Rapport de Projet : Memoris

AMZALAC Avi - CADERBY Grégory

L3 Informatique - 15 Mai 2020

Table des matières

Résumé

Dans le cadre de l'unité d'enseignement "Développement Mobile 2", nous avons réalisé un travail de groupe centré sur l'élaboration d'une application fonctionnant sous Android. L'application doit proposer : plusieurs écrans, une présentation sous forme de listes, une sauvegarde persistante et fonctionner correctement sur tout type d'écran (grand, petit...), en mode portrait et paysage : ressources alternatives sous Android en utilisant les contraintes sur les composants graphiques.

1 Introduction

"Memoris" est une application qui est catégorisée dans les jeux cérébraux. Nous avions pour but de créer une application qui exercerait la boucle phonologique et la mémoire de travail de l'utilisateur. Dû aux contraintes de la période, nous avons développé le projet uniquement sous Android. Toutefois nous tenions à ajouter quelques fonctionnalités supplémentaires tout en respectant le cahier des charges.

2 Description générale de l'application

Comme cité précédemment, l'application est un jeu mobile disponible en langue française et anglaise qui permet à l'utilisateur d'exercer sa mémoire. Le jeu propose plusieurs niveaux de difficultés. Selon celles-ci, un diaporama de X images défile et amène ensuite l'utilisateur à sélectionner les noms des images qu'il a visionnées. Chaque erreur est enregistrée dans une base de données qui au travers d'un tableau de score, permet à l'utilisateur de suivre sa progression. L'application est structurée de la manière suivante :

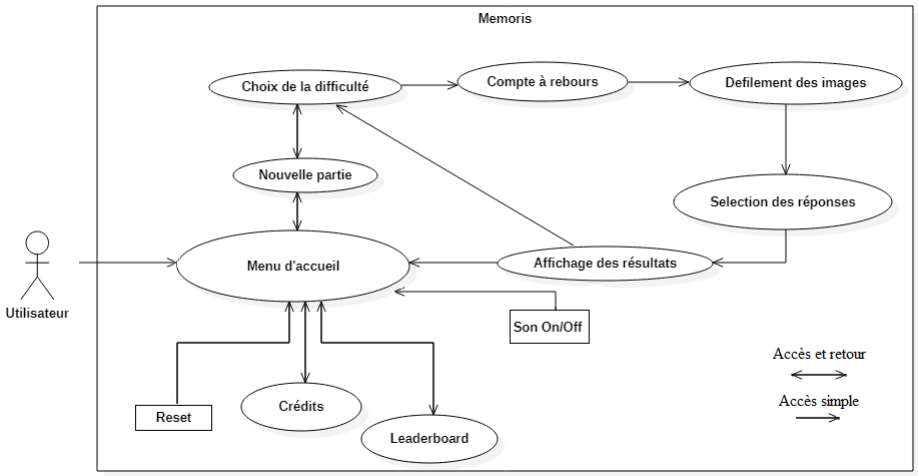

Image 1 : Représentation graphique de la structure

— **Le Menu d'accueil** : C'est l'activité principale. Depuis ce menu, on peut accéder à divers écrans tels que les crédits, le tableau des scores ou encore faire une nouvelle partie.

1. **Le bouton Son On/Off** : Ce bouton permet d'activer/désactiver le service de fond sonore
2. **Le bouton Reset** : Ce bouton permet l'effacement des données de la base de données. Il vide entièrement le tableau de scores et en prépare un nouveau pour l'accueil des prochains scores.

— **Le Menu de Nouvelle partie** : Pour une meilleure compréhension et lecture, nous avons volontairement distingué l'instanciation d'une partie avec le choix de la difficulté sur l'image 1. Toutefois, la réalité est un peu différente. En effet, lorsque l'utilisateur désire commencer une partie, l'application l'emmène directement sur la sélection de difficulté. (Il n'y a pas de "réelle" activité <Nouvelle partie>)

Le menu de cette activité propose quatre choix de difficulté à l'utilisateur :

- **Facile** - Le diaporama est composé de **cinq** images au total.
- **Moyen** - Le diaporama est composé de **six** images au total.
- **Difficile** - Le diaporama est composé de **sept** images au total.
- **Cauchemar** - Le diaporama est composé de **huit** images au total.

