UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Travaux Pratiques INFO0306 : Programmation mobile

TP n◦2 – Jeu 2048

Table des matières

1. Objectif

Les objectifs pédagogiques de ce TP sont les suivants :

- Réalisation d'un mini-jeu : le 2048
- Développement spécifique (partie interface / partie moteur)
- Personnalisation du jeu
- Publication du jeu

2. 2048

Ce jeu a été développé pour mobiles en 2014 par un développeur autodidacte italien Gabriele Cirulli.
Son principe de fonctionnement est décrit sur http://fr.wikipedia.org/wiki/2048_(jeu_vidéo).
Vous avez la possibilité de tester le jeu directement en version web : http://jeu2048.fr/

Exercice 1 : Lire et comprendre le principe et le fonctionnement générale du jeu 2048 (version standard 4x4)

3. En route

Il sera judicieux de séparer le développement de l'interface utilisateur du moteur de fonctionnement.
Le TP vous accompagnera tout au long à respecter ce principe afin de pouvoir éventuellement réutiliser le moteur écrit en JAVA.

Ce TP vous permettra d'obtenir une base commune afin de bien mettre en place votre 2048. Cependant, la personnalisation de l'interface utilisateur et ensuite de certaines fonctionnalités sera souhaitée dans la suite du TP (mais y reviendrons en temps voulu).

Commençons par créer le projet sous votre IDE préféré.

3.1. Création du projet

Exercice 2 : En respectant la convention de nommage du TP de prise en main. Créez un projet nommé « My2048 » avec l'icône de votre choix contenant une activité principale. On personnalise déjà !

Exercice 3 : Ajoutez à votre package, une classe nommée « Game2048 ». Elle s'occupera de la partie « moteur ».

4. Mise en page

4.1. Création des zones

Notre interface utilisateur devra comprendre 3 zones distinctes (de haut en bas)
- 1ère zone : le score
- 2ème zone : le plateau de jeu (board)
- 3ème zone : les commandes du jeu

Chaque zone devra être empilé verticalement en respectant une occupation respective de 10%, 60% et 30% (score, plateau, commande), (pas toute suite, patience).

Exercice 4 : Déterminez le bon type de mise en page contenant ces 3 futures zones et mettez la en place. (LinearLayout, RelativeLayout, WebView, etc.).

Indice : Vous pouvez modifier directement le type du Layout avec « Change layout » dans le menu contextuel de la fenêtre de hiérarchie des vues = Eclipse, AStudio à vérifier, sinon modifier à la main).

Exercice 5 : Nommez ce layout : **@+id/globalLO**

Exercice 6 : Supprimez le TextView et la ressource associée (hello_world)

Exercice 7 : Ajoutez **3 RelativeLayout** au sein de la mise en page principale respectivement nommée **scoreLO**, **boardLO**, **controlLO** en respectant les proportions (10, 60, 30)

Indice : hauteur à 0dp (height) et affectation d'un poids (weight).

4.2. Gestion des couleurs

Exercice 8 : Créez un fichier XML de type ressources pour la gestion des couleurs dans « res/values » que vous nommerez *color.xml*

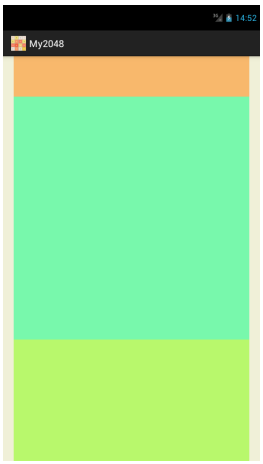
Exercice 9 : Ajoutez à ce fichier XML, 4 couleurs de votre choix (personnalisation !). Elles détermineront les couleurs de fond de chaque zone (global, score, board, control).

Exercice 10 : Revenons à l'interface utilisateur et affectez à chaque zone (global, score, board, control) une couleur définie dans votre fichier color.xml

Indice : propriété « Background » du layout et @color/

Exercice 11 : Afin d'observer la couleur de fond de la zone globale (globalLO) affectez lui une marge gauche et droite de 20dp (de chaque côté).

INFOS : Vous devriez obtenir un écran ressemblant à celui ci-dessous.



5. Plateau de jeu

IMPORTANT : Avant de passer à cette partie, assurez-vous (auprès de moi, ou par vous-même) que la partie « Mise en page » est fonctionnelle !

5.1. Plateau carré

Le plateau de jeu est destiné à recevoir 16 cases (4x4) et doit être carré.
Précédemment nous avons affecté une hauteur de manière proportionnelle (60%) mais cela n'assure pas que cette zone soit carrée.

Objectif : Obtenir une zone carrée en modifiant le dynamiquement le poids en fonction de l'écran.

La classe « Activity » dont hérite votre activité principale dispose d'une méthode nommé *onWindowsFocusChanged(boolean focus)*.

Exercice 12 : Dans le code de votre activité (java), surchargez cette méthode.

Exercice 13 : Dans cette méthode et à l'aide de la méthode *findViewById*, récupérez les 4 layouts sous forme d'objets Java.

Indice : findViewById retourne une *View* à vous de « caster » le type de retour en RelativeLayout pour un RelativeLayout etc. (Plus d'explication au besoin en tutorat).

La hauteur du plateau (**h**) est reliée à son poids (**w**) et la hauteur de globalLO (**H**) par la formule suivante : h = H*w/100.
Sachant que pour un carré, la hauteur (**h**) = la largeur (**l**), on en déduit la formule suivante

w = 100*l/H.

Exercice 14 : Récupérez les valeurs **l** et **H** afin d'en déduire **w**. (en float)

Exercice 15 : Maintenant que vous disposez de **w** (correspondant au poids du plateau pour qu'il soit carré), vous devez répartir le poids résiduel entre les zones **scoreLO** et **controlLO** respectivement de ¼ et ¾.

Poids de scoreLO (**wScore**) = 0,25*(100-w)
Poids de controlLO (**wControl**) = 0,75*(100-w)

Exercice 16 : Affecter les poids (wScore, w, et wControl) aux mises en pages respectives.

