Développement pour mobiles 2
MASTERMIND : Unlimited Edition

GASTELLIER Alison et BOYER Luis
L3 Informatique

2018

Résumé

Ce document est un rapport concernant le projet donné au cours de l'Unité d'enseignement Développement pour mobiles 2. Ce projet avait pour objectif la création du même jeu sous iOs et Android avec un certain nombres de contraintes. On y décrira donc toutes les étapes importantes de sa conception.

1 Introduction

D'après les dernières études effectuées sur les parts du marché concernant les systèmes d'exploitations (OS) utilisés par les mobiles, Android et iOS occupent 99.6% du marché. La quasi totalité des mobiles utilisant donc ces OS, il est important pour un développeur de pouvoir créer des applications utilisables par ces deux derniers. L'Unité d'enseignement Développement pour mobiles 2 a été pour nous l'occasion de les rencontrer et de pouvoir nous essayer au développement d'une application qui devrait fonctionner sur Androit et iOS.

L'objectif était de créer un jeu comportant les contraintes suivantes : un score doit être associé à chaque partie, possibilité de sauvegarder une partie et son score, fonctionne sur tout type d'écran en mode portrait et paysage.

Le jeu que nous avons décidé de créer est une reprise du célèbre jeu MasterMind. Nous allons par la suite commencer par vous présenter les principes de notre jeu. Puis nous décrirons l'interface sous les deux plateformes. Après, nous soulignerons quelques particularités du code. Enfin, nous parlerons du système de sauvegarde que nous avons mis en place. Nous conclurons à la fin sur le résultat obtenu et sur les améliorations possibles de l'application.

2 Principes

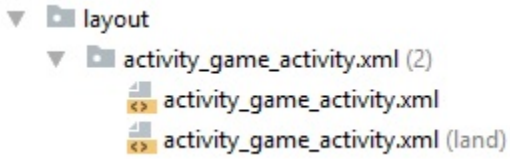
MASTERMIND : Unlimited Edition reprend les principes du jeu MasterMind, avec une légère variante qui est l'ajout d'un score associé à chaque partie afin de respecter les contraintes imposées. Le joueur devra donc essayer de deviner une combinaison de quatre couleurs choisies aléatoirement parmi six couleurs différentes. Pour l'aider, deux couleurs supplémentaires apparaîtront à côté de chacun de ses essais pour lui indiquer le nombre de bonnes couleurs et si ces derniers sont placées ou non au bon endroit. Le but du joueur est donc d'essayer à chaque partie de battre ses précédents scores.

3 Interface

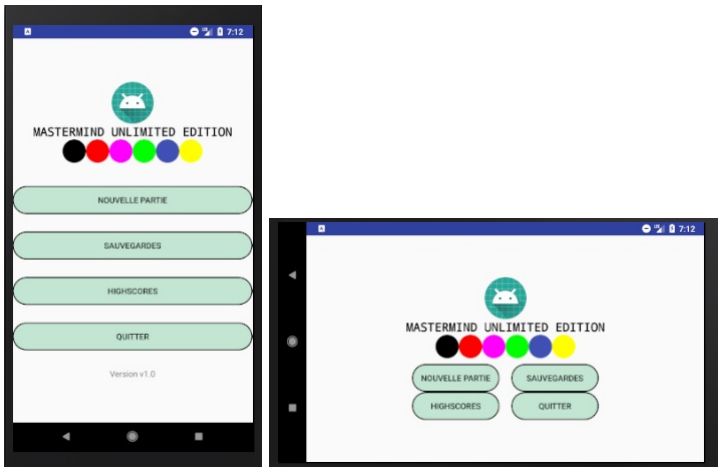
L'idée était d'offrir à l'utilisateur suffisamment d'essais pour rendre le jeu accessible tout en gardant une interface claire. Nous avons décidé que l'interface finale comporterait donc :
- 12 lignes pour 12 essais pour l'utilisateur,
- un bouton " VALIDER " pour valider son essai et la comparer à la solution,
- un bouton "? " pour rappeler à l'utilisateur les règles du jeu,
- un bouton " QUITTER " afin de pouvoir quitter la partie en cours,
- un label " SCORE " pour rappeler à l'utilisateur son nombre de points,
- un bouton " SOLUTION " afin de laisser à l'utilisateur la possibilité d'abandonner et de voir la combinaison gagnante avant de passer sur l'écran suivant.

3.1 Android

Sous Android, il est très facile de gérer les tailles et orientations d'écran, il suffit de créer un *Layout* réservé soit à la taille de l'écran, soit sa position.