— **L'écran de compte à rebours** : Cette activité intermédiaire permet à l'utilisateur de se préparer et de ne commencer le jeu qu'à son signal. Un décompte de trois secondes s'effectuera si le bouton *Prêt* est appuyé. Une fois le temps écoulé, le jeu commence et les images défilent.

— **L'écran de Jeu** : Cette activité est le cœur du jeu. L'application, dépendant de la difficulté choisie fait défiler un nombre d'images sélectionnées aléatoirement et sans doublons dans les ressources de l'application. Lorsque le diaporama est terminé, l'utilisateur est automatiquement dirigé vers l'écran des réponses.

— **L'écran de Réponse** : Dans cette activité, plusieurs choix (sous formes de TextView) sont proposés à l'utilisateur. Ils sont composés bien évidemment des bonnes réponses mais aussi de réponses pièges, le tout affiché dans un ordre aléatoire.

En bas de l'écran, un compteur d'erreur s'incrémente à chaque mauvaise réponse de la part de l'utilisateur. Si le compteur atteint son maximum, ou que toutes les bonnes réponses ont été trouvées, la partie s'achève et l'utilisateur est redirigé vers l'écran de résultat.

L'utilisateur peut aussi utiliser une aide grâce au bouton d'indice. Une bonne réponse sera alors retirée des propositions.

— **L'écran de Résultat** Cette activité clos un cycle de vie d'une partie. Il présente le nombre de bonnes réponses que l'utilisateur a trouvé. L'utilisateur a le choix entre recommencer une partie (retour sur le choix de difficulté) ou de quitter (retour sur le menu d'accueil.

3 Architecture du code sous Android

Pour se donner une idée du développement à réaliser nous avons imaginé un cycle de vie d'une partie classique (un cas d'utilisation) en précisant les effets de bord mais aussi ce que le système effectue durant chaque action. Voici donc ce sur quoi nous avons abouti [1] :

3.1 Cycle de vie de l'application - côté Utilisateur

Étape 1 - Appuie sur l'icône de l'application.

Étape 2 - Appuie sur le bouton de Nouvelle Partie.

Étape 3 - Appuie sur une difficulté.

Étape 4 - Appuie sur le bouton *Prêt*.

Étape 5 - Attente de trois secondes.

1. La connexion entre l'utilisateur et le système est implicitement représentée : Les étapes 1 et A se déroulent en même temps, B2, C3, etc...

Étape 6 - Visualisation des X^2 images.

Étape 7 - Joue en appuyant sur les boutons de réponses.

Étape 8 - Obtient son score et fait le choix de recommencer ou non

3.2 Cycle de vie de l'application - côté Système

Étape A - Démarrage de l'application

Étape B - Affichage de l'activité *Difficulty_page*.

Étape C - Affichage de l'activité *Timer_page*.

Étape D - Démarrage du compteur de trois secondes.

Étape E - Décrémentation du compteur.

Étape F - Création de sept ArrayList : List_mot, List_Items, List_affiche, List_affiche_image List_reponse, list_rep et sliderItems :

List_mot est composé de tous les noms d'objets de jeu [3] récupérés par le strings.xml pour permettre l'adaptation du code si la langue change.

List_Items est composé de tous les drawables de jeu [4].

List_affiche et List_affiche_image sont générés aléatoirement et sans remise respectivement à partir de List_mot et List_items. *La graine de génération est la même.*

List_reponse et list_rep sont générés aléatoirement et sans remise respectivement à partir de List_affiche et List_affiche_image. *La graine de génération est la même.*

sliderItems est composé des éléments de list_rep, c'est cette liste qui permet l'affichage des images (les bonnes réponses) dans le diaporama.