Pour cela, il faut
- Récupérer les « layout_parameters » de chaque layout (getLayoutParams)
- Modifier la valeur **weight**.
- Recharger les « layout_parameters » à chaque layout (setLayoutParams(X))

Exercice 17 : Modifiez les poids dans le XML de manière grossière (10, 10, 80) afin de s'assurer qu'à l'affichage de l'application le plateau devienne bien carré.

5.2. Structure du plateau

5.2.1. Interface

Une fois votre plateau parfaitement carré. Vous allez pouvoir disposez des tuiles à l'intérieur de ce dernier (4x4).

Info : Tuile = Case

Exercice 18 : Remplacer le RelativeLayout (**boardLO**) par un **TableLayout** en passant à modifier le code Java correspondant (findViewById, etc.)

Exercice 19 : Ajouter 4 **TableRow** (ligne) en ajustant leurs hauteurs par stratégie collective par **poids égaux** à savoir **25**(%) et leur largeur par remplissage total (père).

Exercice 20 : Ajouter une ressources de **couleur** nommée « **colBoard** » afin de remplir le fond de votre plateau de jeu. (Bordure entre les tuiles et pourtour du tableau, oui modifier le background déjà présente).

Exercice 21 : Insérer un **TextView** destiné à visualiser une tuile dans l'un des TableRow en réglant sa hauteur par remplissage du père et sa **largeur à 25**(%).

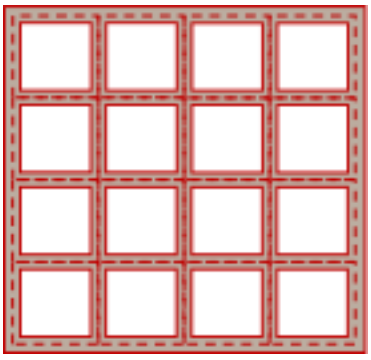
Exercice 22 : Régler les attribues et les valeurs suivantes
- **background** à **blanc** en passant par une ressource
- son contenu (**texte**) avec le texte « **131072** ».
- sa **gravité** sur « **center** »
- son **style** sur « **bold** »
- et sa **taille de police** de manière à visualiser le texte sur une ligne (on améliorera cela plus tard). 28sp ?
- Choisissez (pas de code ici) une valeur **d'intervalle entre les tuiles** (exemple : 10dp) puis régler la marge externe globale du TextView à la moitié de cette largeur (code xml, exemple : 5dp)

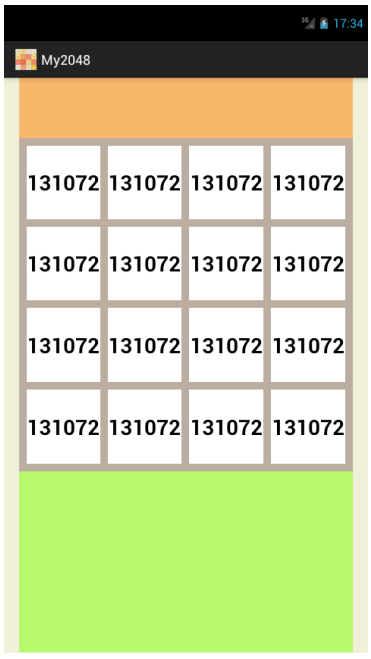
Exercice 23 : **Dupliquez** 3 fois ce TextView dans le même TableRow (ligne) afin d'obtenir une ligne avec 4 TextView.

Exercice 24 : **Dupliquez** les 4 TexView au sein des 3 autres TableRow afin d'obtenir une grille régulière de 4x4 tuiles.

Exercice 25 : Réglez les **marges internes** de **boardLO** à une valeur égale à la moitié de l'intervalle entre les tuiles choisi.

Exercice 26 : Vous devriez obtenir un résultat propre des illustrations ci-dessous.





5.2.2. Code

Exercice 27 : Pour chaque TextView, ajoutez un **identifiant** en respectant la suite logique **box<l><c>** ou l et c seront compris entre 0 et 3.

Exercice 28 : Dans votre activité, ajouter un membre interne privé de type **tableau 2D** de TextView nommé « **box** » sans oublier d'allouer l'espace (4x4).

Info : Le tableau 2D contiendra les TextView.

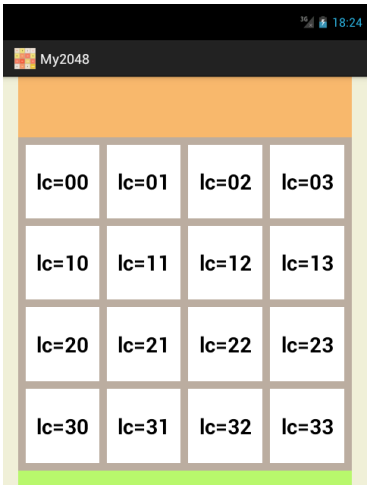
Exercice 29 : A la fin du onCreate(), ajoutez une déclaration d'un **tableau 2D d'entier** nommé « **boxid** » et initialisé à la main de manière à ce que chaque case contiennent **l'identifiant** de la TextView correspondant

Indice : R.id.box00

Exercice 30 : A l'aide d'une boucle 2D, **remplir votre tableau 2D** de TextView (box) avec les TextView récupérés à l'aide de findViewById et le tableau d'entier.

Exercice 31 : Afin de vérifier le bon fonctionnement du code, remplacer le texte de vos TextView par le texte suivant : lc=00, lc=01, etc.

Indice : utiliser setText dans la boucle 2D puis pensez à commenter l'appel au setText ensuite.



6. Mise en place du moteur

6.1. Codage des tuiles

Le plateau de jeu est une grille de 4x4 cases pouvant être vides ou contenir une tuile de valeur contenue dans une liste de puissances de 2 (2,4,8,16,…, 131072) qui seront disposé dans un tableau 1D statique.

De plus, chaque tuile disposera d'un état (**flag**) en respectant les règles suivantes.

