

# Rapport - Développement Mobile Magic Tiles

Kamarouzamane Combo,  
Hajanirina Randimbisoa  
M1 informatique

9 novembre 2018

## Résumé

Dans ce rapport, nous vous présentons le projet sur lequel nous avons travaillé dans le cadre de l'UE Développement pour Mobiles. Nous avons choisi de nous inspirer du jeu Magic Tiles et d'en faire une version pour Android[1] et iOS[2]. Nous vous exposons ici le principe du jeu, le fonctionnement de l'application et son développement, ainsi quelques points des deux codes.

**Mots clés :** Android Studio, Xcode, Java, Swift, Magic Tiles, Piano

## 1 Introduction

Dans le cadre de l'UE Développement pour mobiles, nous devions réaliser un projet sur Android et iOS. En effet, il nous était demandé de réaliser une application mobile sur les deux plateformes. Nous avons donc décidé de réaliser une version simple du jeu Magic Tiles, le jeu consiste à modéliser/simuler des touches de piano, tout en utilisant des différentes notions vues en cours comme : changements de configuration (changement de langue, passage de portrait à paysage), géolocalisation, Lecture audio, gestion des gestes courants (double tap, balayage ...) et l'utilisation de l'appareil photo. Pour commencer, nous vous présenterons d'abord quel type de jeu nous avons voulu développer, puis nous explorerons l'architecture des deux applications.

## 2 Description générale de l'application

Le jeu consiste à taper sur les touches noires ou blanches (selon le mode choisi) qui se présentent sur l'écran. À partir du moment où l'on commence à taper sur la première touche, la partie commence et un chronomètre est lancé. Ensuite, le joueur va devoir taper sur les touches noires (ou blanches), le plus rapidement possible. Enfin, lorsque la partie se termine. Celui qui détient le temps le plus rapide est le joueur qui a le meilleur score. Cependant, si le joueur taper par erreur sur une des mauvaises touches, la touche devient rouge et la partie est perdue.

Nous avons décidé de proposer aux joueurs deux modes de jeu :

- un mode normal : les touches sur lesquelles il faut appuyer sont noires et celles qu'il faut éviter sont blanches.
- un mode inversé : comme son nom l'indique, dès lors que ce mode est activé, les touches sont inversées, et il faut alors taper sur les touches blanches pour ne pas perdre.

Voici des captures d'écran du jeu sur iOS et Android

## 3 Architecture du code

### 3.1 Android

Lorsque l'application est lancée, un écran d'accueil apparaît et donne la possibilité au joueur de choisir entre commencer une nouvelle partie, de pouvoir se localiser ou accéder à la liste des scores enregistrés et grâce aux capteurs du smartphone, le joueur peut démarrer l'accéléromètre. Nous n'avons pas laissé la possibilité au joueur de reprendre une partie déjà entamée, étant donné que c'est un jeu dont le score se base sur la rapidité du joueur.

Si ce dernier décide de lancer une nouvelle partie, il pourra choisir de jouer en mode portrait ou paysage, de changer le mode (classique ou inversé). Enfin, dès que la partie commence, il n'a plus la possibilité de revenir en arrière. Il doit donc terminer la partie, soit en gagnant ou en perdant.

À partir du moment où le joueur gagne, il va être redirigé vers une nouvelle activité de l'application, qui affichera son score ainsi que le meilleur s'il y a des scores enregistrés au préalable, et si le joueur vient d'effectuer un nouveau record, cela sera affiché à la place du meilleur. Il aura alors la possibilité d'enregistrer son score s'il le désire, de rejouer ou bien tout simplement de revenir sur l'écran d'accueil.

Dans le cas où le joueur perd la partie, un écran affichera ce résultat ainsi que le meilleur score enregistré pour le moment s'il y en a un. Il n'aura dès lors que la possibilité de rejouer ou de revenir au menu.

Lorsque le joueur décide d'enregistrer son score, une nouvelle activité est lancée, affichant son score, la date ainsi que l'heure à laquelle il a terminé la partie. De plus, une zone pour que le joueur puisse entrer son nom est disponible. Cependant, il devra entrer au moins 3 caractères, et son nom ne devra pas excéder une quinzaine de lettres. Il pourra aussi se prendre en photo, s'il le souhaite.