Affichage de l'activité de jeu propre à la difficulté.

Étape G - Affichage des X boutons nommés grâce à List_affiche.

Lors d'un appui, appel à la méthode de vérification qui utilise List_reponse Affichage de l'activité Results_page lorsque les conditions sont validées.

Étape H - Affiche le nombre de bonnes réponses

Inscrit le numéro de l'essai (auto-incrémenté), la difficulté choisie et le score à l'intérieur d'un objet Score et l'enregistre dans la base de données Memoris.db. Propose à l'utilisateur de recommencer une partie ou de quitter vers l'accueil.

3.3 Structure des activités principales

Chaque activité comporte une partie de codes commune avec les autres. Elle est généralement composée de la gestion : des services de son, des entrées utilisateurs sur les boutons du téléphone, des états du téléphone et des méthodes permettant le passage à l'activité suivante. Pour une meilleure lisibilité nous ne traiterons pas de ces différentes parties de codes dans cette section.
Lors de la création de l'application nous nous sommes tout de suite heurtés aux limites qu'imposait le support de librairie de base *android* que proposait Android Studio. Nous avons alors choisi de migrer vers le support de librairie *androidx* pour nous permettre de créer les éléments que notre application exigeait.

Aussi nous avons dû apprendre l'usage de la librairie *viewPager2* et le fonctionnement des *fragments* notamment pour la création du diaporama.

Abordons désormais plus en détails le fonctionnement des principales activités de l'application en se concentrant sur leur méthode onCreate().

2. Varie selon la difficulté
3. L'ensemble des mots que l'on a décidé d'incorporer comme réponses
4. L'ensemble des images que l'on a décidé d'incorporer comme réponses

3.3.1 Activité Jeu

L'activité Jeu est le cœur de l'application. C'est elle qui gère la création des données requises pour le fonctionnement du jeu. Elle affiche aussi le diaporama d'images.

Le code ci-dessous est disséqué en plusieurs parties pour une meilleure compréhension

Dans cette section on assigne l'objet viewPager2 et on crée puis remplit les listes : List_mot qui sera composée de R.string *(tous les noms des objets pouvant être utilisés dans le diaporama)* donc d'integers et la List_Items qui sera elle, remplie, des drawables du jeu *(toutes les images des objets pouvant être utilisés dans le diaporama)* sous forme d'integers aussi.

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.jeu);
    viewPager2 = findViewById(R.id.viewPagerImageSliderEasy);
    viewPager2.setUserInputEnabled(false);

    // CREATION LISTE DES MOTS DU JEU ( A COMPLETER A CHAQUE AJOUT DE MOT)
    ArrayList<Integer> List_mot = new ArrayList<>();
    List_mot.add(R.string.apple);
    [...] // => Tous les ajouts a List_mot

    // CREATION LISTE DES IMAGES ( A COMPLETER A CHAQUE AJOUT D'IMAGE)
    ArrayList<Integer> List_Items = new ArrayList<>();
    List_Items.add(R.drawable.apple);
    [...] // => Tous les ajouts a List_Items
```

Voici une section extrêmement importante dans le fonctionnement du jeu.
Cette partie, en fonction de la difficulté, sert à remplir des listes aléatoirement sans remise parmi les mots et les images disponibles en ayant la même graine de génération. Elle sert implicitement à faire le lien entre le contenu du *string.xml* et du dossier *drawable*. Une fois les données requises générées, à savoir : une liste d'images pour le diaporama, une liste des bonnes réponses, une liste des mauvaises réponses et une liste de mots pour les noms des boutons "bonnes/mauvaises réponses", on peut alors transmettre à l'activité suivante ces informations grâce à leur attribut *static* préalablement défini dans le code.