- -1 : tuile ajoutée aléatoirement à la fin du tour.
- 0 : case vide ou tuile déjà présente avant la fin du tour.
- 1 = tuile nouvellement fusionnée

Exercice 32 : Au sein de la classe Game2048, ajouter une **classe publique et statique** pour la tuile nommé « **Tile** » avec 2 champs privés :
- **flag** : entier, état de la tuile
- **r** : entier, la puissance de 2 (0,1,2,3,4,5,…) et non la résultat de la puissance.

Exercice 33 : Ajoutez un **constructeur** par **défaut** (r=0 et flag=0)

Exercice 34 : Ajoutez un **constructeur** par **paramètres** (r et flag)

Exercice 35 : A la classe **Tile** ajoutez un **tableau statique 1D d'entier** privé, nommé « **pow2** ». Il sera initialisé à 0 pour la case 0 puis avec les valeurs croissantes des puissances de 2 à partir de 2^1 jusqu'à 2^{17}.

Exercice 36 : **Déclarez** et **définissez** les méthodes suivantes :

- int getRank() ; //getter sur r
- int value() ; //retourne 2^r en utilisant le tableau statique
- boolean isNew() ; //flag == -1 ?
- boolean isFusion() ; //flag == 1 ?
- String toString() ; //surcharge du toString() -> retourne une chaine vide si r==0 ou la chaine représantant 2^r sinon.

6.2. Codage du plateau de jeu

A la classe Game2048,

Exercice 37 : Ajoutez en membre privé un **tableau 2D de tuiles** nommé « **board** » et correctement dimensionné.

Exercice 38 : Ajoutez un **constructeur** par **défaut** créant par défaut les 4x4 Tile que **board** doit « tenir ».

Exercice 39 : Ajoutez une méthode d'instance publique **Tile getTile(int l, int c)** retournant board[l][c].

Exercice 40 : Ajoutez une méthode d'instance publique void init() qui remplit pour le moment le board[l][c].r avec des valeurs décroissantes à partir de 17.

6.3. Liaison avec l'interface

Pour tester le comportement du moteur, nous allons revenir à **MainActivity** afin d'y ajouter un lien avec le moteur.

Exercice 41 : Ajoutez à MainActivity un champ d'instance privé nommé « **game** » de type **Game2048.**

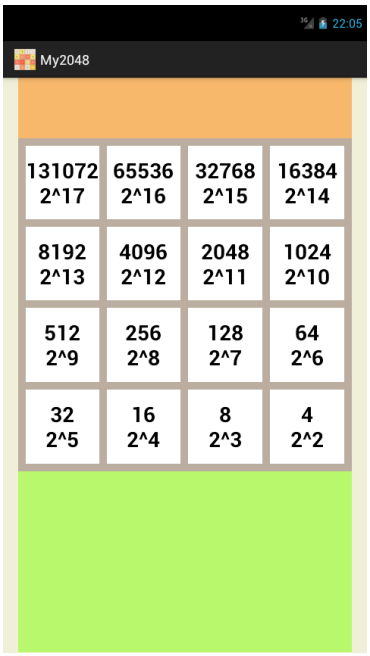
Exercice 42 : Initialisez votre instance de Game2048 à la fin du onCreate() avec la méthode **init()**.

Exercice 43 : Mettez en place une méthode **update()** de type **void** permettant de récupérer une à une (boucle 2D) les tuiles de **game** à l'aide de **getTile** et affiche dans le **TextView** (de même index) une chaine constituée à partir de le tuile récupérée.

Indice : Afficher le résultat de toString() auquel on concaténera « \n2^ » puis le résultat de getRank de façon à afficher la valeur puissance de 2 sur une première ligne et la vision avec exposant sur une seconde.

Exercice 44 : Après l'initialisation de game, effectuer une mise à jour de l'interface à l'aide de la méthode précédemment mise en place (update).

Vous devriez obtenir l'affichage suivant. Il est important d'obtenir le même affichage des valeurs pour la suite du TP. (Valeur et rang).



6.4. Colorisation des cases

Afin de distinguer visuellement les cases identiques (et différentes), nous allons ajouter une colorisation des cases en fonction de leur valeur/rang.

Exercice 45 : Définir une nouvelle ressource couleur pour chaque rang de case y compris pour 0 en nommant ces ressources « col<r> » ou <r> est le rang de valeur sur 2 chiffres (00, 01,…17)

Contrainte : Nous définirons que les tuiles dont le rang est inférieur ou égal à 3 disposeront d'un fond clair. A vous de créer vos couleurs en conséquence.

Indices :
- fichier colors.xml dans le dossier res.
- <color name="col00">#CDC1B4</color>

Exercice 46 : Ajouter 3 couleurs dans ce même fichier
- colNT : couleur du texte d'une tuile nouvellement ajoutée (ici rouge)
- colDT : couleur du texte d'une tuile à fond clair (ici noir)
- colBT : couleur du texte d'une tuile à fond sombre (ici blanc)

Exercice 47 : Créer un tableau statique privé d'entier nommé « **colId** » de taille 21. (0 à 17 couleurs de fond + 3 couleurs de textes).

Exercice 48 : Dans le onCreate(), **initialisez** à l'aide de R.Color.<r> votre tableau.

Nous disposons maintenant d'un tableau d'identifiant de ressources correspondant à nos couleurs. Cependant, les méthodes comme setBackgroundColor ou setTextColor nécessite une valeur de couleur que nous disposerons dans un second tableau.

Exercice 49 : Ajoutez un membre privé de type tableau 1D d'entier nommé « **color** » de même taille que « colId »

Exercice 50 : A l'aide d'une simple boucle for, initialisé votre tableau **color** en utilisant getRessources.getColor sur les cases colId de même index.

Indice : color[i] = getResources().getColor(*colId*[i]);

Nous allons maintenant mettre à jour le texte et les couleurs en fonction de la valeur. Dirigez-vous vers la méthode update() de MainActivity.