Enfin, son score sera affiché dans la liste des scores dès sa validation. Cette liste affiche ainsi les différents scores sauvegardés, avec le meilleur score tout en haut de la liste et le dernier à la fin. Chaque entrée comporte le rang dans le classement, le nom du joueur, le score à proprement parler ainsi que la date. Le joueur a la possibilité de supprimer un score s'il le souhaite, en appuyant sur une entrée. Néanmoins, il n'a pas l'autorisation de modifier un score.

Enfin, il peut accéder au menu en appuyant sur le menu dans la barre du haut. Pour la version Android, nous avons utilisé une dizaine de classes. Parmi elles, deux servent à utiliser la base de données pour enregistrer les scores, huit désignent les activités de l'application, et les autres classes sont utilisées pour le fonctionnement des autres. Nous avons donc les classes *Accueil*, *Game*, *ResultNeg*, *ResultPos*, *Enregistrer* et *ListeScores*, *maps*, *photo* qui constituent les activités de l'application, *Chronometer* et *Score* qui sont utilisées pour le score et la base de données, ainsi que *MyDBAdapter* et *ScoresAdapater* pour gérer la base de données.

Bien entendu, à chacune des classes activité est associée une fichier xml dans lequel sont placés les différents composants (boutons, listview, . . . ). Nous allons faire un petit résumé de ces xml et de ces classes activité.

Pour l'ensemble des xml créés, nous avons décidé de choisir un Constraint Layout, afin de pouvoir déployer facilement et efficacement notre application sur tout type d'écran. Nous avons également choisi d'utiliser des Guidelines afin de placer nos éléments aux bons endroits.

### 3.1.1 Accueil

L'écran d'accueil affiche le titre du jeu et comporte quatre boutons, un lancer une nouvelle partie, un pour la localisation, un autre pour lancer l'accéléromètre et aussi pour accéder à la liste des scores enregistrés. Lorsqu'on appuie sur le premier, on est dirigé vers l'activité du jeu à proprement parler, tandis que l'autre nous envoie sur l'activité gérant les scores.



### 3.1.2 Game

Dans cette activité, nous avons décidé d'afficher un chronomètre dans la partie supérieure de l'écran, à côté duquel nous avons placé un ToggleButton pour changer le mode de jeu, accompagné de sa légende. Une guideline sépare la partie supérieure de l'écran du reste afin de disposer au mieux les éléments. Nous avons choisi de modéliser les touches de piano par de simples boutons, rangés en quatre lignes. Chaque ligne comporte quatre boutons. Dans cette version, tous les boutons sont de la même taille et occupent toute la place disponible en dessous de la guideline. Afin de pouvoir avoir ce résultat et qu'il soit similaire sur tout type d'écran, nous avons utilisé au mieux les fonctionnalités du Constraint Layout, ce qui n'était pas possible avec un Relative Layout, qui s'appuie surtout sur les positions et des tailles prédéfinies.

Intéressons nous maintenant au code de cette activité. Dans un souci de clarté, nous allons parler uniquement du mode classique, l'autre mode fonctionnant exactement de la même façon, mais les couleurs étant inversées.

Liste des fonctions utilisées dans cette activité :

- void initialisation(int l1, int l2, int l3, int l4, boolean b, int c1, int c2)
- void changementLigne(int c1, int c2)
- void finDefilement(int nblignesrestantes, int c2)

L'objectif ici était de pouvoir ajouter une couleur aux boutons pour simuler des touches de piano. Ainsi, dès que cette activité est lancée, nous générerons un nombre aléatoire compris entre 1 et 4 pour chacune des lignes afin de déterminer quel bouton sera noir. Les autres seront dès lors coloriés en blanc (appel de la fonction initialisation).

Une seule ligne de boutons est cliquable - nous l'appelleront «ligne 0» -, celle qui se situe tout en bas de l'écran, le reste des boutons étant désactivé. Nous n'avons donc ajouté des écouteurs qu'aux boutons de la ligne 0. Voici le déroulement de l'action faite lorsqu'un joueur appuie sur l'un de ces boutons :

