

Département de génie informatique et de génie logiciel

**INF8405 - Informatique Mobile**

**Laboratoire #1 : Application de jeu pour Android**

**Sacha Licatese-Roussel, 1635849**

**Yann Dago, 1635682**

**Golnoush Rahimzadeh, 1759612**

**Travail soumis à Aurel Josias Randolph**

**Remis en date du**

**17 février 2016**

# **Sommaire**

[Sommaire 2](#_Toc443136494)

[Introduction 3](#_Toc443136495)

[Présentation technique du travail 3](#_Toc443136496)

[Aspect de présentation de l’application 4](#_Toc443136497)

[Logique d’avancement au sein du jeu 5](#_Toc443136498)

[Logique & construction de la grille de jeu 5](#_Toc443136499)

[Difficultés rencontrées 7](#_Toc443136500)

[Critiques et suggestions 7](#_Toc443136501)

[Conclusion 7](#_Toc443136502)

# Introduction

**Dans ce premier travail pratique nous allons nous intéresser au développement d’une application de jeu communément appelée flow-free par l'entremise du logiciel Android Studio. Nous étudierons le fonctionnement des activités, composants, gestion des évènements et menus. Finalement, nous examinerons l’application de jeu sur différents appareils pour assurer l’utilisabilité et la portabilité.**

# Présentation technique du travail

Dans cette première partie, nous présenterons les différents aspects relatifs à la conception des différentes parties de l’application et la manière selon laquelle nous avons abordé chacun de ces aspects techniques.

Nous avons premièrement analysé les requis liés à l’énoncé de laboratoire afin de s’assurer de bien incorporer chacun des éléments essentiels. Voici donc une liste qui représente la division du travail qui s’est effectuée au début du travail :

* Aspect de présentation de l’application (Menus, boutons, transitions entre les activités)
* Logique d’avancement au sein du jeu (Complétion de niveau et passage d’un à l’autre)
* Logique & construction d’une grille de jeu générique, ainsi que vérification de contraintes

Nous effectuerons donc la présentation technique du travail de manière à respecter cette division citée plus haut.

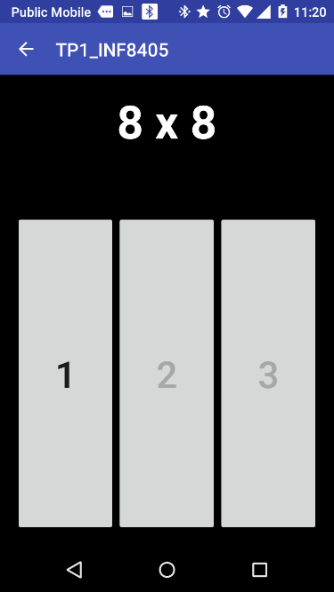
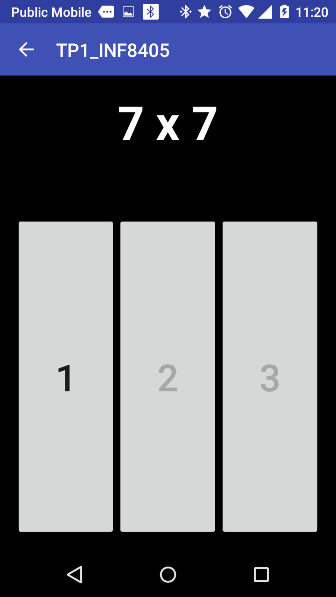
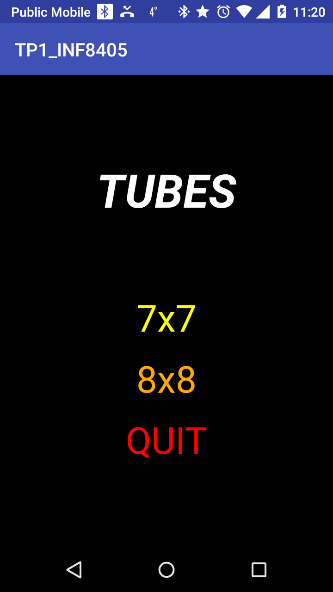
## Aspect de présentation de l’application

L’application présente trois pages principales qui sont les suivantes :