Exercice 51 : Modifiez le texte affiché afin d'y faire apparaître uniquement la **valeur de la tuile** (2,..32,64…)

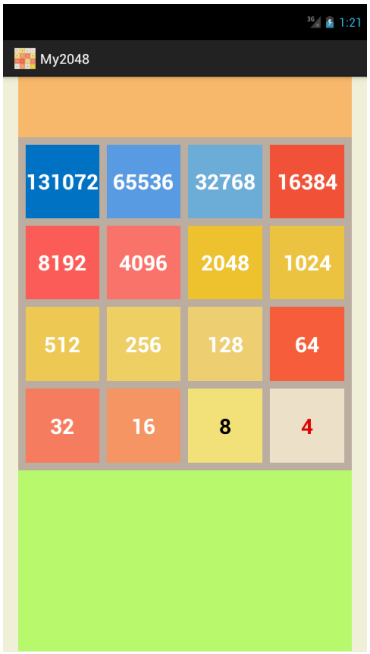
Exercice 52 : Modifiez la **couleur de fond** de la tuile à l'aide de setBackgroudColor. En utilisant le tableau « color » et le rang de la tuile courante.

Indice : color[tile.getRank()];

Exercice 53 : Réglez la **couleur du texte** suivant son état
- isNew()
- rang <= 3
- rang > 3

Exercice 54 : Modifiez la case 3,3 dans la méthode init de Game2048 afin qu'elle soit considéré comme nouvellement ajouté.

Vous devriez obtenir l'exemple ci-dessous (sauf couleurs perso).
Observez bien le 4 en rouge ! le 8 en noir et le reste en blanc (dans mon cas).



6.5. Gestion du score et du rang

Dans la classe 2048.

Exercice 55 : Ajoutez les champs d'instance privés suivants à la classe Game2048 :

- score (int)
- bestR (int)
- lastP (String).

Exercice 56 : Initialisez leurs valeurs à 0 ou chaines vide dans le constructeur ou à leur déclaration.

Exercice 57 : Ajoutez les **accesseurs** publics pour ces champs.

Dans le XML.

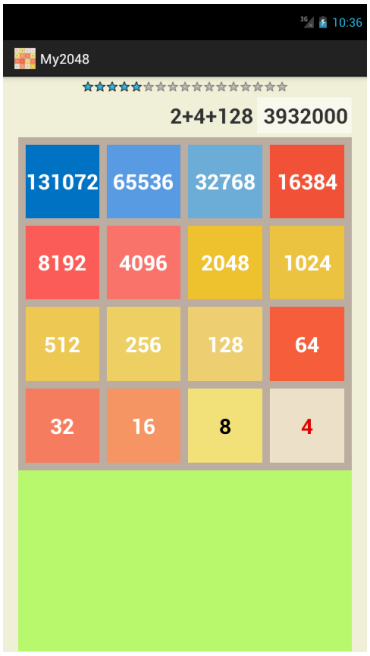
Exercice 58 : Donnez à scoreLO un **padding** léger sur les bords haut et bas seulement (5dp par exemple) et annuler son background au besoin.

Exercice 59 : Ajoutez dans le **scoreLO**, un **TextView** nommé **scoreTV** cadré à droite et en bas (positionnement relatif au père) avec les propriétés suivantes :
- Taille de la police à définir
- Gravité du texte à droite
- Style gras
- Hauteur selon son contenu
- Largeur sur une largeur fixe.
- Texte sur une seule ligne

Exercice 60 : Ajoutez un **background blanc** avec **un peu de transparence** à ce TextView avec un peu de marge interne (quelques dp).

Exercice 61 : Ajoutez à **scoreLO**, un autre **TextView** nommé **lastTPV** (copier-coller le scoreTV) ainsi qu'une « **ratingBar** » nommé « **bestTRB** » et attribuez des propriétés pour obtenir l'affichage suivant.

Infos :
- Rating bar : valeur initiale 0, valeur max 17.
- Texte du textView (question plus bas)



Exercice 62 : Dans le code de l'activité pensez maintenant à **ajouter** et **récupérer** les :
- 2 TextView (score et lastP)
- le ratingbar (bestT)

Exercice 63 : Effectuez leur **mise à jour** afin d'afficher les résultats des accesseurs concernés de Game2048.

Exercice 64 : Modifiez l'initialisation de Game2048 pour provisoirement mettre

- le « score » à 3 932 000
- le « lastP » à « 2+4+128 »
- et le « bestR » à 5.

Vérifiez votre affichage (affichage précédent)

7. Finalisation de l'interface

7.1. Barre d'action d'initialisation du jeu

Exercice 65 : Ajoutez à votre application un item à la barre de menu nommé « action_new » de titre @string/launch (ressource string à ajouter de valeur « nouveau jeu ») et d'icône un fleche d'actualisation (à vous de trouver la ressource).

Exercice 66 : La réponse au clic sur cette item doit appeler la méthode init(). Trouvez à quel endroit effectuez cet appel et mettez en œuvre.

Exercice 67 : **Commentez** les appels à init() et update() dans le onCreate() et **vérifiez** le fonctionnement de votre nouvel item. Une fois satisfaisante, **décommentez** les appels dans le onCreate().

7.2. Flèche de contrôle du jeu

Exercice 68 : Trouvez **4 images de flèches** de même taille, (haut, bas, gauche, droite) (ou une image à pivoter) qui vous servirons de commande du jeu.

Exercice 69 : Dans le **controlLO**, ajoutez **4 ImageButton** possédants les caractéristiques suivantes :

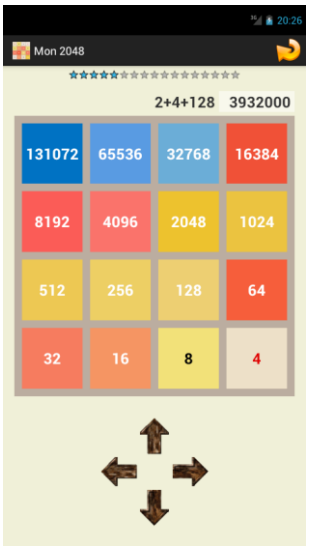
- identifiant de type buttonL pour gauche (=Left), buttonU pour haut (=Up), etc.
- Dimensions (selon le ratio de vos images) par exemple 66dpx48dp (et inversement)
- Background transparent
- Centré comme l'illustration plus bas.

