Détection de gestes

Détecter des gestes dans une application iOS, tels que des taps, des pincements ou des rotations est extrêmement facile avec Swift et les classes `UIGestureRecognizer` intégrées à Xcode.

Dans ce tutoriel/TP, vous apprendrez comment ajouter facilement des outils de reconnaissance des gestes dans votre application, à partir du Storyboard de Xcode. Pour cela, vous allez créer une application simple dans laquelle vous pouvez déplacer un chat et une bouteille de lait en les faisant glisser, en les pinçant et en les faisant pivoter à l'aide des outils de reconnaissance de gestes. Vous verrez également comment ajouter de la décélération à un mouvement et reconnaitre plusieurs gestes de façon simultanée. Finalement, dans une dernière partie vous implémenterez la detection d'un geste de type `Swipe`.

Partie I : Présentation des classes UIGestureRecognizer

Avant le développement des classes `UIGestureRecognizer`, si vous vouliez détecter un geste tel qu'un glissement, vous deviez vous enregistrer pour recevoir des notifications de la part d'Apple à chaque touché dans un objet de type `UIView` - comme `touchesBegan`, `touchesMoved`, et `touchEnded`. Chaque programmeur écrivait un code légèrement différent pour détecter les touchés, ce qui entraînait des bugs subtils et des incohérences entre applications.

Depuis iOS 3.0, Apple a proposé les classes `UIGestureRecognizer` qui fournissent une implémentation par défaut de la détection des gestes communs tels que les pincements, les rotations, les balayages, les pressions longues, etc. En les utilisant, non seulement cela vous

permet d'économiser l'écriture de code, mais cela permet également à vos applications de fonctionner correctement ! Bien sûr, vous pouvez toujours utiliser les anciennes notifications tactiles, si votre application en a besoin.

Les classes `UIGestureRecognizer` sont extrêmement simples à utiliser. Il suffit d'effectuer les deux étapes suivantes:

1. **Créez un outil de reconnaissance de gestes**. Lorsque vous créez un outil de reconnaissance des gestes, vous spécifiez une fonction de rappel (callback)[1] afin que le module de reconnaissance des gestes puisse vous envoyer des mises à jour lorsque le mouvement commence, change ou se termine.
2. **Ajoutez le module de reconnaissance des gestes à une vue**. Chaque reconnaissance de gestes est associée à une (et une seule) vue. Lorsqu'un contact se produit dans les limites de cette vue, l'outil de reconnaissance des gestes vérifie s'il correspond au type de toucher recherché et si une correspondance est trouvée, il notifie la fonction de rappel.

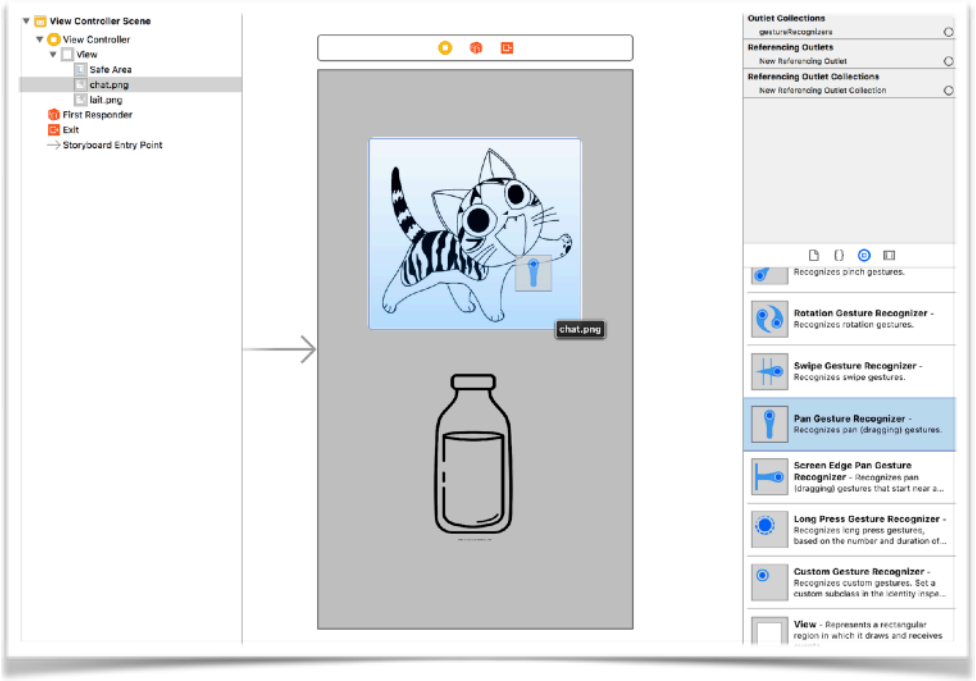
Vous pouvez effectuer ces deux étapes par programmation, mais il est encore plus facile d'ajouter visuellement un outil de reconnaissance de gestes directement dans votre fichier `main.Storyboard`.