On obtient les résultats suivants :



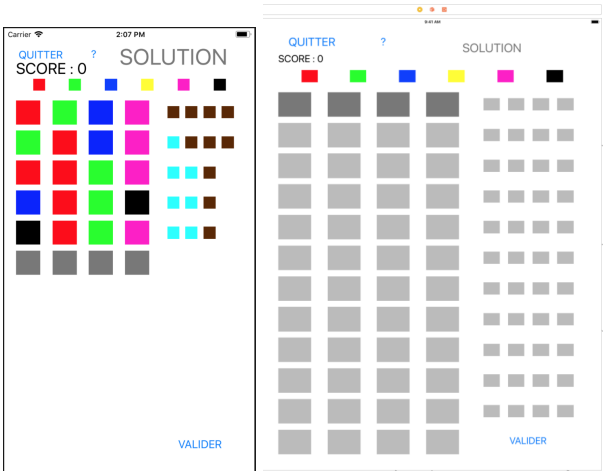
Notre application étant assez simple au niveau graphique, nous n'avons pas eu besoin de traiter chaque taille d'écran disponibles. La gestion des conteneurs *Layout* sont suffisants pour bien présenter sous tout téléphone.

3.2 iOS

Sous iOS, il ne nous était pas offert la possibilité de pouvoir gérer la partie portrait et paysage séparément et en fonction de tous les écrans. Il fallait donc apporter des contraintes graphiques sur chaque élément (boutons, textes, ...) afin qu'ils gardent une certaine proportion par rapport à l'appareil sur lequel l'application était lancée. Effectivement, ayant pour objectif de pouvoir jouer avec des cercles afin de rester le plus proche possible du jeu de société, il fallait sous iOS utiliser des carrés pour pouvoir ensuite transformer ces derniers en cercle. Cependant, nous nous sommes rendus compte plus tard qu'il n'était pas possible de garder des carrés en mode paysage et portrait et ce sur tous les appareils en même temps. Nous avons donc opté pour une solution offerte par

l'Interface Builder de Xcode dans le Size Inspector : Autoresizing. En cochant les flèches comme on peut le voir ci-dessous, l'élément graphique concerné gardera automatiquement une certaine proportion sembable sur tous les écrans par rapport à la proportion qu'a cet élément sur l'écran actuel. Nous pouvions alors obtenir des carrés plus ou moins parfaits.

Une fois ces carrés obtenus nous nous attaquions donc à la transformation de ces carrés en cercle en utilisant la propriété *cornerRadius* des boutons. Afin de faire d'un carré un cercle il faut donner comme valeur à cette propriété la moitié de la valeur de la largeur du carré concerné. Malheureusement, au travers des différents test effectués nous avons pu observer qu'en dessous d'une certaine taille, on obtenait un oval et non un cercle. Pour garder une certaine homogénéité nous avons donc gardé des carrés sous l'interface iOS. Voilà ce à quoi elle ressemble :



A gauche on peut observer ce qu'aura l'utilisateur. (iPhone 8 Plus)
A droite on peut observer ce que contient réellement l'application. (iPad Pro 12.9")

4 Algorithmes

Le jeu comporte peu de fonctions complexes. Nous en avons retenues deux : la validation d'un essai et l'aide apportée à l'utilisateur à la suite de son essai. Nous présenteront ici les événements composants la vérification d'un essai et les conséquences d'une bonne ou mauvaise réponse.

4.1 Validation

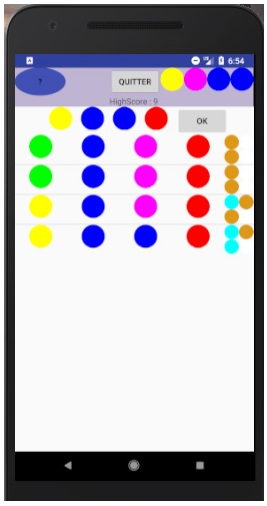
Les couleurs associées à l'essai sont comparées avec celles qui composent la solution à la même place. Si l'essai et la combinaison à découvrir correspondent alors l'utilisateur obtiendra un nombre de points égal à 12 - (le nombre d'essais - 1). L'interface sera alors réinitialisée, le score sera mis à jour et une nouvelle combinaison sera générée.

Remarque : L'utilisateur ne pourra pas valider une combinaison si elle ne comporte pas quatre couleurs.