Exercice 70 : Mettez en place directement dans l'activité 4 **onClickListener** répondant aux clics sur les 4 boutons.

Exercice 71 : Mettez en œuvre dans le MainActivity une méthode **tryMove(int dir)** vide pour le moment, qui sera appelez depuis le **onClick** des listener précédemment mis en place suivant la réglé suivante

- **Gauche : tryMove(0) ;**
- **Haut: tryMove(1) ;**
- **Droite : tryMove(2) ;**
- **Bas : tryMove(3) ;**

Vous devriez obtenir un écran proche de celui ci-dessous.



Voici les ressources que j'ai utilisées.

8. Gestion du jeu

8.1. Initialisation du jeu

8.1.1. Initialisation

Dans la classe Game2048.

Exercice 72 : Dans la méthode init(), mettez toutes les cases à l'état vide (r = 0 et flag = 0) ainsi que les champs score = bestR = 0 ; et lastP = "". Vérifiez l'affichage.

Exercice 73 : Ajoutez un champ **nbv** à la classe Game2048.

Exercice 74 : Ajoutez une **méthode privée addTile()** (vide pour le moment) et lancez la **2 fois** dans le init() après avoir initialisé **nbv** à **16**.

8.1.2. Tirage au sort

Le jeu commence avec 2 tuiles. Ces 2 tuiles peuvent avoir la valeur 2 ou 4 (rang 1 ou 2) au démarrage du jeu. 90% du temps, la valeur 2 (rang 1) est choisie.

Exercice 75 : Ajoutez un champ privé de type **Random** nommé « **rand** ». Servira à ajouter un

Exercice 76 : Ajoutez une méthode privée **set(int rk, int fl)** à la classe Tile. Cette méthode permettra de modifier le rang et le flag d'une tuile.

Procédure du tirage au sort
1/ tirer au sort [0,nbv[
2/ Parcourir le jeu (board) en comptant les cases vides rencontrées jusqu'à en avoir trouvé autant que l'entier aléatoire obtenu.
3/ A cette case, ajoutez la tuile avec les valeurs rang = 1 et flag = -1 (pour l'instant).
4/ Décrémentez nbv pour tenir compte de l'ajout.

Exercice 77 : Codez cette procédure dans la méthode **addTile** en pensant à initialiser correctement votre nombre aléatoire.

Étape importante pour la suite du TP.

Exercice 78 : **Vérifiez** le bon placement des tuiles (logs, débug, nbv initial, valeur du rand, index l et c, case choisi etc...)

Vous allez maintenant modifier la méthode addTile() pour tirer au sort le rang de la tuile nouvellement ajoutée.

Exercice 79 : A l'aide de votre « Random » tiré aléatoirement un nombre entier entier [0,100[ou un double entre [0,1[. En déduire le rang de la tuile en respectant les probabilités souhaitées (90% : rang = 1,10% : rang = 2).

Exercice 80 : Pensez à mettre à jour bestR si la tuile ajoutée est d'un rang supérieur à sa valeur actuelle.

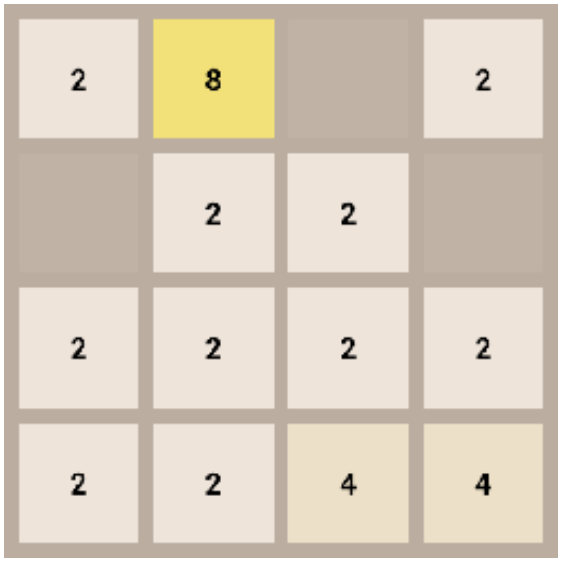
Exercice 81 : Vérifiez le bon fonctionnement du tirage en cliquant plusieurs fois de suite sur la flèche de votre barre d'action.

8.2. Gestion d'un coup

8.2.1. Préparation

Un coup correspond à une demande du joueur de « compactage » du jeu vers l'un de ses bords codés par l'entier « dir » paramètre de tryMove (0 pour gauche ou ouest, 1 pour haut ou nord, 2 pour droit ou est, 3 pour bas ou sud).

Exercice 82 : Afin de tester vos développement à venir dans move, codez une méthode public provisoire de Game2048 de prototype **void initTest()** qui initialise le jeu « à la main », sans tirage aléatoire au contenu de plateau ci-dessous. Remplacez provisoirement dans onCreate() l'appel à init par l'appel à initTest() et validez votre affichage de démarrage d'Activité selon plateau ci-contre.



Exercice 83 : Ajoutez à la classe Game2048 une méthode **void move (boolean croiss, boolean vert)**.
Infos : Cette méthode indique si le coup est horizontal ou non avec le booléen « vert », et quel bord est visé avec le booléen « croiss ». La combinaison de ces 2 booléens détermine le bord visé.

Exercice 84 : Appelez judicieusement depuis la méthode **tryMove()** la méthode **move()** avec les paramètres qui vont bien en fonction des 4 différents cas (déplacement possible).

Exercice 85 : Dans move(), affichez un log ou un toast permettant de vérifier les valeurs lors d'un clic sur une flèche.

