Département Informatique

Documentation Technique

Zo RABARIJAONA
Jérémy MOROSI
Willy FRANÇOIS
Raya DJADLI

Année : 2013-2014

Table des matières

1 Analyse

1.1 Des applications difficilement modifiables

Castle Defense, librarie, code natif, attaque héxa, rétro engeneering...

1.2 L'Application Checkers

1.2.1 Principe

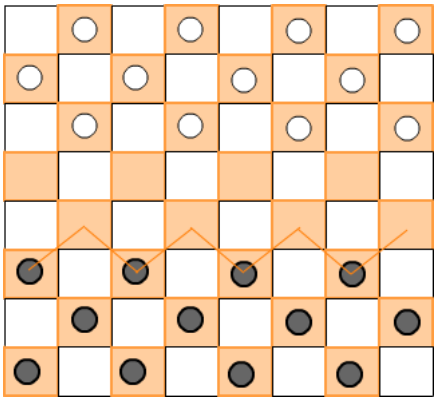


FIGURE 1: Principe

Le jeu est composé de 8 cases sur 8, c'est-à-dire 4 cases en moins que le jeu de dame classique. Se joue en deux modes : utilisateur contre ordinateur ou utilisateur contre utilisateur.
Un joueur peut choisir une couleur (des pions/dames noires ou des pions/dames blanches). Le jeu est initialement constitué de 3 lignes de 4 pions espacés d'une case. Chaque pion/dame peut se déplacer en diagonale. Une dame ou un pion peut faire une prise en faisant un déplacement en diagonal. Le jeu oblige à faire une prise (Must capture) si l'utilisateur n'a pas activé l'option capture optionnelle. Un pion se déplace uniquement en avant et se transforme en une dame (K) lorsqu'il arrive à la dernière ligne du camp de l'adversaire. Une dame peut effectuer une prise en faisant un déplacement en arrière ou en avant.
Un joueur est désigné gagnant si l'adversaire ne possède plus de dames ou de pions.

1.2.2 Décompilation et analyse du code

Afin de pouvoir analyser le code de l'application plus aisément, nous avons dû trouver un moyen de décompiler l'application vers du code java. Pour ce faire, nous avons utilisé *dex2jar* [1], une api permettant de décompiler un fichier apk en un fichier jar contenant des fichiers class. Ensuite, nous avons dû utiliser *jd-gui* [2] sur ce fichier jar pour en afficher le code source. L'application permet d'exporter les sources ainsi décompilées.

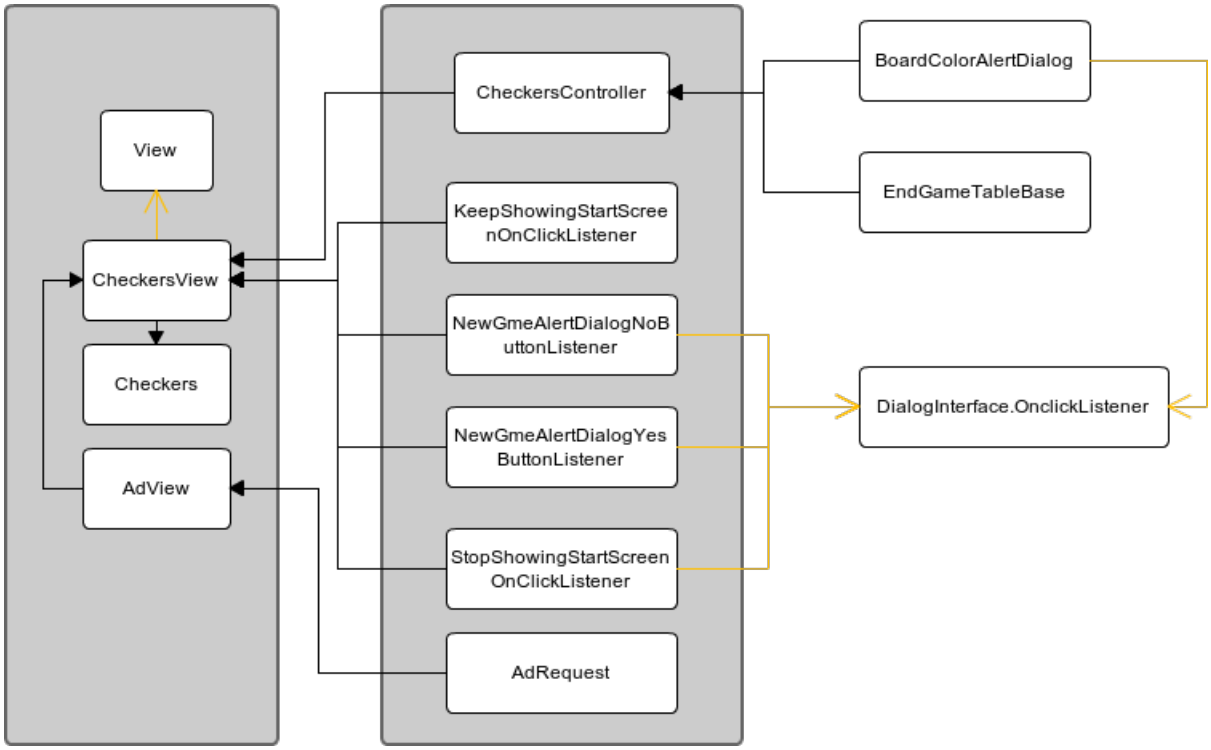
Une fois le code java de l'application en notre possession, nous avons pu commencer à l'analyser. Nous avons vite remarqué que le code avait été obscurcit avant d'être compilé