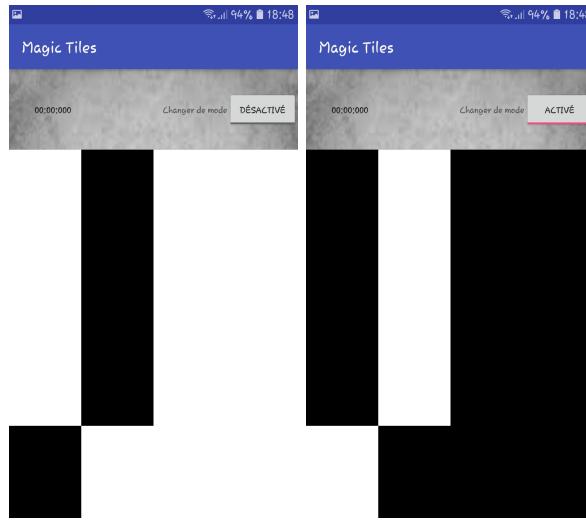
- si la partie n'a pas encore débuté, le chronomètre sera démarré et le choix du mode ne sera dès lors plus accessible.

— nous allons toujours vouloir savoir dans quel mode nous nous trouvons, étant donné qu'il faut mettre une couleur différente en fonction de cela, ainsi les paramètres des fonctions appelées changeront également.

Nous allons également vouloir déterminer si l'utilisateur a appuyé sur une touche blanche ou noire. Dans ce cas-ci, si la touche est blanche, la partie est perdue, la touche sélectionnée est peinte en rouge, on arrête le chronomètre et on accède à l'activité des résultats négatifs. Si au contraire c'est une touche noire, nous faisons défiler les lignes suivantes (appel à la fonction changement-Ligne) et le jeu continue.

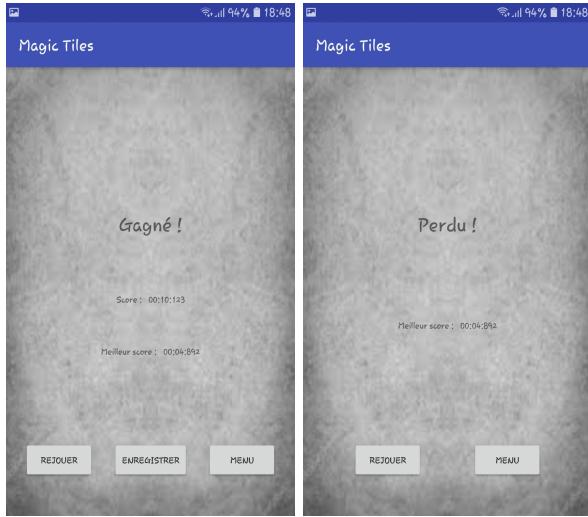
Afin de pouvoir arrêter le jeu, nous allons également utiliser une variable dans laquelle nous allons mettre le nombre de touches, qui sera décrémenté à chaque fois que le joueur appuiera sur un bouton. Lorsque ce nombre sera inférieur ou égal à quatre, nous allons lancer la fonction *finDefilement*, afin de colorier les dernières touches en blanc.

Enfin, lorsque la partie est terminée, nous arrêtons le chronomètre, récupérons sa valeur, ainsi que la date et l'heure de la fin de la partie. Nous envoyons ensuite ces données à l'activité suivante, celle des résultats positifs, afin de pouvoir les afficher et éventuellement les enregistrer dans la liste des scores. Un des points délicats de ce code était de pouvoir sauvegarder l'état de la partie si le joueur décidait de passer du mode paysage au mode portrait et inversement. Ici nous devions donc faire attention aux données, et devions les enregistrer au cas où nous nous retrouvions dans ce cas là. Nous avons donc utilisé la fonction *onSavedInstanceState* afin de récupérer l'état des lignes (pour savoir de quelle couleur peindre les boutons), des valeurs randoms (au cas où la partie n'a pas encore débuté), le nombre de touches, l'état du chronomètre ainsi que le mode (classique ou inversé).



### 3.1.3 ResultatPos - ResultatNeg

Ces deux activités sont similaires, mis à part le fait que sur l'une on ne peut pas afficher ni enregistrer son score. Sur ces écrans, on affiche le résultat de la partie (gagné ou perdu), le meilleur score s'il y en a un ainsi que le score (si gagné). On donne ensuite la possibilité au joueur de rejouer, d'enregistrer son score ou d'accéder à l'écran d'accueil. Si on gagne, on récupère les données transmises par l'activité Game dans l'intent et on les retransmet ensuite pour l'activité Enregistrer si on appuie sur ce bouton.



