TP1 – Application du jeu pour Android

INF8405 – Informatique Mobile

Hiver 2016

Par

Clément Gamache - 1642792

Cédric Noiseux - 1588195

Soumis à

Aurel Josias Randolph

Polytechnique de Montréal

17 février 2016

Introduction

Le but de ce travail pratique était de concevoir une application mobile de jeu mono-joueur inspiré du jeu déjà existant *flow-free*. Le but du jeu est de relier deux points de même couleur à travers une grille, par des déplacements horizontaux et verticaux, en générant des « tubes » tout au long du parcours. Pour compléter un niveau, toutes les paires de jetons doivent être reliées et toutes les cellules d'une grille doivent être utilisées. Une partie échoue si des tubes se croisent ou si des cellules ne sont pas utilisées. L'application conçue contient 6 niveaux prédéfinis de difficultés et de tailles variées.

Le fonctionnement de l'application est assez simple. D'abord, un choix de grilles de départ est proposé à l'utilisateur, soit une grille 7 x 7 ou 8 x 8. Ensuite, un choix de niveaux est proposé : niveau 1, niveau 2 ou niveau 3. Au départ, le niveau 1 de chaque grille seulement est disponible. Le succès d'un niveau permet d'accéder au niveau supérieur, mais le joueur peut également rejouer la même partie ou une partie précédente. En tout temps, le joueur peut quitter le jeu.

Présentation Technique

L'application mobile *flowfree* a été construite avec 3 activités et 1 classe java, respectivement : *MainActivity*, *SecondaryActivity*, *Level* et *Game*. Ces quatre composantes seront présentées individuellement plus en détail.

MainActivity

L'activité *MainActivity* affiche la fenêtre principale de l'application lorsqu'elle est démarrée. Le logo du jeu est affiché, et ce pour toutes les activités suivantes, de même que trois boutons : 7x7, 8x8 et EXIT. Les boutons appellent chacun une méthode spécifique lorsque l'utilisateur appuie sur ceux-ci. Ces méthodes sont respectivement : *goToLevel_7x7*, *goToLevel_8x8* et *showSimplePopUp.*

La méthode *goToLevel_7x7* permet de démarrer la deuxième activité *SecondaryActivity* et lui envoie un booléen TRUE pour lui indiquer les dimensions de la grille de jeu (7x7).

La méthode *goToLevel_8x8* permet de démarrer la deuxième activité *SecondaryActivity* également et lui envoie un booléen FALSE pour lui indiquer les dimensions de la grille de jeu (8x8).

La méthode *AppExit* permet de quitter l'activité et de retourner à la fenêtre d'accueil de l'appareil mobile utilisé. Cette méthode est *protected* pour qu'elle puisse être utilisée par les activités suivantes.

La méthode *showSimplePopUp* permet d'afficher un message à l'écran qui demande la confirmation de l'utilisateur pour quitter l'application. Si l'utilisateur désire quitter l'application, la méthode *AppExit* est appelée lorsque l'option positive est appuyée, sinon, la fenêtre s'efface de l'écran.

SecondaryActivity

L'activité *SecondaryActivity* affiche une fenêtre présentant les différents niveaux disponibles. Quatre boutons sont affichés : 1, 2, 3 et EXIT. Le bouton EXIT agit de la même manière que dans *MainActivity*. Les boutons 1, 2 et 3 appellent tous *playLevel(int level, boolean grid)* avec comme seul différence la valeur des paramètres.