par le développeur de l'application. Cela signifie que toutes variables et fonctions avaient un nom sans aucun sens, ce qui gêna la compréhension du code. Nous avons aussi remarqué que la décompilation c'était mal passée à cause de return et de break à des endroits incongrus. Nous avons donc commencé par supprimer ces erreurs afin de pouvoir avoir un code "compilable" sous Eclipse. Ceci nous a offert un accès à l'outil de refactorisation de code pour renommer les variables après avoir compris à quoi elles pouvaient servir.

Malgré cela, à cause du problème de décompilation, certaines méthodes n'avaient aucun sens. Nous avons donc décidé de relire le code smali des méthodes correspondantes afin de le traduire manuellement en java et ainsi avoir le code original exact.

1.2.3 Architecture

Dans un point de vue général, le développeur du jeu Checkers a adopté une architecture MVC. Ce qui nous a beaucoup aidé à determiner les différentes classes.
La classe Checkers est la classe principale de l'application. La classe CherckersView



architecture

(b.smali) hérite de la classe android.view.View qui gére les vues de l'application. La classe AdView gère la vue pour la publicité. CheckersController (a.smali) est le controleur principal de l'application. Les classes KeepShowingStartScreenOnclickListener (e.smali), NewGameAlertDialogNoButtonListener (d.smali), NewGameAlertDialogYesButtonListener (c.smali) et StopShowingStartScreenOnclickListener (f.smali) sont les classes qui gèrent les clics. Elles implémentent l'interface DialogInterface.OnclickListener. AdRequest est la classe qui est responsable du chargement de la publicité.

2 Modifications apportées à l'Application

2.1 Suppression de la publicité

Nous avons remarqué que l'application ne possèdait qu'un seule point d'entrée pour afficher la publicité. L'application charge la vue pour la publicité dans la méthode onCreate().

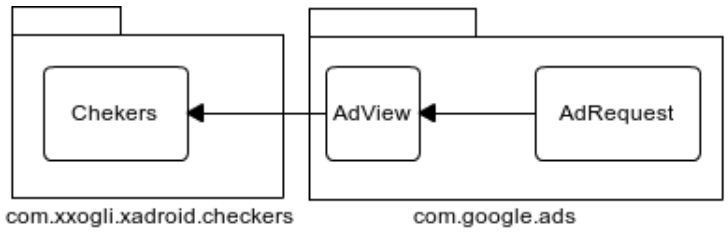


FIGURE 2: Affichage de la publicité

```
const/16 v1, 0x50
invoke-virtual v0, v1, Lcom/google/ads/AdView;->setGravity(I)V
new-instance v1, Lcom/google/ads/AdRequest;
invoke-direct/range v1 .. v1, Lcom/google/ads/AdRequest;-><init>()V
invoke-virtual v0, v1, Lcom/google/ads/AdView;->loadAd(Lcom/google/ads/AdRequest;)V
```

TABLE 1: code smali pour la publicité

Pour supprimer la publicité, il a suffit de supprimer ces quelques lignes. La variable v1 contient l'instance de la classe *com.google.ads.AdRequest* qui est le point d'entrée de la publicité dans l'application. La variable v0 est la vue *com.google.ads.AdView* associée à la publicité. Le chargement de la publicité se fait par la méthode *loadAd(AdRequest adrequest)*.