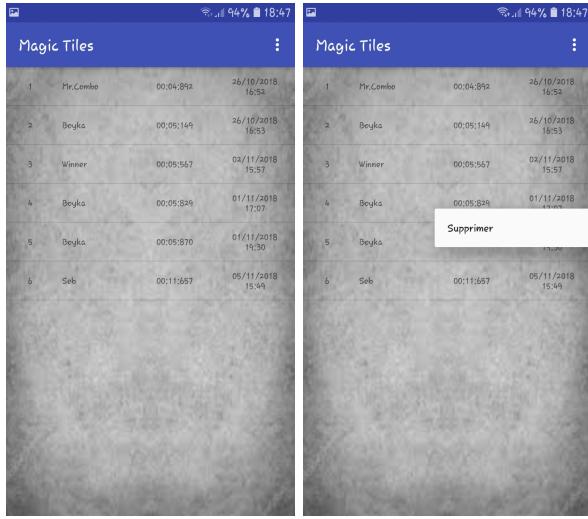
### 3.1.4 Enregistrer

Dans cette activité, on dispose de trois boutons, l'un pour valider son enregistrement, un autre pour prendre une photo et un autre pour annuler (pour revenir au menu). Le joueur a la possibilité d'écrire son nom, mais ne doit pas dépasser une quinzaine de caractères, et doit en entrer au moins trois. Les données seront également transmises à l'activité ListeScores.



### 3.1.5 ListeScores

Dans cette activité, nous utilisons les classes *Score*, *ScoresAdapter*, ainsi que *MyDBAdapter* pour pouvoir créer des scores, les afficher et les enregistrer dans une base de données. Nous avons donc choisi de créer des cellules personnalisées pour afficher les scores, et utilisons donc un cell-layout.xml. Le joueur peut supprimer des entrées dans la liste ou tout simplement revenir au menu principal. Lorsque l'utilisateur supprime des éléments dans la liste, celle-ci est mise à jour ainsi que le classement.

