Cours n°4 - Notions avancées sur les vues

Champs textes

Il est possible de configurer les champs textes d'une application afin d'améliorer l'expérience utilisateur. Les paramètres sont très variés comme la gestion des majuscules, la correction automatique des fautes d'orthographe ou le type de clavier proposé.

Comme d'habitude, il est possible d'intervenir à ce niveau en passant par l'interface des storyboards dans l'onglet « Inspecteur d'attributs » (diapositive 3) ou par du code (diapositive 5).

Il est également possible de laisser iOS faire une partie du travail automatiquement en spécifiant le « Content Type » d'un champ (par exemple, en précisant qu'il s'agit d'un champ attendant une adresse e-mail). (c.f. diapositive 4).

Il est à noter, pour la réalisation des tests, que par défaut, le clavier n'est pas affiché sur le simulateur car les champs peuvent être remplis avec le clavier de l'ordinateur. Il est possible de l'afficher au besoin. (c.f. diapositive 9)

Les vues personnalisées

Malgré la gamme très fournie de vues que propose le SDK, il est parfois nécessaire de définir ses propres vues pour intervenir à des endroits très spécifiques. Pour ça, il suffit de définir des classes qui héritent de la classe `UIView`. Il suffira ensuite de redéfinir certaines méthodes afin d'intervenir dans le cycle de vie de l'élément ou bien lorsque certains évènements se déclenchent (c.f. diapositives 11 et 12).

Cependant, il est préférable, autant que possible, d'éviter d'avoir recours à cette pratique si une alternative est disponible. (c.f. diapositive 13)

Chaîne d'évènements

Petit point à souligner concernant la propagation des évènements : lorsqu'une vue est contenue dans une autre, si un évènement se déclenche sur celle-ci, il se déclenche également sur la vue parente.

En diapositives 14 et 15, on en retrouve un exemple. Si seule la partie rouge (parent) est touchée, la partie jaune ne subit aucun changement. En revanche, au touché de la partie jaune, la partie rouge réagit également.

Gesture recognizers

Beaucoup de gestes sont très communs aux applications mobiles comme le swipe, ou le pincement. Pour fournir la reconnaissance de ces gestes facilement dans les applications iOS, le SDK propose des « gesture recognizers ». Une classe est fournie pour chaque geste (c.f. diapositive 17 et spécifiquement 18 pour la documentation sur ce qui est proposé).

Deux types de « recognizers » existent : les discrets et les continus. La différence réside dans l'appel à la méthode faisant office de gestionnaire d'évènement. Dans le cas d'un recognizer discret, la méthode ne sera appelé qu'une fois (e.g. tapement). Dans le cas d'un continu, la méthode sera appelée tout au long du geste (e.g. pincement).

Comme d'habitude, il est possible de les utiliser via du code (diapositive 19) ou les storyboards (diapositives 20 à 23).

Animations

Par soucis d'esthétisme, il est possible d'animer les vues dans une application. iOS est capable d'animer plusieurs propriétés par défaut ce qui permet de gérer des animations relativement classiques (c.f. diapositive 24).

Pour animer des éléments, il suffit d'utiliser la méthode `UIView.animate` et de lui passer une durée et une closure contenant le code qui représentera l'état dans lequel les éléments doivent se trouver une fois l'animation terminée (c.f. diapositive 25).

S'adapter aux terminaux

Sachant que les applications peuvent se retrouver sur des terminaux de configurations différentes, il est important de s'assurer que l'interface va correctement s'adapter selon la configuration. Plusieurs solutions sont offertes pour gérer cette adaptation.

Les classes de tailles permettent de faire varier la valeur de certaines propriétés selon l'appareil et son orientation.

Auto-layout est la solution la plus poussée et sans doute la plus répandue, permettant d'ajouter des contraintes sur les éléments pour définir la façon dont ils doivent s'adapter.

Auto-resizing est une solution relativement ancienne qui permet de définir la taille ou la position selon la configuration de l'écran. Il ne sera pas utilisé dans le cadre de ce cours.

Finalement, il est possible de passer par du code pour configurer l'apparence de nos éléments par rapport à la configuration de l'écran même si ce sera en dernier recours.