2.2 Ajout d'un bouton Exit

Beaucoup d'applications n'ayant pas de bouton "Quitter" explicite et ne proposant parfois même pas de quitter (sans faire le bouton Home), nous avons voulu en ajouter un dans le menu de l'application.

Pour cela, il a fallu modifier les méthodes onCreateOptionsMenu et onOptionsItemSelected de la classe *Checkers* afin de pouvoir ajouter le bouton et son action.

Lors de l'appui sur le bouton, un appel à la méthode finish() est exécuté. Ainsi, les méthodes onPause (sauvegardant les données) et onStop (exécutant un System.exit()) sont appelées.

```
    const/4 v9, 0x7
...
    const-string v0, "Undo"
    invoke-interface {p1, v5, v4, v4, v0},
        Landroid/view/Menu;->add(IIILjava/lang/CharSequence;)
        Landroid/view/MenuItem;
    const-string v0, "Exit"
    invoke-interface {p1, v5, v9, v6, v0},
        Landroid/view/Menu;->add(IIILjava/lang/CharSequence;)
        Landroid/view/MenuItem;
    const-string v0, "Switch Side"
```

FIGURE 3: Ajout du bouton dans onCreateOptionsMenu

```
    const/16 v5, 0x7
...
    :cond_2
    if-ne v1, v3, :cond_42 # On va au test pour le bouton Exit
...
    :cond_42
    if-ne v1, v5, :cond_3 # On retourne au test suivant
    invoke-super {p0}, Landroid/app/Activity;->finish()V
    goto :goto_0
    :cond_3
```

FIGURE 4: Action du bouton dans onOptionsItemSelected

2.3 Commencer avec des Dames

Afin d'être à notre avantage, nous avons voulu faire commencer les noirs, sous-entendu le joueur humain en début de partie, avec uniquement des dames. Pour cela, il a fallu inverser les placements respectifs des pions et dames du joueur noir dans la méthode a.a() (correspondant à initPlateau).

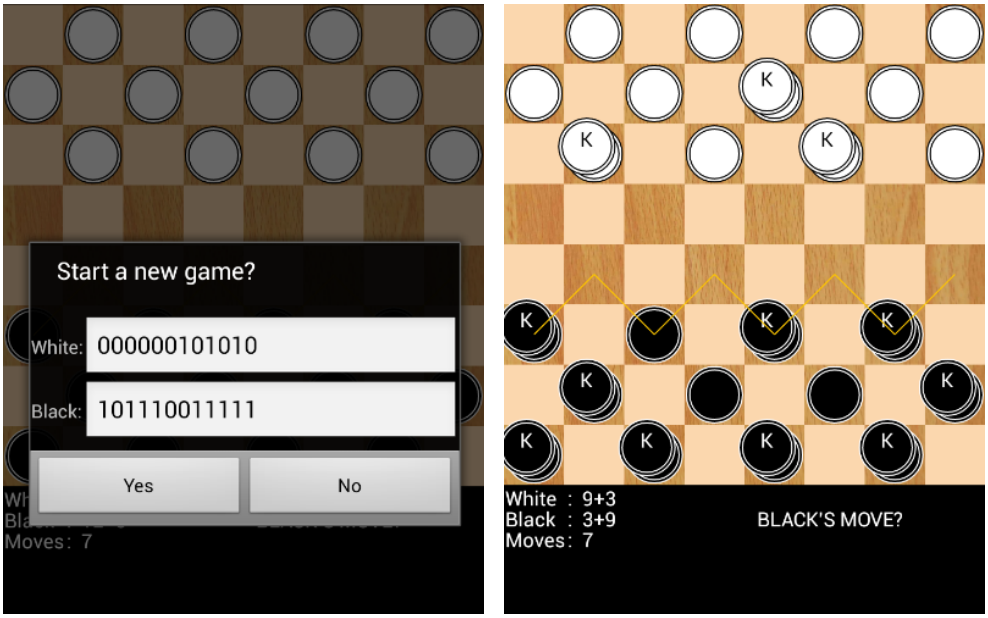
```
    const/high16 v0, -0x10
# Interversion des pions et dames des noirs v1 et v0
    iput v1, p0, Lcom/xxogli/xadroid/checkers/a;->f:I
    iput v0, p0, Lcom/xxogli/xadroid/checkers/a;->g:I
```