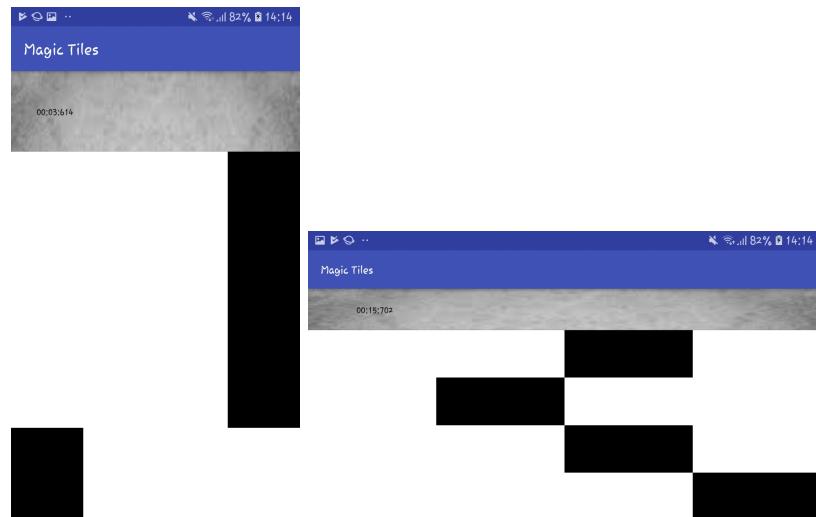


### 3.1.6 MyDBAdapter – ScoresAdapter – Score

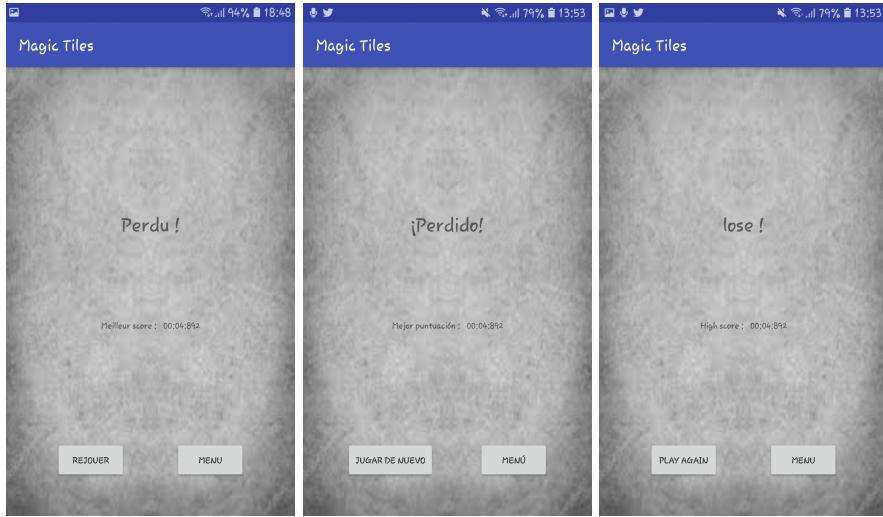
La classe *MyDBAdapter* permet de créer une base de données utilisant sqlite. Nous avons décidé d'appeler notre base de données «*scoresdataBase.db*», et de créer une table «*tableScores*» qui contient quatre colonnes : id, score, date et nom. Afin de pouvoir afficher une liste triée, nous avons utilisé une fonction, *getAllScores*, qui renvoie un *ArrayList<Score>* et qui trie la liste en fonction de la position du curseur dans la table qui est elle-même triée en fonction de la colonne des scores. La classe *ScoresAdapter* permet de récupérer les attributs d'un score (rang, nom, valeur du score et date) et de les afficher dans une cellule (celllayout). Les classes *MyDBAdapter* et *ScoresAdapter* sont très liées et sont essentielles pour le bon fonctionnement de la liste des scores.

### 3.1.7 Changement de configuration

L'application, gérer les changements de configuration sans perte de données. Lorsque l'utilisateur configure son téléphone en plein partie soit en changeant de langue ou bien en passant du mode portrait au mode paysage, les données seront sauvegarder. En cas de changement de configuration, l'activité courante est détruite (avec *onDestroy*) puis une nouvelle activité adaptée à la nouvelle configuration est créée (appel à *onCreate* ).



L'application prend en compte la langue de l'utilisateur.



### 3.1.8 Lecture audio

Different son sont instancier dans l'application, lorsque le joueur va appuyer sur le touch du piano une note sera jouée.

```
public void playDo(){
    stopGameSong();
    gameOverSong = MediaPlayer.create(this, R.raw.song_do);
    gameOverSong.setOnCompletionListener(
        new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mp) {
                stopGameSong();
            }
        });
    gameOverSong.start();
}

public void playRe(){
    stopGameSong();
    gameOverSong = MediaPlayer.create(this, R.raw.re);
    gameOverSong.setOnCompletionListener(
        new MediaPlayer.OnCompletionListener() {
            @Override
            public void onCompletion(MediaPlayer mp) {
                stopGameSong();
            }
        });
    gameOverSong.start();
}
```

A la fin de la partie, deux type de son seront joué, un en cas de victoire et l'autre en cas de défaite .

## 3.2 iOS

Dans cette partie nous allons vous montrer le fonctionnement de l'application sous iOS. Nous avons utilisé Xcode pour avoir développer l'application sous iOS[3]. L'application est composée de

5 écrans :

- L'écran d'accueil, sur lequel l'application s'ouvre
- Game.swift : l'écran du jeu, accessible depuis toute l'application
- Save.swift : l'écran de sauvegarde, accessible à la fin d'une partie
- Highscores.swift : l'écran avec la liste des scores sauvegardés, accessible depuis le menu et après une sauvegarde
- Geolocatilisation.swift : elle fournie la position du joueur

### 3.2.1 Activity : Game

Principe : faire défiler les 20 notes de la partition vers le bas d'une grille de 4x4 boutons, en actionnant les boutons de la dernière ligne. Une note est un entier compris entre 0 et 3, une note 3 apparaitra dans cette ordre sur les boutons bp33, bp23, bp13 et le bouton actif de la dernière ligne bp03. Le chiffre des dizaines du bouton correspond à sa ligne, et celui des unités à sa colonne, bp00 est en bas à gauche et bp33 est en haut à droite. Il y a donc un seul bouton représentant la note par ligne, remarquable par une couleur différente des autres boutons de la ligne. Lorsque bouton correspondant à une note est frappé, la partition descend sur la grille.