Figure 1: Main Menu

Figure 2: Levels in 7\*7

Figure 3: Levels in 8\*8



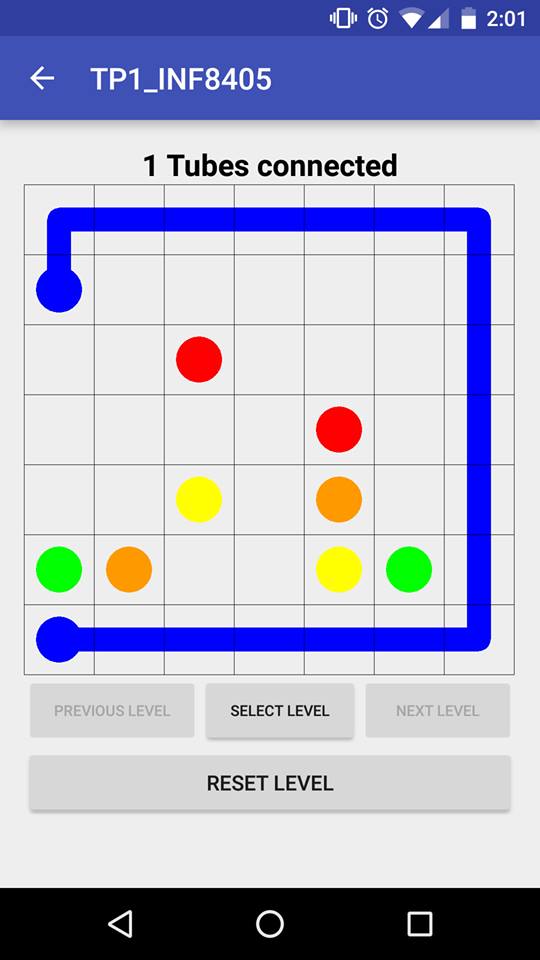


Figure 4 : Gaming

* Le menu principal :

Ce menu présente à l’utilisateur l’option de jouer sur une grille de taille 7x7 à l’aide d’un bouton ‘7x7’, 8x8 à l’aide d’un bouton ou de quitter l’application à l’aide du bouton ‘QUIT’. Le choix d’une taille de grille amène à la page de sélection de niveau pour la dimension de grille choisie tandis que le bouton ‘QUIT’ ferme simplement l’application.

* La page de choix de niveau :

Cette page présente à l’utilisateur la taille de la grille courante, ainsi que les différents niveaux associés à cette taille de grille. Chacun des niveaux est représenté par un bouton, qui est désactivé tant que le niveau n’a pas été débloqué. L’utilisateur peut donc choisir un niveau débloqué, ce qui l’enverra à la page de jeu, ou revenir en arrière au menu principal.

* La page de jeu :

Cette page contient l’activité principale qui est le jeu. L’utilisateur peut résoudre le puzzle du niveau choisi. Il peut recommencer le niveau courant en appuyant sur ‘Reset level’, naviguer entre les niveaux débloqués grâce aux boutons ‘Previous level’ et ‘NextLevel’. I peut aussi revenir à l’écran de sélection de niveau en appuyant sur le bouton ‘Select level’. Chacun de ces boutons crée une alerte à l’usager lui demandant de confirmer l’action s’il serait amené à perdre sa partie courante.

## Logique d’avancement au sein du jeu

Au début d’une partie, le joueur a accès aux niveaux 1 des grilles 7x7 et 8x8. Pour qu’une partie soie gagnée, il faut que toutes les cases de la grille soient utilisée et que tous les cercles soient légalement reliés. La partie est considérée comme perdue ou en cours tant que cette condition n’est pas atteinte. Lorsque cette condition est atteinte, le niveau suivant est débloqué s’il ne l’était pas, et le joueur peut maintenant y accéder soit n cliquant sur le bouton ‘Next level’ ou en allant dans le menu de sélection de niveaux. Les niveaux débloqués sont débloqués durant toute la session de jeu, mis perdu à la fermeture de l’application.

## Logique & construction de la grille de jeu

Dans cette troisième partie, nous aborderons l’aspect central de l’application et du travail. Il s’agit du développement de la grille de jeu en soi et des mécaniques de jeu. Débutons donc par l’architecture adoptée afin de réaliser la tâche de manière simple et efficace d’un point de vue de développement logiciel. Nous avons donc créé une mise en page de type *TableLayout* dans le fichier XML de l’activité de jeu nommé *GamingActivity*. Cette activité sera unique pour toutes les versions du jeu donc il faut faire en sorte de rendre la grille qu’elle contient générique et réutilisable pour les différents niveaux. Pour se faire nous n’avons pas spécifié les dimensions du *TableLayout* dans le XML directement. Nous avons plutôt décidé de les instancier au moment de la création de l’activité de jeu. Ceci nous permet donc de paramétrer le nombre de lignes et de colonnes par-rapport au niveau actuel, rendant l’activé de jeu totalement transparente de l’état actuel de l’application.

Une fois que les lignes et colonnes du *TableLayout* étaient créés, il nous fallait les remplir avec des composantes graphiques personnalisées. Une par cellule pour être exact. Nous avons défini une classe supplémentaire dans le projet. Celle-ci s’appelle *CellView* et représente, comme son nom l’indique, la vue d’une des cellules du plateau de jeu. Cette classe dérive donc de la superclasse *View*, nous permettant de surcharger la fonction *OnDraw* et de dessiner à l’intérieur de chacune des cellules, de manière indépendante des autres cellules du jeu. À partir de là, il ne suffisait plus que d’ajouter quelques champs privés ainsi que des accesseur et modificateurs afin d’indiquer à une cellule si elle représente un point d’entrée/sortie d’un tunnel, si elle doit se dessiner d’une certaine couleur, avec une certaine forme, afin qu’elle puisse appeler cette fonction surchargée et se dessiner par soi-même.