FIGURE 5: Interversion des pions et dames des noirs

La variable a.f correspond aux pions et a.g correspond aux dames. v0, que l'on place sur les dames, contient le placement initial des noirs. Pour les pions noirs, on place v1 = 0 qui correspond à aucun pion placé. Ainsi, tous les pions noirs sont des dames.

2.4 Choisir le placement des pions

Après avoir modifié le placement des pions de manière statique dans le code, nous avons voulu aller plus loin en donnant la possibilité au joueur de choisir ce placement. Nous savions déjà quel endroit du code il fallait modifier pour changer les pions à la création d'une nouvelle partie, il fallait juste réussir à modifier la boîte de dialogue qui demande confirmation au joueur lorsque l'on choisit 'New Game' dans le menu. Nous voulions que cette boîte de dialogue contienne deux champs de saisie supplémentaires (un pour les pions blancs et un pour les noirs) dans lesquels l'utilisateur puisse saisir une suite de douze '0' et/ou '1', et qu'en cliquant sur 'Yes' pour lancer la nouvelle partie, le plateau soit initialisé avec des pions normaux là où les champs contiennent des '0', et des dames là où ils contiennent des '1'.



(a) Boîte de dialogue (b) Initialisation du plateau

FIGURE 6: Placement des pions

Pour ce faire, l'idée était de faire un vrai projet Android pour pouvoir créer et tester facilement une boîte de dialogue, puis d'exporter le projet en `.apk`, décompiler cet `.apk`, récupérer le code de la boîte de dialogue dans le fichier `.smali` correspondant, et le mettre à la place du code de la vraie boîte de dialogue du jeu.

Nous avons donc fait un projet Android contenant les classes `b` (listing 1) et `a` (listing 2), ainsi qu'une classe `MainActivity` présente uniquement pour pouvoir lancer l'application et tester la boîte de dialogue. La classe `b` correspond à la classe `CheckersView` qui est la vue du jeu, et la classe `a` correspond à la classe `CheckersController` qui est le contrôleur. À noter que certaines lignes de la classe `b` ont été retirées (comme l'affectation du layout pour les éléments de la boîte de dialogue ou le constructeur) afin de la simplifier.

On peut remarquer plusieurs choses. Tout d'abord, nous avons gardé les noms des packages, classes, variables ainsi que les signatures des fonctions tels qu'ils apparaissent dans les fichiers `.smali` du jeu pour pouvoir injecter notre code sans trop de problèmes. Par contre, pour les fonctions et variables supplémentaires qui ne sont utilisées que par notre code, nous avons mis des vrais noms puisque ça ne pose pas de problème.

Ensuite, la méthode `a` de la classe `b` ne fait que retourner `false`. Cette méthode est uniquement présente pour pouvoir compiler le code puisque l'on doit y faire appel quand l'utilisateur confirme la nouvelle partie. Quant à la méthode `a` de la classe `a` qui doit initialiser le plateau, elle ne contient que le code à compiler et à injecter dans la méthode originale.

Pour entrer plus en détail dans le code, et plus précisément la classe `b`, on a les quatres variables `newGame(White|Black)(Pieces|Kings)Placement` (ligne 9) qui contiennent les positions des pions normaux et des dames pour les deux joueurs. On peut voir que, par défaut, la variable `newGameWhitePiecesPlacement` contient un entier avec les douzes premiers bits à '1', et que la variable `newGameBlackKingsPlacement` contient un entier avec les douzes derniers bits '1'.