Les méthodes :

- viewDidLoad : initialise les attributs, remplit la partition de 20 notes aléatoire, initialise la grille avec les 4 premières notes
- setNote : met les couleurs sur les boutons d'une ligne pour en faire une note
- frappeOK : utilise setNote pour faire descendre la partition
- chronoStart : lance le chronomètre
- chronoStop : arrête le chronomètre
- prepare : pour passer le chronomètre à Save.swift

Code d'un des boutons de la dernière ligne :

```
@IBAction func frappeBP02(_ sender: Any) {
    if(isPlaying) {
        if(bp_02.backgroundColor == UIColor.black || bp_02.backgroundColor == UIColor.green)
            if(avancement == 0) { chronoStart() }
            if(avancement == avancementMax-1) { chronoStop() }
            frappeOK()
    }
    else {
        bp_02.backgroundColor = UIColor.red
        chronoStop()
        bp_save.isHidden = true
    }
}
```

Code de la méthode setNote :

```
func setNote(noteLigne: [UIButton], avancementNote: Int) {
    noteLigne[0].backgroundColor = UIColor.white
    noteLigne[1].backgroundColor = UIColor.white
    noteLigne[2].backgroundColor = UIColor.white
    noteLigne[3].backgroundColor = UIColor.white
    var colorNote = UIColor.black
    if(avancementNote == 0 || avancementNote == avancementMax-1) {
        colorNote = UIColor.green
    }
}
```

```

if(avancementNote<avancementMax) {
    switch partition[avancementNote] {
        case 0:
            noteLigne[0].backgroundColor = colorNote

        case 1:
            noteLigne[1].backgroundColor = colorNote

        case 2:
            noteLigne[2].backgroundColor = colorNote

        case 3:
            noteLigne[3].backgroundColor = colorNote

        default:
            print("erreur switch")
    }
}

```

Code la méthode frappeOK :

```

func frappeOK() {
    avancement += 1
    setNote(noteLigne: noteLigne0, avancementNote: avancement)
    setNote(noteLigne: noteLigne1, avancementNote: avancement+1)
    setNote(noteLigne: noteLigne2, avancementNote: avancement+2)
    setNote(noteLigne: noteLigne3, avancementNote: avancement+3)
}

```

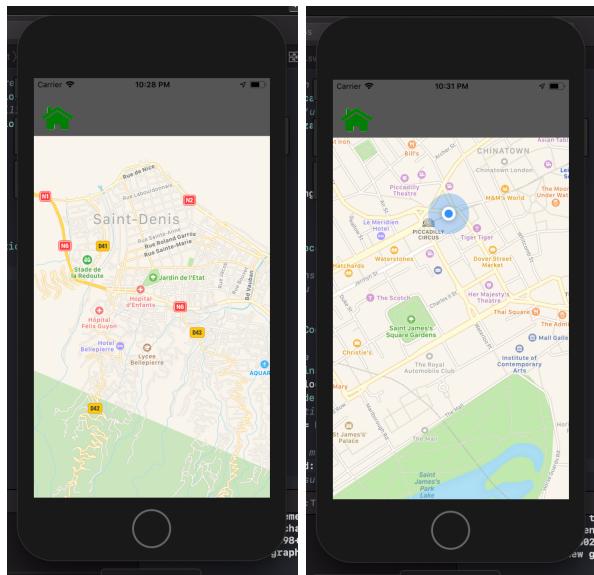
### 3.2.2 Activity : Save

Cette classe récupère le temps du chronomètre et permet à l'utilisateur de rentrer un pseudonyme pour sauvegarder son score. Elle envoie dans une méthode prepare, le temps, le pseudonyme, la date et l'heure de la sauvegarde à Highscores.swift