La méthode *playLevel* prend en paramètre le niveau de jeu choisi de même que le booleen *grid* qui a été envoyé par *MainActivity*. La paramètre *grid* sera identique pour les trois boutons, TRUE ou FALSE dépendamment de la grille choisie, et le niveau du jeu sera 1, 2 ou 3 dépendamment du bouton qui a été appuyé. La méthode compare le niveau reçu en paramètre avec les variables statiques *maxLevelAllowed7x7* et *maxLevelAllowed8x8*. Ces variables permettent de connaitre les niveaux pour chaque type de grille qui ont déjà été complétés par l'utilisateur. Si *level* reçu en paramètre est plus élevé que *maxLevelAllowed7x7* ou *maxLevelAllowed8x8*, selon la grille choisie, une fenêtre d'avertissement est affichée à l'écran et l'accès au niveau est refusé. Dans le cas contraire, l'activité *Level* est démarrée et le niveau du jeu choisi par l'utilisateur est généré.

Level

L'activité *SecondaryActivity* affiche une fenêtre contenant un niveau de jeu, son titre, un bouton BACK, NEXT, RETRY et EXIT. Le bouton EXIT agit de la même manière que dans *MainActivity*. Tous les niveaux de jeu et leur titre sont générés automatiquement au sein de cette activité selon les manipulations de l'utilisateur sur l'application. Les boutons BACK, NEXT et RETRY appellent respectivement les méthodes : *back(int level)*, *next()* et *retry()*. La grille affichée à l'écran permet qu'on interagisse avec elle, une méthode *handleTouch(MotionEvent event)* gère ces interactions.

La méthode *back* permet de revenir au niveau précédent celui courant. Si le niveau est le premier de la grille 7x7, l'activité *SecondaryActivity* est redémarrée. Si l'utilisateur est au premier niveau de la grille 8x8,

l'accès au niveau 3 de la grille 7x7 est refusé s'il n'a pas été complété d'abord. Pour tous les autres cas, la méthode génère le niveau précédent. La méthode *updateHeader* est ensuite appelée. Cette méthode permet de mettre à jour le titre du niveau en fonction du numéro de celui-ci et de la taille de la grille.

La méthode *next* permet de naviguer au niveau supérieur à celui courant. Si *maxLevelAllowed7x7* ou *maxLevelAllowed8x8*, dépendamment de la grille, est plus élevé que le niveau supérieur qui désire être accédé, le prochain niveau est généré. Par exemple, l'utilisateur commence à jouer par le niveau 1 de la grille 7x7, il désire accéder au niveau 2 et clique donc sur NEXT. La variable statique *maxLevelAllowed7x7* est initialisée à 1 à l'ouverture de l'application. Puisque le niveau supérieur est de grandeur 2 et *maxLevelAllowed7x7* de grandeur 1, l'accès lui est refusé. Si l'utilisateur avait complété le niveau 1 précédemment, *maxLevelAllowed7x7* aurait été incrémenté à 2 et le niveau 2 aurait pu être accéder à partir du niveau 1. Dans le cas où l'utilisateur a complété le niveau 3 de la grille 8x8 et qu'il clique sur NEXT, une fenêtre s'affiche à l'écran pour lui informer qu'il a complété le jeu.

La méthode *retry* permet de recommencer le niveau courant. La méthode *restart* de la classe *Game*, expliquée dans la prochaine section, est appelée avant de régénérer le niveau maintenant dépourvu des chemins.

La méthode *handleTouch* permet de gérer toutes les actions que l'utilisateur fait sur la grille de jeu. Trois types d'action sont reconnus par cette méthode : ACTION_DOWN lorsque l'utilisateur pose son doigt sur la surface de jeu, ACTION_UP lorsque l'utilisateur retire son doigt et ACTION_MOVE lorsque l'utilisateur bouge son doigt sur la surface. Chaque action appelle une méthode propre à la classe *Game* qui sera expliquée dans la prochaine section. La grille de jeu est mise à jour après que l'action est reconnue et la méthode appelée. *handleTouch* vérifie ensuite si la partie est terminée, donc, si le nombre de chemins est égale au nombre de paires de points. Si c'est le cas, on vérifie si la partie est gagnée ou non. Si la réponse est positive, *maxLevelAllowed7x7* ou *maxLevelAllowed8x8* est incrémenté et une fenêtre apparait pour mentionner la victoire et deux options sont offertes : accéder au niveau suivant ou recommencer le niveau courant. Si la réponse est négative, une fenêtre apparait pour mentionner la défaite à l'utilisateur et il retourne au niveau courant. Finalement, la méthode *updateHeader* est appelée à la toute fin.