Le jeu utilise des entiers pour représenter le plateau. Les douzes premiers bits correspondent aux trois premières rangées, les douzes derniers aux trois dernières, le premier bit est la case tout en haut à gauche, et le dernier est la case tout en bas à droite. C'est pour ça que l'on fait des opérations étranges (ligne 50 et lignes 64 à 69) sur la saisie de l'utilisateur.

Pour ce qui est de la classe `a`, on assigne simplement les valeurs de nos variables aux vraies positions des pions avant l'initialisation du plateau (lignes 14 à 17).

Une fois notre `.apk` généré et le code de notre application Android décompilé, on peut très simplement injecter notre code dans les `.smali` du jeu.

```
1  package com.xxogli.xadroid.checkers;
2
3  // CheckersView
4  public class b extends View {
5
6    private Context a;
7    private a p;    // CheckersController
8
9    public int newGameWhitePiecesPlacement = 4095;
10   public int newGameWhiteKingsPlacement = 0;
11   public int newGameBlackPiecesPlacement = 0;
12   public int newGameBlackKingsPlacement = -1048576;
13
14   private TextView textView(String s) {
15     TextView tv = new TextView(a);
16     tv.setText(s);
```

```
17      return tv;
    }
19
    private EditText editText(String s) {
21      EditText et = new EditText(a);
        et.setText(s);
23      return et;
    }
25
    private TableRow tableRow(String s, EditText et) {
27      TableRow tr = new TableRow(a);
        tr.addView(textView(s));
29      tr.addView(et);
        return tr;
31  }

33  private TableLayout tableLayout(TableRow ...rows) {
        TableLayout tl = new TableLayout(a);
35      for(TableRow tr : rows) {
          tl.addView(tr);
37      }
        return tl;
39  }

41  private LinearLayout linearLayout(View ...views) {
        LinearLayout ll = new LinearLayout(a);
43      for(View v : views) {
          ll.addView(v);
45      }
        return ll;
47  }

49  private int value(EditText et) {
        return Integer.parseInt(new StringBuilder(et.getText().
           toString()).reverse().toString(), 2);
51  }

53  // newGameDialog
    public void f() {
55      AlertDialog.Builder b = new AlertDialog.Builder(a);
        final EditText et1 = editText("000000000000");
57      final EditText et2 = editText("000000000000");
        b.setView(tableLayout(
59        tableRow("White:", et1), tableRow("Black:", et2)
        )).setMessage("Start a new game?")
```

```
61      .setCancelable(false)
        .setPositiveButton("Yes", new OnClickListener() {
63        public void onClick(DialogInterface dialog, int which)
              {
            int i = value(et1);
65          newGameWhitePiecesPlacement = ~i & 0xFFF;
            newGameWhiteKingsPlacement = i & 0xFFF;
67          i = value(et2) << 20;
            newGameBlackPiecesPlacement = ~i & 0xFFF00000;
69          newGameBlackKingsPlacement = i & 0xFFF00000;
            a(false, -1, 0, 0, 0);
71          postInvalidate();
          }
73      })
        .setNegativeButton("No", new OnClickListener() {
75        public void onClick(DialogInterface dialog, int which)
              {
          }
77      }).show();
    }
79
    // gameStatus
81  private final boolean a(boolean b, int m, int v, int d,
        int n) {
      return false;
83  }

85 }
```

Listing 1: b.java (CheckersView.java)

```
1  package com.xxogli.xadroid.checkers;

3  // CheckersController
   public class a {
5
     private b j;     // CheckersView
7    private int d;   // lastWhitePiecesPlacement
     private int e;   // lastWhiteKingsPlacement
9    private int f;   // lastBlackPiecesPlacement
     private int g;   // lastBlackKingsPlacement
11
     // initPlateau
13   public final void a() {
       this.d = j.newGameWhitePiecesPlacement;
```

```
15      this.e = j.newGameWhiteKingsPlacement;
        this.f = j.newGameBlackPiecesPlacement;
17      this.g = j.newGameBlackKingsPlacement;
    }
19
}
```

Listing 2: a.java (CheckersController.java)

2.5 Échanger pions et dames à changement de joueur

Dans cette section, nous avons ajouté une option qui nous permet d'avoir le choix d'échanger des jetons noirs et blancs en pions ou dames.