```
    // LISTE DES MOTS AFFICHES POUR LE JOUEUR
    ArrayList<Integer> List_affiche_image = new ArrayList<>();
    int x;
    int nb_mot_affiche;
    switch(Difficulty_page.difficulty){
        case R.id.button_easy:
            nb_mot_affiche = 12;
            break;
        case R.id.button_medium:
            nb_mot_affiche = 14;
            break;
        case R.id.button_hard:
            nb_mot_affiche = 16;
            break;

        default:
```

```
        nb_mot_affiche = 18;
        break;
}

for(int i = 0; i< nb_mot_affiche; i++){
    do{ x=(int)(Math.random()*(List_mot.size()-1+1)+0);}
    while(List_affiche.contains(List_mot.get(x)));
    List_affiche.add(List_mot.get(x));
    List_affiche_image.add((List_Items.get(x)));
}
// LISTE DES BONNES REPONSES
switch(Difficulty_page.difficulty){
    case R.id.button_easy:
        nb_reponse = 5;
        break;
    case R.id.button_medium:
        nb_reponse = 6;
        break;
    default:
        nb_reponse = 7;
        break;
}
for(int i = 0; i< nb_reponse; i++){
    do{ x=(int)(Math.random()*(List_affiche.size()-1+1)+0);}
    while(List_reponse.contains(List_affiche.get(x)));
    List_reponse.add(List_affiche.get(x));
    List_rep.add(List_affiche_image.get(x));
}
// LISTE DES MAUVAISES REPONSES
for(int j = 0; j< List_affiche.size();j++){
    if(List_reponse.contains(List_affiche.get(j))){

    }else{
        List_mauvaises_reponses.add(List_affiche.get(j));
    }
}
```

Enfin nous pouvons générer une liste de sliderItem qui va accueillir les images réponses puis les afficher avec un écart de 1,5 secondes.

```
//CREATION DU SLIDER
final List<SliderItem> sliderItems = new ArrayList<>();
for (int i=0;i<nb_reponse;i++) {
    sliderItems.add(new SliderItem(List_rep.get(i)));
}

viewPager2.setAdapter(new SliderAdapter(sliderItems, viewPager2));
viewPager2.registerOnPageChangeCallback(new
ViewPager2.OnPageChangeCallback() {
    @Override
    public void onPageSelected(int position) {
        super.onPageSelected(position);
        sliderHandler.removeCallbacks(sliderRunnable);
        sliderHandler.postDelayed(sliderRunnable, 1500); //Durée pour 1500
    }
```

```
        });
```

Important : La mise en œuvre de ce diaporama a impliqué la création d'une classe SliderItem et d'une classe adapteur SliderAdpater.

3.3.2 Activité X_User_Interface

L'activité X_User_Interface *(avec X la difficulté)* est l'activité d'interaction avec l'utilisateur. C'est là où le jeu se déroule, l'utilisateur a le choix parmi plusieurs réponses et doit appuyer sur celles qu'il a préalablement visionnées. L'activité se sert des données que *Jeu* lui a transmis pour son fonctionnement. Une fois les conditions de défaite ou de victoire validées, la difficulté choisie et le score du joueur sont envoyés à l'activité suivante sous forme de *putExtra*. On pourra alors insérer ces éléments dans la base de données.

Le code ci-dessous est disséqué en plusieurs parties pour une meilleure compréhension. La même démarche est adoptée pour les différentes difficultés.

On assigne et nomme tous les boutons grâce aux données transmises.

```
protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.easy_user_interface);
      error_text = findViewById(R.id.error_counter);
      indice_text = findViewById(R.id.help);

      text_1 = findViewById(R.id.text_1);
      [...] // => tous les noms des boutons
      List_button.add(text_1);
      [...] // => tous les boutons
```

Cette méthode permet d'effectuer une vérification sur le bouton choisi et le fait disparaitre. Si le choix est mauvais on enlève le mot de la liste de mauvaises réponses.