Game

Game est la classe qui contient toutes les informations et les méthodes de contrôle sur la partie en cours. Elle possède trois rôles principaux : contenir l'état du niveau en cours, gérer les événements et afficher l'état de la grille.

C’est grâce aux variables privées de cette classe qu’est enregistré l’état de la partie. En effet, un objet de la classe Game possède une liste de chemins tracés, le chemin en cours de traçage et les propriétés de la grille.

Les événements envoyés par l’utilisateur sont un doigt posé sur la grille, le déplacement du doigt sur cette dernière et finalement la levée du doigt de l’écran. Les méthodes *down()*, *move()* et up() implémentent respectivement ces gestions d’événements. Ces méthodes appellent les méthodes qui contrôlent l’évolution du jeu pour son bon fonctionnement.

Finalement, l’affichage de la grille se fait aussi au sein de la classe Game. Ce fut pratique puisque cette dernière contient déjà toutes les informations à l’affichage. La classe *level* lui envoie donc la vue du carré pour que Game puisse librement y afficher ce qu’elle veut avec l’aide d’un Canvas dans lequel est premièrement tracée l’image de la grille. Par-dessus cette image sont tracés tous les cercles initiaux du niveau en cours, puis les chemins tracés par l’utilisateur et finalement le chemin en cours de traçage.

Difficultés Rencontrées

Quelques difficultés ont dû être surmontées pour obtenir le résultat désiré. D’abord, la nature des Activités et leur logique. Il a fallu se familiariser avec ce concept avant de pouvoir appliquer la logique désirée pour l’application. De manière similaire, la façon d’interagir avec les widgets et les layouts n’étaient pas une compétence maitrisée à prime abord.

Trouver la manière de modifier les fichiers xml a aussi été ardue, par exemple, pour modifier l’apparence des boutons. Il a fallu se familiariser avec la grande quantité d’options offertes et la structure de ce type de fichier.

Il a fallu un certain temps avant de trouver un moyen de gérer et de manipuler les interactions utilisateur/application. Cette difficulté a été surmontée grâce à un *setOnTouchListener* appliqué à un *ImageView* représentant la grille de jeu. Il a ainsi été possible de suivre les actions d’un utilisateur précisément, sur une grande surface et de manière continu.

La manière de dessiner les points et les chemins sur la grille a aussi causé problème. Après avoir fait quelques recherches, l’utilisation des classes par défaut *Canvas* et *Paint* a été jugée idéale pour accomplir les tâches de dessin.

Critiques et Suggestions

Quelques décisions auraient pu être faites différemment au niveau de l'implémentation. D'abord, la logique derrière le traçage des chemins et des points. Tous les points et chemins sont redessinés à chaque fois que la méthode *handleTouch* est appelée, ceci peut causer un délai avant qu'un chemin soit tracé lorsque l'utilisateur bouge son doigt sur l'écran. Une autre façon de faire aurait pu être plus performante.

De plus, les deux grilles ont été insérées dans l'application en tant qu'image. Par conséquent, la qualité des grilles semble moins élevée que le reste de l'application et le positionnement des points moins précis. Les grilles auraient pu être tracées de manière similaire aux chemins et points de jeu pour obtenir un résultat d'avantage uniforme.

Une addition intéressante à l'application conçue aurait été la génération aléatoire de niveaux de jeu à chaque fois qu'un utilisateur démarre une nouvelle instance de l'application mobile. Ceci aurait pour conséquence d'inciter d'avantage l'utilisateur à rejouer à cette version du jeu *flowfree*. Également, un choix de grilles et de niveaux de jeu plus important rendrait l'application plus complète et intéressante.