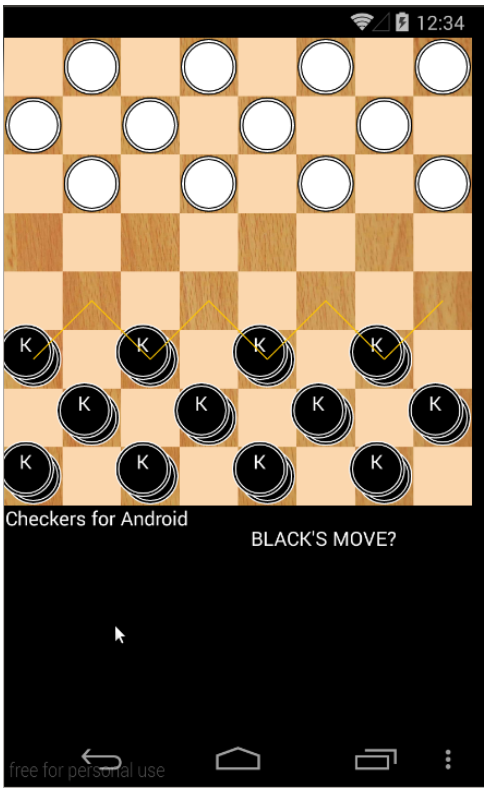


FIGURE 7: Avant modification

Pour celà nous avons d'abord ajouté une option "Switcher les Dames" dans le menu "Options". On a effectué cet ajout dans *Chechers.smali* comme on peut le voir sur la Figure 9.

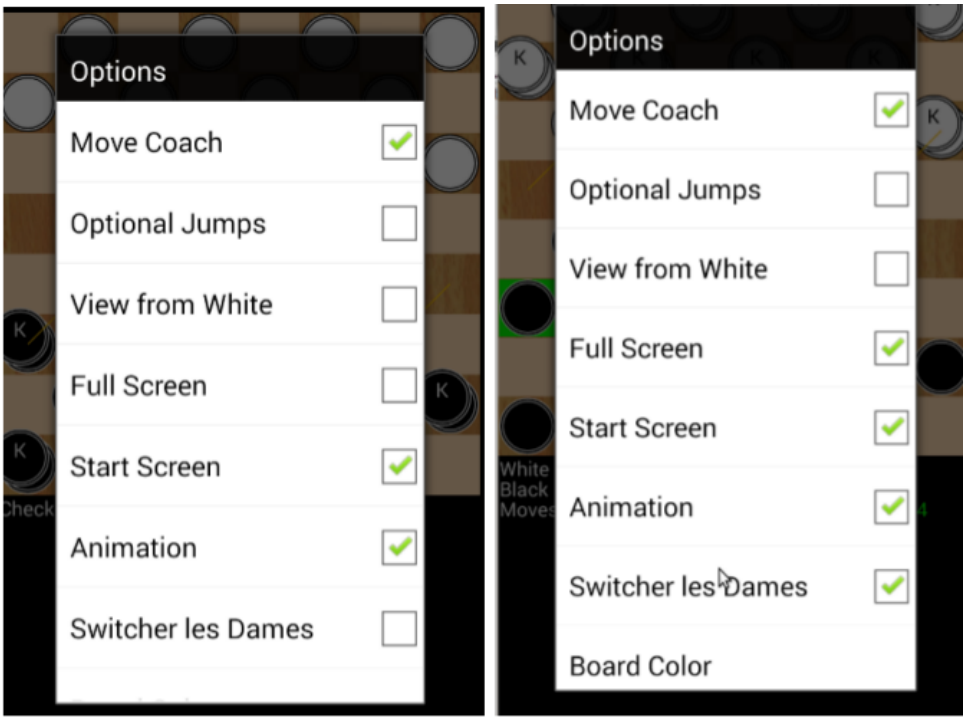


FIGURE 8: ajout d'une option dans le menu

```
const/4 v1, 0x6
    const/4 v2, 0x6
    const-string v3, "Switcher les Dames"
    invoke-interface {v0, v4, v1, v2, v3}, Landroid/view/SubMenu;->
add(IIILjava/lang/CharSequence;)Landroid/view/MenuItem;
    move-result-object v1
    invoke-interface {v1, v4}, Landroid/view/MenuItem;->
setCheckable(Z)Landroid/view/MenuItem;
    move-result-object v1
    iget-object v2, p0, Lcom/xxogli/xadroid/checkers/Checkers;->
a:Lcom/xxogli/xadroid/checkers/b;
    invoke-virtual {v2, v5}, Lcom/xxogli/xadroid/checkers/b;->
permuter(Z)Z
    move-result v2
    invoke-interface {v1, v2}, Landroid/view/MenuItem;->
setChecked(Z)Landroid/view/MenuItem;
```