```
    public void push_button(View view) {

        switch (view.getId()) {
            case R.id.text_1:
                text_1.setVisibility(View.INVISIBLE);
                recuperation_indice_mauvaise_rep(view);
                //  verification(view);
                if (!verification(view)) {
                    Jeu.List_mauvaises_reponses.remove
                    (recuperation_indice_mauvaise_rep(view));
                }
                afficher_list();
                break;
            case R.id.text_2:
                text_2.setVisibility(View.INVISIBLE);
                recuperation_indice_mauvaise_rep(view);
                //  verification(view);
                if (!verification(view)) {
                    Jeu.List_mauvaises_reponses.remove
                    (recuperation_indice_mauvaise_rep(view));
                }
```

```
            afficher_list();
            break;
        [...] // tous les cases requis
```

Vérification qui permet de savoir si le mot choisi est correct ou non, s'il est correct on retourne vrai sinon on retourne faux et on augmente le nombre d'erreurs de 1. Si on a fait une erreur on débloque l'option indice, si on arrive à la limite du nombre d'erreurs acceptées, on va sur la page de résultats.

```
public boolean verification(View view) {
    int flag = 0;
    int i = 0;

    TextView b = (TextView) view;
    String buttonText = b.getText().toString();
    System.out.println(" NOM DU TEXT DU BOUTON : " + buttonText);

    //        System.out.println(getResources().getString(text1));
    while (flag == 0 && i < Jeu.List_reponse.size()) {

        int text1 = Jeu.List_reponse.get(i);

        if (buttonText.equals(getResources().getString(text1))) {
            System.out.println(getResources().getString(text1) + " OK");
            flag = 1;

            right_answer++;
        } else {

            System.out.println(getResources().getString(text1) + " Faux");
        }
        i++;
    }
    if (flag == 1) {
        System.out.println("Nombre d'erreur: " + nb_error + "/3");
        if (right_answer == Jeu.nb_reponse) goto_resultspage();

        return true;

    } else {
        nb_error++;
        // image indice
        if(nb_error< 1){
            indice_text.setBackgroundResource(R.drawable.lock);
        }else{

            indice_text.setBackgroundResource(R.drawable.lightbulb);
            flag_indice = true;
        }
        System.out.println("Nombre d'erreur: " + nb_error + "/3");
        error_text.setText(nb_error + "");
        if (nb_error >= 3) {
            goto_resultspage();
        }
```

```
            return false;
        }
    }
```

Cette méthode permet de récupérer l'indice du mot dans la liste des mauvaises réponses.

```
    public int recuperation_indice_mauvaise_rep(View view) {
        int flag = 1;
        int i = 0;
        TextView b = (TextView) view;
        String buttonText = b.getText().toString();
        while (flag == 1 && i < Jeu.List_mauvaises_reponses.size()) {
            int text1 = Jeu.List_mauvaises_reponses.get(i);
            if (!(buttonText.equals(getResources().getString(text1)))) {
                i++;

            } else {
                flag = 0;
            }
        }
        System.out.println("i = " + i + "  textbouton= " + buttonText + "  ");
        return i;
    }
```

Enfin, on peut afficher l'indice sous forme de Toast de courte durée.

```
    public void indice_toast(View view) {
    if(flag_indice == true){
        int text = Jeu.List_mauvaises_reponses.get(0);
        String indice = getResources().getString(text);
        Toast toast = Toast.makeText(Easy_User_Interface.this,
        getResources().getString(R.string.help) + indice, Toast.LENGTH_SHORT);
        View toastview = toast.getView();
        TextView tv = toastview.findViewById(android.R.id.message);
        tv.setTextSize(21);
        tv.setTextColor(Color.parseColor("#000000"));
        tv.setCompoundDrawablesWithIntrinsicBounds(R.drawable.toast, 0, 0, 0);
        tv.setCompoundDrawablePadding(15);
        toastview.setBackgroundColor(Color.parseColor("#00000000"));
        toast.show();} else { ;}
    }
```

Une fois la partie achevée on envoie les données de scores et de difficulté pour la page de résultats.