4.2 Bien placées et mal placées

Afin de pouvoir aider le joueur dans le jeu voici comment est vérifié si l'emplacement d'une couleur est le bon si la couleur appartient à la solution. Dans notre jeu, le nombre de bonnes couleurs au bon endroit sera signalé par le même nombre de carrés de couleur cyan. De même pour les bonnes couleurs au mauvais endroit avec une couleur marron. Comme ci-dessous :

Remarque : La solution du jeu est affichée pour bien montrer que l'algorithme est juste.



On définit ces valeurs en deux étapes. Premièrement on compare la couleur de chaque même emplacement entre l'essai et la solution. A chaque fois qu'on trouve une correspondance alors on augmente la variable qui contient le nombre de couleurs au bon endroit de 1. On marque alors les deux emplacement utilisés à l'aide de deux listes. Puis on compare chaque couleur à toutes les autres couleurs. Si deux couleurs sont égales, on vérifie si ces deux couleurs n'ont pas déjà été utilisées. Si elles ne le sont pas alros on augmente la variable qui contient le nombre de couleurs au mauvais endroit de 1 puis on marque ces deux emplacements.

Soit le code commenté sous Android :

```
public void check(int[] tab, int x) {
    boolean win = false;
    int indexx = x;
    nbbienplace = 0;
    nbmalplace = 0;
    int[] verifa = new int[4]; //tableau servant à marquer les couleurs déjà testées ou non pour
    int[] verifb = new int[4]; //pour la combinaison proposée

    //on test si on trouve les bonnes couleurs aux bons endroits
    for (int i = 0; i < 4; i++) {
        if (combi_soluce[i] == combinaison[i]) {
            nbbienplace++;
            indexx += 1;
            tab[indexx] = Color.CYAN;
            //on coche pour montrer que ces cercles sont déjà comparés et valides
            verifa[i] = 1;
            verifb[i] = 1;}
    }


    //on compare les cercles restants entre eux pour vérifier ceux qui sont mal placés
```

```
        for (int i = 0; i <= 3; i++) {
            for (int j = 0; j <= 3; j++) {
                //S'ils sont de la même couleurs et non marqués ils sont mal placés
                if ((combi_soluce[i] == combinaison[j]) && verifa[i] != 1 && verifb[j] != 1) {
                    nbmalplace++;
                    indexx += 1;
                    tab[indexx] = Color.rgb(223, 152, 20);
                    verifa[i] = 1;
                    verifb[j] = 1;}
            }
        }

        //Si tout est bien placé, le joueur a gagné
        if (nbbienplace == 4) {
            win();
            win = true;
        }

        //si le joueur dépasse 12 lignes, il a perdu
        if (nbrow >= 12 && !win) {
            lose();
        }
    }
```

5 Sauvegarde

Une des contraintes que comportait ce projet était la possibilité de pouvoir quitter une partie, de la sauvegarder puis de la reprendre ou la supprimer. La liste des parties sauvegardées et la liste des scores devaient être présentées sous la forme de liste.

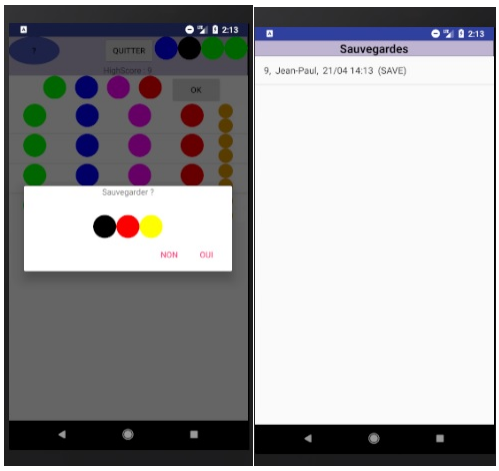
5.1 Android

Android possède plusieurs manières pour enregistrer les données de manière persistante. Nous avons choisi d'utiliser les SharedPreferences pour stocker toutes les informations nécessaires. Les ShraredPreference servent à stocker des informations grâce à un couple (valeur/clé). Les valeurs sont principalement de type simple (String, entier, booléen). Le seul type complexe géré dans les dernières versions d'Android sont les set. Ils permettent de stocker une liste de Strings.

```
SharedPreferences settings1 = getSharedPreferences("data", 0); //Créer/récupérer les sharedpreferenc
Set<String> set2 = settings1.getStringSet("data", null); //Récupérer les données
```

Si le sharedPreference du mot clé data est null, alors set2 sera null.

Pour ce qui est de la présentation, nous avons choisi de faire confirmer la sauvegarde à l'utilisateur qui sera ensuite automatiquement redirigé vers l'écran des sauvegardes.