FIGURE 9: Ajout de l'Option dans onCreateOptionsMenu

Après avoir ajouté l'option "Switcher les Dames" dans le menu des Options, nous insérons la méthode permuter() qui prend un booléen en paramètre et qui renvoie l'état

de la variable **switchdame** qui est un attribut de la "classe b" de type booléen que nous avons ajouté.

```
.method public final  permuter(Z)Z
.locals 1
    if-eqz p1, :cond_0
    iget-boolean v0, p0, Lcom/xxogli/xadroid/checkers/b;->switchdame:Z
    if-eqz v0, :cond_1
    const/4 v0, 0x0
    :goto_0
    iput-boolean v0, p0, Lcom/xxogli/xadroid/checkers/b;->switchdame:Z
    :cond_0
    iget-boolean v0, p0, Lcom/xxogli/xadroid/checkers/b;->switchdame:Z
    return v0
    :cond_1
    const/4 v0, 0x1
    goto :goto_0
.end method
```

FIGURE 10: Ajout de l'action dans onCreateOptionsMenu

Nous avons ensuite ajouté l'action à exécuter lorsqu'on choisit l'option "Switcher les Dames" dans *onOptionsItemSelected*.

```
    :cond_9
    const/4 v2, 0x6
    if-ne v1, v2, :cond_10
    iget-object v1, p0, Lcom/xxogli/xadroid/checkers/Checkers;->
a:Lcom/xxogli/xadroid/checkers/b;
    invoke-virtual {v1, v0}, Lcom/xxogli/xadroid/checkers/b;->
permuter(Z)Z
    move-result v1
    invoke-interface {p1, v1}, Landroid/view/MenuItem;->
setChecked(Z)Landroid/view/MenuItem;
    goto/16 :goto_0
```

FIGURE 11: Ajout de l'action dans le onOptionsItemSelected

Le résultat est affiché comme suit.

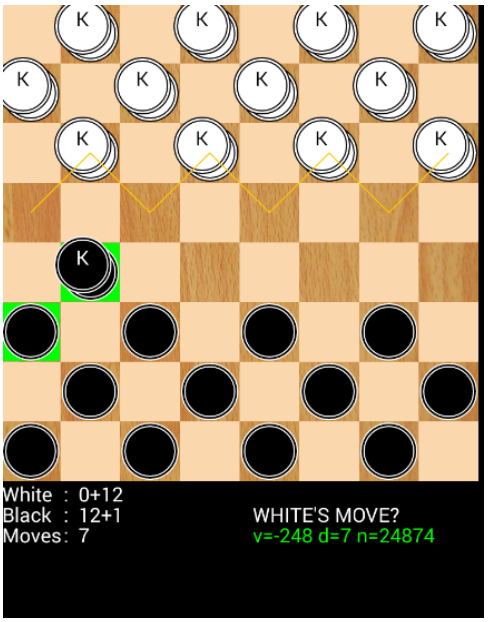


FIGURE 12: avant modification

Remarque : Lors de l'execution, nous avons constaté qu'il y avait un soucis avec la permutation. Ce dernier est dû au au fait que le contrôleur doit prévoir à l'avance le coup qu'il va jouer en fonction de la position des jetons qu'il a avant d'effectuer le switch. Ainsi, on se retrouve avec un jeton supplémentaire de l'ancien type.

Références

[1] Panxiaobo. dex2jar is a tool for converting android's .dex format to java's .class format. https://code.google.com/p/dex2jar/, 2011.

[2] thinkf... xinbiao... Jd-gui is a standalone graphical utility that displays java source codes of .class files. https://code.google.com/p/innlab/, 2010.