8.2.2. Comptage directionnel

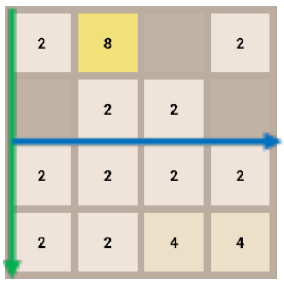
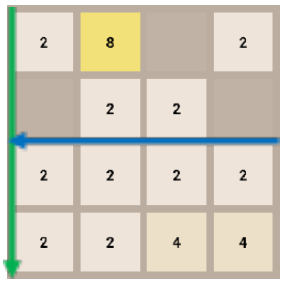
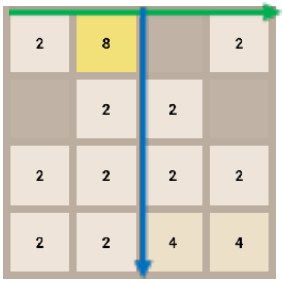
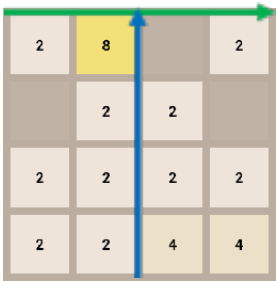
Le « compactage directionnel » peut être simplifié en le traitant par ligne (vert=false) ou colonne (vert=true) en 3 étapes :

1. *Empilement des rangs des tuiles non vides par parcours selon « croiss » de la ligne ou colonne. Cet empilement sera réalisé dans un ArrayList<Integer> pile de capacité 4 réutilisé après vidage par chaque ligne ou colonne. Les rangs r de tuiles non vides (r>0) y seront stockés au cours du parcours de la ligne ou colonne par pile.add(r) donc dans des numéros de case croissants*

2. *Parcours croissant de pile avec transfert des rangs empilés dans la ligne ou colonne dans l'ordre dicté par « croiss » : fusion de 2 cases de pile voisines et de même rang (rang augmenté de 1 et « consommation » des 2 cases de pile pour 1 case de la ligne ou colonne) lorsque cela se présente ou recopie simple du rang sinon (consommation d'1 case de pile pour 1 case de la ligne ou colonne).*

3. *Complément éventuel de la ligne ou colonne traitée par des cases vides.*

Pour éviter de gérer les 4 directions séparément et constatant que leur traitement ne diffère que par l'indexation des cases du jeu, nous allons proposer une méthode d'accès aux cases prenant en compte les 2 booléens « croiss » et « vert » et 2 index entiers « lc » identifiant la ligne ou colonne et « i » identifiant dans le sens de parcours « croiss » la case de cette ligne ou colonne. Le parcours réel du jeu lors des évolutions croissantes de 0 à 3 de « lc » et « i » sont identifiés ci-dessous par les flèches vertes pour « lc » et bleues pour « i »

Gauche	**Droite**	**Haut**	**Bas**
Parcours de gauche à droite	Parcours de droite à gauche	Parcours de haut en bas	Parcours de bas en haut
Évolution de « lc » de haut en bas	Évolution de « lc » de haut en bas	Évolution de « lc » de droite à gauche	Évolution de « lc » de droite à gauche
			
Vertical = faux	Vertical = faux	Vertical = vrai	Vertical = vrai

Croissant = vrai	Croissant = faux	Croissant = vrai	Croissant = faux
Tableau de rangs suivant le parcours			
1301 0110 1111 1122	1031 0110 1111 2211	1011 3111 0112 1012	1101 1113 2110 2101

Exercice 86 : Codez dans Game2048 la méthode privée **Tile getTile (int lc, int i, boolean croiss, boolean vert)** qui retourne la bonne tuile du jeu les selon explications ci-dessus.

Infos : Pour cela, remarquez que dans l'accès aux tuiles par board[][], vert contrôle l'utilisation de « lc » comme premier ou second index alors que l'autre sera i ou 3-i selon « croiss ».

Exercice 87 : Dans la méthode **move()**, remplacer le toast/log par une boucle 2D « lc » de 0 à 3 puis « i » de 0 à 3 permettant de remplir une chaine de 4 lignes avec les rangs parcourus.

Exercice 88 : Déclarez un chaine vide nommée « info » avant les boucles puis ajoutez (=concaténez) lui le rang retournée par **getTile(lc,o,croiss,vert)**. Pensez à ajouter des '\n' à la fin du corps de la boucle externe « lc ».

Exercice 89 : **Affichez** la chaine dans le LogCat après la boucle puis **vérifiez** en appuyant sur les flèches le bon résultat (respectivement gauche, droite, haut et bas).

Important : Ne passez pas à la suite que si votre code produit le bon résultat !

8.2.3. Gestion d'un coup

Exercice 90 : Ajoutez à la méthode move(), une variable local **ArrayList<Integer>** nommée pile, initialisé comme vide avec une capacité de 4. **Videz la pile** avant chaque boucle interne.

Exercice 91 : Modifiez la boucle interne pour **vérifier** si le rang de la tuile retourné par getTile est **non nul**. Uniquement dans ce cas, **ajoutez ce rang à la pile**. Retirer à la pile l'ajout à la chaine « info ».

Exercice 92 : Ajoutez à la suite de la boucle interne (mais toujours dans la boucle externe), une autre boucle parcourant la pile de manière croissante par un entier « ip » en ajoutant à la chaine « infos » les entiers obtenus par pile.get(ip).

Vous devriez obtenir le résultat suivant (aucun '0').

Tableau de rangs suivant le parcours			
131 11 1111 1122	1031 11 1111 2211	111 3111 112 112	111 1113 211 211

Exercice 93 : Modifiez maintenant votre seconde boucle interne (parcours de la pile) en y ajoutant un second index « i » croissant à partir de 0 par pas de 1 sans modifier le test de poursuite basé sur « ip ».

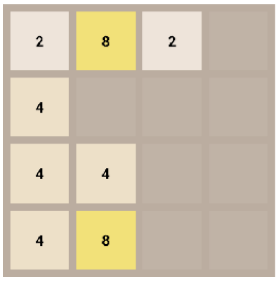
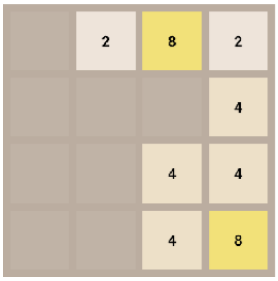
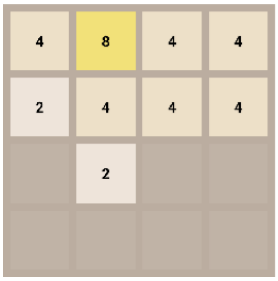
Infos : L'index « i » servira à adresser la bonne tuile à remplir par getTile(lc,i,croiss,vert)...