```
public void goto_resultspage() {
        Intent intent = new Intent(Easy_User_Interface.this, Results_page.class);
        intent.putExtra("Ranswer", right_answer);
        intent.putExtra("Difficulty", "EASY"); //=> ou autre difficultés
        startActivity(intent);
    }
```

4 Quelques points délicats/intéressants

Comme nous l'avons précédemment cité, nous avons fait face lors du développement à plusieurs difficultés et problèmes. En effet, trois parties se sont avérées un peu plus compliquées que prévu : la construction du slider d'images avec liaison entre les données, la gestion du service de son et la persistance des données. Chacun de ces points a nécessité d'intenses recherches et de très nombreuses heures de debuggage et de tests.

4.1 Slider d'images et liaison entre les données

Pour construire le diaporama, nous avons créé un objet SliderItem qui représente en fait un simple entier. L'intérêt de l'approche objet ici est d'avoir accès à un getter qui plus tard nous donnera plus facilement accès aux données. Ensuite pour faire le lien entre l'activité et les données, il nous a fallu créer une classe SliderAdapter héritant de RecyclerView.Adapter<SliderAdapter.SliderViewHolder>.

Les difficultés survinrent lors de la liaison entre les images du slider et les textviews proposés à l'utilisateur. N'étant pas du même type il nous a été impossible de les lier entre eux comme l'exigeait notre approche. Afin de palier à cela, nous sommes passés par la création de listes jumelles telles que : les listes List_affiche et List_affiche_image. L'une contenant les identifiants des strings provenant du fichier strings.xml qui seront utilisés pour la création des réponses et l'autre contenant les identifiants des drawables pour l'affichage des images. La création d'une liste en lien avec le strings.xml nous permet ainsi de faire la vérification entre les valeurs des Textviews et celles de la liste réponses et ce, en dépit de la langue choisie.

Grâce à cette "liaison", nous avons pu construire notre slider d'images à partir des drawables et du fichier string.xml, facilitant les méthodes de vérification des réponses .

4.2 Service de son

Dans le but de rendre le jeu plus immersif, nous avons décidé d'inclure dans le cahier des charges de l'application, un service de fond sonore. Pour ce faire, nous avons tout d'abord créé une classe et activé les services pour celle-ci dans le AndroidManifest.xml avec les lignes suivantes :

```
<service
android:name="com.example.memoris.Music_Background"
android:enabled="true" />
```

La classe Music_Background hérite de "Service" et implémente "MediaPlayer.OnErrorListener". Cette interface nous permet de paramétrer le lecteur de musique et de contrôler plus précisément son comportement dans diverses situations.

À noter que cette classe comporte une "sous-classe ServiceBinder" qui hérite de "Binder" pour permettre de "lier" ou "délier" le service de musique

Une fois cette classe réalisée, nous avons créé une nouvelle classe, HomeWatcher, qui va nous permettre de gérer le comportement du service lors d'un appui sur les boutons éteindre ou accueil du téléphone. Enfin dans chaque activité de l'application, nous préparons et contrôlons le service de musique via les méthodes de bases et celles d'HomeWatcher.

Exemple avec la classe Credits

```
public class Credits extends AppCompatActivity {
    //GESTION MUSIC
    HomeWatcher mHomeWatcher;
    private boolean mIsBound = false;
    private Music_Background mServ;
    private ServiceConnection Scon = new ServiceConnection() {
        public void onServiceConnected(ComponentName name, IBinder binder) {
            mServ = ((Music_Background.ServiceBinder)binder).getService();
        }

        public void onServiceDisconnected(ComponentName name) {
            mServ = null;
        }
    };
    void doUnbindService() {
        if(mIsBound) {
            unbindService(Scon);
            mIsBound = false;
        }
    }
    ////////////////////////////////////////////////////////////////
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.credits);
        mHomeWatcher = new HomeWatcher(this);
        mHomeWatcher.setOnHomePressedListener(new HomeWatcher.OnHomePressedListener() {
            @Override
            public void onHomePressed() {
                if (mServ != null) {
                    mServ.pauseMusic();
                }
            }
            @Override
            public void onHomeLongPressed() {
                if (mServ != null) {
                    mServ.pauseMusic();
                }
            }
        });
        mHomeWatcher.startWatch();
    }
    @Override
    protected void onResume() {
        super.onResume();
        if (mServ != null) {
            mServ.resumeMusic();
        }
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
```