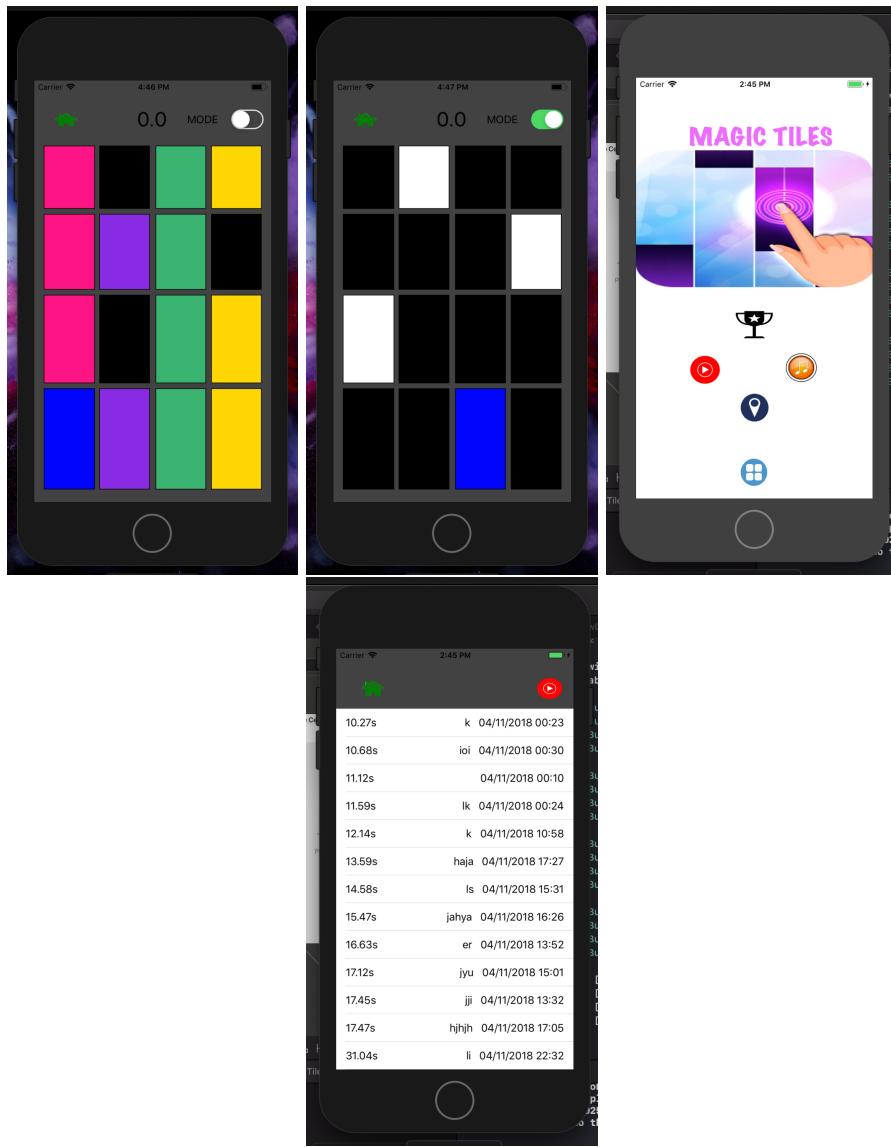
### 3.2.3 Activity : Highscores

C'est la classe qui hérite de UITableViewDelegate et UITableViewDataSource. C'est elle qui va contenir le tableView des scores triés avec persistance des données. Elle reçoit les données de Save.swift, elle les insère dans le coreData. et il est possible de supprimer un élément de la liste avec un geste glisser sur la gauche.

### 3.2.4 Activity : Geolocalisation



### 3.2.5 Autres captures d'écran



## 4 Conclusion générale

Cette UE nous a permis de pouvoir développer des applications sur les deux plateformes, Android et iOS, de créer plusieurs projets. Ainsi, celui décrit dans ce rapport nous a également permis de consolider les connaissances acquises durant ce cours. Grâce à cela, nous avons pu travailler en autonomie et de ce fait, effectuer nos propres recherches, tester de nouvelles choses, faire des erreurs, et en apprendre davantage grâce à elles. Dans l'ensemble, c'était une bonne expérience, et qui plus est, enrichissante.

- [4] [5] [6]
- [7] [8]

## Références

- [1] Doc android. <https://developers.google.com/maps/documentation/android-sdk/start?hl=fr>.
- [2] Swing. <https://docs.oracle.com/javase/tutorial/uiswing/index.html>.
- [3] swift. <https://www.a-j-evolution.com/tutoriels/swift/>.
- [4] Tkinter. <https://docs.python.org/2/library/tkinter.html>.
- [5] Sensorevent. <https://developer.android.com/reference/android/hardware/SensorEventl>.
- [6] Accelerometre. [https://www.frandroid.com/android/developpement/3039\\_tutorial-se-servir-de-laccelerometre](https://www.frandroid.com/android/developpement/3039_tutorial-se-servir-de-laccelerometre).
- [7] Blog swift. <https://swift.org/blog/swift-4-1-released/1>.
- [8] Swift book. <https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html>.