Maintenant que le concept de grille et de cellule sont définis, il ne restait plus qu’à décrire les écouteurs d’évènements requis, afin de répondre aux actions d’un joueur. Ceux-ci ont été ajoutés directement à la classe mentionnée plus tôt, *GamingActivity* qui contient le *TableLayout* ainsi que chacune des cellules de jeu. Cette dernière affirmation rend, selon nous, évident le fait que cette classe devrait-être celle qui gère les interactions entre le jeu et le joueur, puisqu’il s’agit de l’activité centrale des composants dynamiques de l’application. Donc, nous avons défini un *MotionListener* sur le *TableLayout* lui-même afin d’être en mesure de détecter les actions de l’utilisateur et de les refléter sur l’état de la grille. Pour ce faire, nous avons détecté, lorsque le joueur appuie sur la grille, la coordonnée (ligne, colonne) affectée par le clic afin de détecter la cellule actuellement tracée. Ensuite, sur l’action de *Drag* nous effectuons une certaine quantité de vérifications afin de déterminer si la cellule à le droit de se dessiner en fonction de l’état actuel de la grille et des autres cellules du jeu. En fonction de ces vérifications, on utilise les modificateurs de la cellule qui possède le focus afin de lui indiquer de se dessiner et selon quels paramètres le faire, à l’aide de ses champs de classe privés. Cette méthode est celle qui contient la majorité des mécaniques de jeu. Finalement, dans l’action qui détecte quand le joueur relâche son clic sur l’écran, nous effectuons les dernières vérifications simples afin de détecter si le tracé actuel se termine sur une cellule valide (par-rapport à la couleur du tunnel actuellement dessiné) qui se trouve à être une cellule d’entrée/sortie, ou bien si le tracé n’est pas valide il sera immédiatement effacé de la grille.

Pour ce qui est de la condition de victoire, l’architecture décrite dans cette section rend la tâche tout à fait simple. Il suffit de parcourir l’ensemble des cellules du jeu afin de vérifier si elles sont actuellement toutes utilisées. Cette condition très simple est suffisante, puisque de toute manière, tel que présenté dans le paragraphe précédent, les mécaniques de jeu ne permettront pas d’occuper une cellule de manière définitive à moins que l’ensemble des conditions de traçage soient respectées, autrement les cellules concernées sont automatiquement vidées et la victoire n’est pas possible, étant donné que toutes les cellules ne soient pas utilisées.

# Difficultés rencontrées

Au cours de la réalisation de ce travail, la totalité des difficultés furent rencontrées lors de la mise en place des mécaniques de jeu telles que décrite dans la section précédente. En effet, la méthode qui possède les différentes règles permettant de déterminer le comportement d’une cellule lors d’un *drag* de l’utilisateur est définitivement la méthode la plus imposante du projet d’un point de vue logique. Cette affirmation n’est pas surprenante étant donné qu’il s’agit d’une grosse partie du travail en soit, cependant on remarque quelques aspects qui auront été particulièrement ardus à implémenter. Le premier est le traçage des « coudes » dans les tunnels. En effet, ceux-ci ne sont pas facile à détecter lors du traçage. La solution adoptée a donc été d’ajouter deux champs privés à chaque instance de la classe *CellView*, il s’agit de la position du voisin précédent et du voisin suivant. Ceux-ci auront donc permis d’effectuer les vérifications nécessaires dans chacune de cellules, afin de déterminer si la cellule actuelle, par-rapport à la position de ses voisins, doit se tracer comme étant un « coude » et si oui, dans quelle orientation ceci devrait se faire. Finalement, la dernière difficulté notable se trouve au niveau de la fonctionnalité permettant se rétracter lors du traçage d’un tunnel. Celle-ci se trouve aussi dans l’évènement de *drag* de l’utilisateur et s’est régler en ajoutant une liste temporaire contenant les cellules actuellement comprises dans un tunnel temporaire, c’est-à-dire un tunnel incomplet/en cours de traçage. Cette liste temporaire a donc servi à effectuer un certain nombre de vérifications afin de détecter si l’utilisateur revenait sur ses pas. Dans quel cas, on retire des cellules du tunnel actuellement en cours de traçage.

# Critiques et suggestions

Afin d’améliorer l’application, il serait utile de sauvegarder la session de jeu à la fermeture de l’application pour ne pas la perdre et de pouvoir la poursuivre plus tard. De plus, il serait aussi intéressant de clarifier la notion de navigation entre les niveaux puisqu’elle est un peu ambiguë.

# Conclusion

Donc dans ce laboratoire nous avons vu comment les activités fonctionnent sous Android. Nous savons maintenant que comment nous pouvons créer une application simple de jeu dans Android. Aussi, nous possédons de meilleurs notions par-rapport aux fonctionnalités offertes par l’API Android et les possibilités que celle-ci permet d’implémenter.