5.2 iOS

Sous iOS, le système de sauvegarde mis en place utilise *UserDefaults.standard*. Nous avons donc déclaré en variable globale une variable *userData* que nous utiliserons pour stocker et extraire les différentes valeurs nécessaires à la partie de l'utilisateur telles que son score ou son nom. La déclaration et l'utilisation de cette variable se fait comme suit :

```
let userData = UserDefaults.standard // Declaration
userData.set(0 , forKey: "s") // On stocke la valeur 0 avec comme clef "s"
s = userData.integer(forKey: "s") // On extrait la valeur avec comme clef "s"
```

La sauvegarde n'est pas automatique. Elle se déclenche lorsque l'utilisateur décide de quitter l'application en utilisant le bouton " QUITTER " et presse " OUI " à la demande d'enregistrement. Il lui sera alors proposé d'entrer son nom. L'utilisateur est obligé d'entrer un nom pour enregistrer sa partie. Il pourra alors choisir l'un des trois emplacements disponibles.

Lors de chaque sauvegarde on actualise la liste des trois scores les plus hauts atteints. Même si le joueur perd, son score peut entrer dans ce classement si il est supérieur au score le plus bas du classement.

Lorsque l'utilisateur perd une partie, il lui sera aussi proposer d'enregistrer sa partie. Il pourra donc créer sa propre sauvegarde mais recommencera avec un score égal à 0.

Remarque : A chaque sauvegarde est associé le nom du joueur, son score ainsi que la date à laquelle la sauvegarde a été effectuée. La liste des scores ne contiennent eux que le nom et le score du joueur.

Afin de savoir à qui appartient la sauvegarde on utilise une variable globale *MonIndex* qui contient le numéro de la ligne sur laquelle l'utilisateur a appuyé lors du choix de la sauvegarde. On vérifie donc toujours ce qu'elle contient afin de pouvoir sauvegarder correctement.

La sauvegarde est donc programmé de cette manière :

```
if(ChampTexte.text != ""){
            if ( monIndex == 0 ) {
                userData.set(ChampTexte.text,forKey: "n1")
                userData.set(dateDuJour(),forKey: "t1")
                userData.set(s,forKey: "s1")
                if(userData.integer(forKey: "s1")>=classement[2]){
                    classement[2] = userData.integer(forKey: "s1")
                    classement_nom[2] = userData.string(forKey: "n1")
                }
```

```
                if(perdu){ userData.set(0,forKey: "s1") }
            }
            ...
}
```

Remarque : Les variables *classement* et *classement_nom* servent pour la sauvegarde des meilleurs scores (nom et score du joueur)

Pour ce qui est de la suppression de la sauvegarde, un choix est proposé au joueur quand il utilise le menu de sauvegarde. Il lui est demandé si il tient à chargé sa sauvegarde ou si il souhaite la supprimer. La suppression d'une sauvegarde remet à zéro les valeurs des variables qui lui sont associées de la manière suivante :

```
if(myIndex == 0){
userData.set(0,forKey:"s1")
userData.set(nil,forKey:"n1")
userData.set("",forKey:"t1")
}
```

5.3 Problèmes rencontrés

Dans notre cas, nous n'avons pas pu mettre en place un système 100% fonctionnel de la sauvegarde sur les deux plateformes. En effet, dans le cas d'Android, il était assez difficile de sauvegarder l'intégralité des choix proposés par l'utilisateur ainsi que les indices de ces lignes dans un seul set. Il en était de même pour iOS qui demandait un nombre de ligne de code assez conséquentes sans possibilité de raccourcir. Nous avons donc décidé de ne sauvegarder que les scores des parties choisies.

6 Conclusion

Globalement, nous avons pu respecter les règles imposées dans le CCTP, le jeu est fonctionnel, il y a un système de score complet avec classement ainsi que la possibilité de les supprimer. Seul le système de sauvegarde n'est pas optimal, mais nous avons pu arriver à un résultat fonctionnel sous forme de liste avec possibilité de suppression. Avec cet exercice complet, nous avons pu nous rendre compte de la force et des faiblesses des logiciels comme Android Studio et Xcode et de l'étendu des possibilités que l'on avait à portée de main pour développer sous téléphone. Notre jeu a été une bonne expérience que l'on pourra toujours améliorer au fil du temps.

7 Bibliographie

Références

[1] Android developers. https://developer.android.com.

[2] Android sharedprefrences. www.android-dev.fr/utilisation_des_shared_preferences.

[3] iOS developer. https://developer.apple.com.

[4] Sébastien PEROCHON Sylvain HEBUTERNE. *Android Guide de développement d'applications pour Smartphones et Tablettes 2ème édition*. Expert IT, 2014.