de cette vue sont dirigés vers l'objet ScoreController. Cette vue doit permettre de mettre à jour le pourcentage de bonnes/mauvaises réponses dans la barre de progression. Cette vue a une hauteur fixe de 50 points imposés par Apple.

## 2.2.10 SoundController

Cette classe est l'interface pour le jeu de son. Elle doit abstraire et simplifier l'utilisation d'une bibliothèque de lecture de son.

playNote : cette fonction prend une note en paramètre, convertit cette note en fréquence grâce à la méthode noteToFrequence et lance la lecture d'un son grâce à cette fréquence trouvée.

## 2.2.11 StaffController

Cette classe a pour vocation de lancer les fonctions d'actualisation sur ViewController.

changeNotesStaff : cette fonction interroge le GameStaffModel pour connaître les notes à afficher sur la portée. Il peut ainsi invoquer les méthodes :

- highlightNextNote : surbrillance jaune
- showWrongAnswer : surbrillance rouge
- showGoodAnswer : surbrillance verte

Le StaffController est lui même relié au GameController qui va invoquer les méthodes nécessaires en fonction de l'évolution du jeu.

## 2.2.12 StaffView

Cette vue est une UIImageView, elle affiche l'image de la portée. Les notes sont affichées par superposition sur cette image de base. Les événements en provenance de cette vue sont dirigés vers l'objet

StaffController. Cette vue doit permettre d'afficher des notes données en paramètre sur la portée et de colorer ces notes en rouge, vert, ou jaune. Cette vue est censée ne retourner/recevoir que des coordonnées, jamais des objets Note.