Exercice 94 : Dans cette boucle, placez un test pour détecter si la case « ip » de la pile n'est pas la dernière et si sa suivante est de même valeur. Si ce cas est détecté il faut **fusionner** ces 2 cases « ip » et « ip+1 » dans la tuile lc,i. Pour cela mettez à jour la tuile retournée par getTile(lc,i,croiss,vert) en lui attribuant un rang égal à 1 + la valeur commune des cases ip et ip+1 de pile (2^r+2^r donne $2^{(r+1)}$) et un flag 1 (fusion). Pour cela, utilisez la méthode set de Tile.

Exercice 95 : Dans le cas contraire assignez à la même tuile que ci-dessus un rang égal à la valeur de la case ip de pile et un flag 0.

Exercice 96 : Enfin ajoutez une troisième boucle interne pour mettre toutes les cases restantes de la ligne ou colonne à vide : en partant de la valeur de i au sortir de la boucle précédente (valeur qui correspond à la prochaine case à remplir selon croiss), et en faisant croitre i par pas de 1 jusqu'à 3 inclus. Mettez les tuiles concernées getTile(lc,i,croiss,vert) à vide (rang et flag à 0) par la méthode set.

Vérifiez que, partant de l'état après initTest(), les différents mouvements (appuis sur les boutons flèches) vous donnent les jeux suivants :

Gauche	Droite	Haut	Bas
			

8.2.4. Validité d'un coup

La validité d'un coup est valable pour les deux cas suivants :
- *au moins une fusion a été réalisée ou*
- *une case vide maintenant occupée par une tuile*

Exercice 97 : Dans la méthode *move(),* ajoutez un booléen nommé « modif » initialisé à *false*. Mettez le à *true* au moment d'une fusion et lorsque l'on empile un rang supérieur à 0 alors que des cases vides ont été passées préalablement lors du parcours de la ligne ou colonne.

Indice : ce second cas peut être détecté en comparant l'index i à la taille de la pile.

Exercice 98 : Modifiez le type de retour de la méthode *move()* pour un booléen et retournez « modif ».

Exercice 99 : Modifiez la méthode *tryMove()* dans le MainActivity de façon à ce que *update()* ne soit lancée que si *move()* retourne *true* et qu'un *Toast* soit affiché sinon « Mouvement invalide ».

Testez vos déplacements jusqu'à obtenir l'affichage du Toast.

8.2.5. Gestion du score

Nous pouvons maintenant aborder la gestion du score, de l'affichage des points obtenus sur ce coup et du plus haut rang. Il faut cependant que ces informations n'évoluent que lors d'un mouvement valide.

Exercice 100 : Les mises à jour de score, et de « bestR » peuvent être menées à la volée lors des fusions puisque chaque fusion implique que le mouvement est valide. Pour cela obtenez la valeur associée au rang attribué à la tuile fusionnée (pensez à Tile.pow2…) et ajoutez cette valeur à score. Mettez aussi à jour bestR si le rang fusionné lui est supérieur.

Par contre « lastP » ne doit être écrasée que si le mouvement était valide mais la chaine à produire doit être remise à vide avant les boucles.

Exercice 101 : Réutilisez donc la chaine « info » (initialisée à vide) comme avatar de la future « lastP ». Ajoutez lui à chaque fusion les points ajoutés à score précédés (si info n'est pas vide) de '+'. En fin de méthode, juste avant le retour, si « modif » vaut *true*, affectez « lastP » d'après « info ».

Testez les modifications des affichages (scoreTV, lastPTV et bestTRB) et leur cohérence…

8.2.6. Ajout d'une tuile

Exercice 102 : Gérez « nbv » dans *move()* en le remettant à 0 avant les boucles et en

- lui ajoutant après la seconde boucle interne le nombre de cases à mettre à 0 dans la ligne ou colonne actuelle

OU

- l'incrémentant à chaque mise à vide d'une case dans la troisième boucle interne.

Exercice 103 : En fin de méthode, si « modif » vaut *true*, ajoutez une tuile nouvelle (même procédure que dans init, avec lancement de addTile()). Remplacez l'appel à initTest par celui d'init dans onCreate et vérifiez le fonctionnement général du jeu.

8.3. Détection de victoire

Nous souhaitons maintenant affichez un Toast de victoire et modifier la couleur des étoiles du ratingBar lorsque le joueur obtient sa première tuile « 2048 ». La partie continue néanmoins pour laisser la possibilité de tuiles encore plus élevées…

Exercice 104 : Ajoutez une méthode publique **boolean *hasJustWon()*** à Game2048. Pour la coder, ajoutez un champ d'instance privé **boolean goal** (but atteint), initialisez le à false dans *init()*. La méthode hasJustWon() retourne true si le but n'était pas précédemment atteint et que la meilleure tuile actuelle est de rang 11. Dans ce cas, la méthode aura mis goal à *true* avant de quitter.

Exercice 105 : Modifiez update dans MainActivity pour y lancer hasJustWon() et, si elle retourne *true*, afficher un Toast dont le message est une ressource string nouvelle nommée « win » de valeurs "Vous avez gagné !".

8.4. Détection de fin de partie

Pour finir, il faut détecter et gérer la fin de partie, lorsque plus aucun mouvement n'est valide.