```
        doUnbindService();
        Intent music = new Intent();
        music.setClass(this, Music_Background.class);
        stopService(music);
    }
    @Override
    protected void onPause() {
        super.onPause();
        PowerManager pm = (PowerManager)
                getSystemService(Context.POWER_SERVICE);
        boolean isScreenOn = false;
        if (pm != null) {
            isScreenOn = pm.isScreenOn();
        }

        if (!isScreenOn) {
            if (mServ != null) {
                mServ.pauseMusic();
            }
        }
    }
    public void onBackPressed() {
        Intent gameActivity = new Intent(Credits.this, MainActivity.class);
        startActivity(gameActivity);
    }
}
```

Seule l'activité Main_ Activity possède la méthode doBindService() qui permet de lier le service

4.3 Persistance des données

Afin de garder en mémoire les scores du joueur, nous avons créé une base de données qui enregistre dans une table : l'id, la difficulté et le nombre de bonnes réponses de chaque essai. C'est la classe MyDbAdpater qui a pour rôle d'implémenter le fonctionnement de la base. Plusieurs méthodes telles que insert_score() ou countEvent() permettent d'agir sur la base de données avec plus de facilité.

Grâce à un bouton sur la page d'accueil le joueur a la possibilité d'effacer les données précédemment enregistrées.

Pour simplifier la transmission des données, nous avons créé un objet Score qui va récupérer et stocker les différentes composantes de la table. Cet objet sera ensuite affiché sous forme de ligne composée de cellules (*cell_ score.xml*) dans le layout leaderboard.xml. *(L'affichage des données de la table sous forme de liste requiert la création d'une classe adapteur, ici MyArrayAdapter.)*

4.4 Problèmes connus et solutions possibles

Lors du développement certains problèmes mineurs se sont présentés. Par souci de temps nous avons choisi de mettre de coté ces problèmes pour nous focaliser sur un rendu fonctionnel. Parmi ces problèmes, il y a notamment : un crash de l'application (*white screen)*, la perte de données lors de la rotation de l'écran pendant le minuteur d'avant partie et sur l'écran de réponses, ainsi qu'un dysfonctionnement du service sonore lorsque l'utilisateur décide de couper le service, changer d'activité et de revenir sur la page d'accueil pour relancer le service.

Pour le premier "bug", nous avons pensé qu'il serait réparable grâce à la surcharge de la méthode onSaveInstanceState(). Cependant, cela pourrait s'avérer complexe car les données ne sont pas stockées dans des variables mais sont directement générées par des boucles, ce qui rendrait l'usage de outState.put[Type] difficile et causerait probablement des incompatibilités de types. *(L'application fonctionne toutefois correctement si l'utilisateur ne change pas de rotation pendant le déroulement de la partie.)*

Enfin pour le deuxième "bug", nous pensons qu'il s'agit d'un problème de ré-initialisation de la valeur du champ VISIBILITY du bouton Son On/Off. Empêchant ainsi l'utilisateur dans certaines conditions d'effectuer l'arrêt ou le redémarrage du service (*Possibilité de solution avec un flag*).

5 Conclusion

Ce projet fut très formateur pour nous, il nous a fait découvrir le développement d'application et les différentes facettes du support mobile.

Il nous a aussi poussé dans nos retranchements en nous forçant à trouver des solutions très précises aux problèmes auxquels nous étions confrontés. Malgré certains "bugs" mis de côté par souci de temps, nous sommes satisfaits du rendu car le résultat correspond à l'idée que nous avions couché sur le papier quelques mois auparavant.

Nous espérons que vous prendrez plaisir à jouer à Memoris.