Classes de tailles

Les classes de tailles vont être utilisées pour faire varier la valeur de certaines propriétés selon l'appareil. Il en existe deux : « Compact » et « Regular » (c.f. diapositive 28). Ces classes sont associées à la largeur ou la hauteur de l'écran selon l'appareil et si ce dernier se trouve en mode portrait ou paysage. (c.f. diapositive 29)

Pour rappel, il n'est pas nécessaire de connaître par coeur le tableau se trouvant diapositive 29.

Il est possible de définir une valeur de propriété selon les classes de tailles dans l'interface des storyboards, au niveau du panneau droit. Les propriétés qu'il est possible de faire varier ont un signe « + » se trouvant à leur gauche (comme par exemple en diapositive 30).

Auto-layout

L'auto-layout consiste à appliquer des contraintes aux éléments de l'interface pour qu'ils s'adaptent correctement quelque soit la configuration. On va généralement utiliser le moins de valeurs absolues (c'est à dire des valeurs en points ou pixels) possibles et préférer exprimer les choses en pourcentage par exemple.

Un élément prit en charge implicitement est l'adaptation selon la langue de l'utilisateur. Le concept de gauche et droit n'existe pas vraiment ; la terminologie utilise plutôt « leading » et « trailing » pour désigner respectivement le début et la fin de l'écran qui ne sont pas les mêmes selon la langue de l'utilisateur (à gauche pour le début dans les langues latines, à droite dans la plupart des langues sémitiques par exemple). (c.f. diapositive 32)

Les contraintes appliquées aux éléments peuvent être vues comme des équations auxquelles l'élément doit se conformer. L'ensemble de ses contraintes constitue une équation plus complexe ou chaque membre est pondéré par une priorité (variant entre 250 et 1000). (c.f. diapositive 33)

Dans Xcode, les éléments permettant de gérer l'auto-layout se trouvent dans la partie supérieure droite de la barre en bas de l'interface des storyboards (diapositives 34 et 35) ainsi que dans l'inspecteur de tailles, situé dans le panneau droit (diapositive 42 à 44).

Parfois, il se peut que des contraintes aient été appliquées à un élément mais que ceci ne se reflète pas sur l'affichage de la scène dans Xcode. Dans ce cas, il suffit de faire appel au bouton permettant de mettre à jour les frames (c.f. diapositive 46).

Un des problèmes majeurs de l'auto-layout est la présence de conflits entre contraintes. Dans ce cas, un indicateur rouge se trouve dans le panneau gauche du storyboard ; le clique sur cet indicateur peut afficher plus d'informations et d'éventuelles solutions (c.f. diapositives 47 et 48)

Il existe un bouton permettant de résoudre automatiquement ces problèmes mais il peut s'avérer totalement contre-productif en supprimant des contraintes de manière un peu trop « agressive ». (c.f. diapositive 49 et 50)

Comme d'habitude, il est possible de passer par du code pour définir les contraintes sur les vues mais il existe plusieurs façons d'y procéder et il s'agit globalement d'une pratique peut répandue. (c.f. diapositive 51)

Imbrication des éléments

Il est possible d'imbriquer les vues dans un conteneur parent. Plusieurs conteneurs sont disponibles selon le besoin. (c.f. diapositive 52)

Les scroll-views permettent de s'assurer que l'utilisateur puisse descendre en scrollant si le contenu dépasse la taille de l'écran. (c.f. diapositive 53)

Les stack-views permettent d'empiler des éléments et de les contraindre pour qu'ils soient disposés comme souhaité. Cette vue s'apparente sur beaucoup de points à la fonctionnalité flexbox en CSS. Il est possible de faire varier la direction de l'axe dans lesquels les éléments s'empilent, l'espacement entre ces derniers, etc. (c.f. diapositives 54 à 56).

Pour rappel, il n'est pas nécessaire de connaître par coeur les diapositives 55 et 56 ni le fonctionnement des propriétés d'une stack-view.

Finalement, deux cas particuliers de contrôleurs permettent d'imbriquer des vues dans des conteneurs « prêts à l'emploi » pour designer facilement une application.

Le navigation controller permet de gérer une navigation entre scènes en gérant une pile de scènes, proposant ainsi un bouton fournissant le retour en arrière.

Le « tab bar controller » permet de proposer une barre d'icônes en bas de l'écran pour naviguer également entre différentes scènes.