Exercice 106 : Ajoutez une méthode publique boolean **isOver()** à Game2048. Cette méthode retourne *false* si au moins une tuile est vide (nbv>0) ou, sinon, si au moins un couple de tuiles voisines sont de même rang. Pour tester cette seconde condition plus complexe on lancera une boucle 2D sur les 16 cases en vérifiant pour chacune si elle est de même rang que l'une au moins de ses voisines nord et ouest si elles existent. Dès qu'une telle condition est réalisée, on retourne false

Exercice 107 : Modifiez *update()* dans Mainactivity pour y lancer isOver et, si elle retourne true : • afficher un Toast dont le message est une ressource string nouvelle nommée « gameOver » de valeurs "Game over"/ "Fin de partie" • affichez cette même chaine dans lastPTV (lastP.setText…) • rendez les 4 boutons flèches invisibles par View.setVisibility(View.INVISIBLE) Pensez à rendre les boutons visibles par View.setVisibility(View.VISIBLE) dans le cas contraire.

9. Persistance et sauvegarde des données

Il s'agit là d'une solution de sauvegarde, bien qu'il en existe d'autres méthodes.

Maintenant que le jeu est fonctionnel. Il faut nous assurer de sa persistance en cas d'arrêt inopiné (par le système) et, possiblement de proposer une possibilité de sauvegarde, voire de « undo ». Cela passe à chaque fois par la définition d'un jeu minimal d'informations permettant de représenter le jeu dans son intégralité et de le récréer sans autre information. Pour cela nous allons définir une classe interne de Game2048, nommée SaveBundle contenant ce jeu minimal d'information et les méthodes nécessaires de Game2048 permettant de le générer pour représenter le jeu en cours ou de réinitialiser le jeu dans l'état sauvegardé d'après ses informations.

9.1. Jeu de sauvegarde minimal

Dans la classe Game2048, certains champs internes sont calculables d'après l'état du plateau (bestR, nbv, goal) et d'autres ne nécessitent pas d'être sauvegardés (rand). Enfin, les sauvegardes en bundle comme en préférences se faisant pas couples clé-valeur simple, il apparait intéressant de proposer un codage simple, en String, du contenu du plateau. Nous allons alors constituer cette chaine représentant le plateau en y codant les tuiles sur 2 char, à la suite, dans un parcours classique (boucle externe croissante sur les lignes, boucle interne croissante sur les colonnes). Pour chaque tuile les 2 chars générés coderont le flag pour le premier ('+' pour flag==-1 (nouvelle tuile), ' ' pour flag ==0, '^' pour flag==1 (tuile fusionnée)) et le rang pour le second (codage alphanumérique issu selon le rang de la chaine " 123456789ABCDEFGH" donnant le rang 1 à 17 sur une lettre ou blanc si tuile vide (rang 0)).

Exercice 108 : Ajoutez une méthode publique **String Log()** à Game2048.Tile pour coder la tuile sur 2 caractères selon principe ci-dessus.

Exercice 109 : Ajoutez aussi une méthode publique **void setFromLog(String log)** pour ré-initialiser la tuile d'après les 2 caractères de la chaine log paramètre.

Indice : Le codage de ces 2 méthodes pourra s'appuyer sur 2 Strings privées statiques initialisées à " 123456789ABCDEFGH" et "+ ^".

Exercice 110 : Ajoutez ensuite à Game2048 une seconde classe interne statique et publique nommée **SaveBundle**. Elle contiendra en champs d'instance publiques deux **String board et lastP**, et un **int score**. Dotez de constructeurs par défaut (Strings à "" et int à 0), et par extension (paramètres pour chaque champ).

Dotez ensuite Game2048 des méthodes d'instance suivantes :

Exercice 111 : **private String logBoard()** qui construit la chaine résultat par concaténation des logs individuels des tuiles parcourues dans le sens usuel (lignes croissantes en externe, colonnes croissantes en interne).

Exercice 112 : **public SaveBundle getSaveBundle()** qui construit par extension le SaveBundle résultat à partir de l'instance de jeu ***this*** grâce notamment à logBoard.

Exercice 113 : **public void restoreFromSaveBundle(SaveBundle sb)** qui réinitialise le jeu complet à partir des informations continues dans sb. Il convient notamment de recalculer les éléments non sauvegardés (bestR, nbv, goal).

9.2. Gestion de la persistance

Prenez conscience du problème de persistance : effectuez quelques coups pour faire évoluer l'état de votre jeu et basculez l'orientation de votre terminal (touche 7 pavé num sur émulateur). Vous devriez constater une mort/renaissance de votre application avec passage par onCreate … donc ré-initialisation du jeu...

*Pour remédier à ce problème, vous allez utiliser la méthode intégrée permettant de sauvegarder les informations nécessaires de l'application dans un **Bundle android** puis de restaurer l'état de l'application grâce à ce Bundle. Cela est opéré par surcharge de méthodes dédiées appelées par Android à cet effet.*

Exercice 114 : Surchargez donc les méthodes **protected void onSaveInstanceState(Bundle outState)** puis **protected void onRestoreInstanceState(Bundle savedInstanceState)**.

Ces méthodes opéreront les sauvegardes/restaurations via transferts entre

- Game2048.SaveBundle et Bundle (champs de SaveBundle écrits/lus dans le Bundle sous forme de couples clés-valeurs)

Et

- le transfert Game2048 - SaveBundle grâce à Game2048 (getSaveBundle) et Game2048 (restoreFromSaveBundle)

N'oubliez pas relancer les méthodes héritées super.on….InstanceState. Vérifiez que le test ci-dessus opéré est maintenant fonctionnel (appli récrée sans changement d'état après changement d'orientation).

10. Personnalisation du jeu 2048

Exercice 115 : Maintenant que tout fonctionne, on vous demande de personnaliser le jeu à votre guise. Voici quelques fonctionnalités qui pourront vous inspirez :

- Portabilité de l'application (paysage, multi-terminaux)
- Gestion de l'écran tactile (gauche, droite, haut, bas par un simple geste)
- Utilisation des "Gesture" afin d'accroitre l'expérience utilisateur.
- Sauvegarde des données dans une base de données (classement, jeu en cour, utilisateur, etc.)
- Écrans supplémentaires : SplashScreen, Crédits, Paramètres, Aide, etc..
- Partage des scores sur les réseaux sociaux
- Historique des scores (HighScore)
- Choix d'un thème de couleurs
- Version droitier et gaucher
- Fonction Undo/Redo
- Partage social
- Bluethooth
- Animation
- Audio
- Etc..