UIPan GestureRecognizer

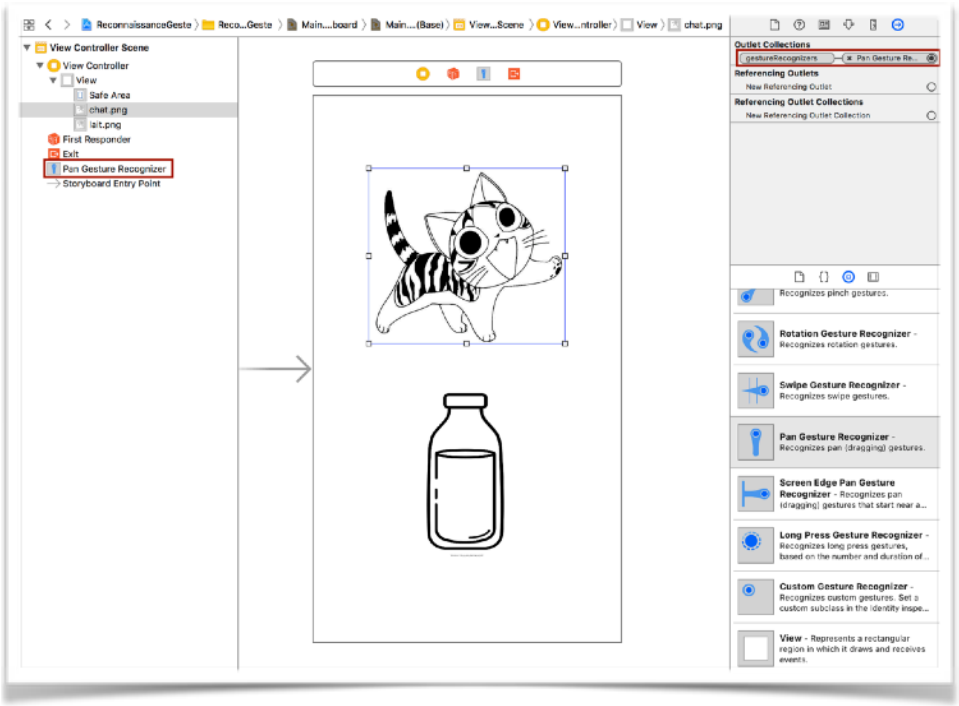
Afin de testez l'utilisation des classes `UIGestureRecognizer`, vous allez commencer par tester la reconnaissance d'un geste de type `UIPan`.

Télécharger le projet `ReconnaissanceGeste` sur Moodle et ouvrez le fichier `Main.storyboard`. Dans la bibliothèque d'objets, recherchez l'objet `Pan Gesture Recognizer`. Faites ensuite glisser l'objet `Pan Gesture Recognizer` sur l'objet `UIImageView` contenant l'image du chat. Cela crée à la fois le système de reconnaissance du mouvement de type Pan et l'associe à l'objet `UIImageView` du chat.

[1] une **fonction de rappel** (*callback* en anglais) ou fonction de **post-traitement** est une fonction qui est passée en argument à une autre fonction

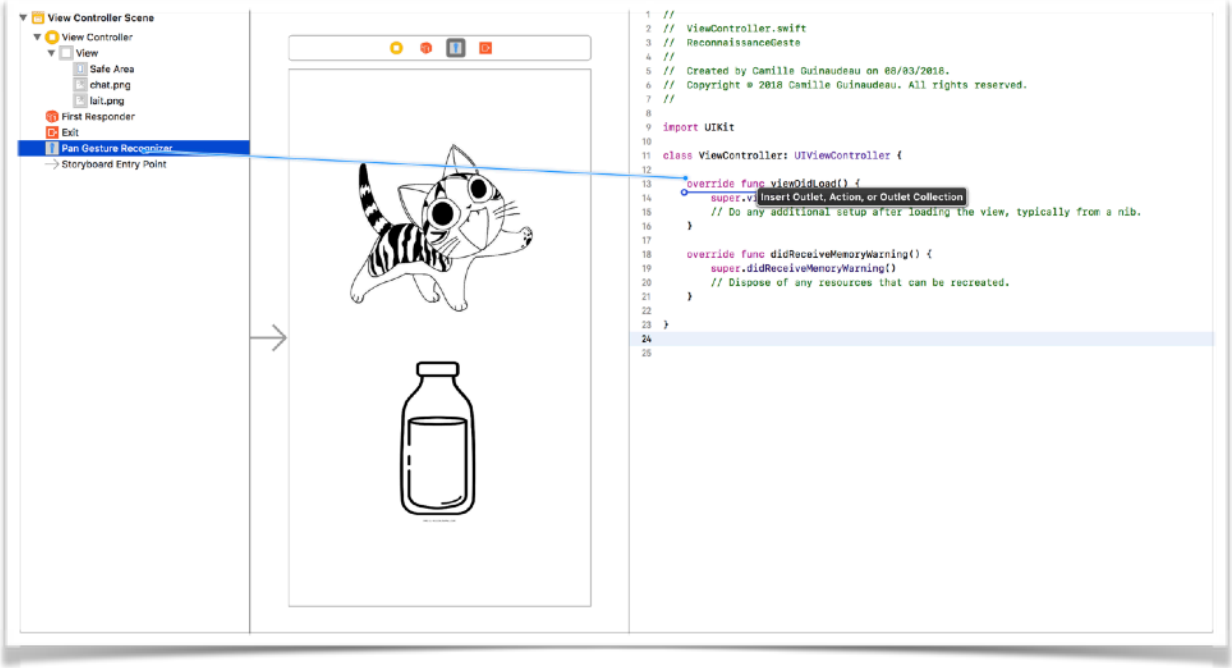


Vous pouvez vérifier que vous avez bien connecté votre objet de type `UIPanGestureRecognizer` en regardant l'inspecteur des connexions et en vous assurant que la reconnaissance de gestes est dans la liste des « « Outlet Collections ». Il doit également apparaitre dans la liste des objets existant dans la vue (partie de gauche de l'image).

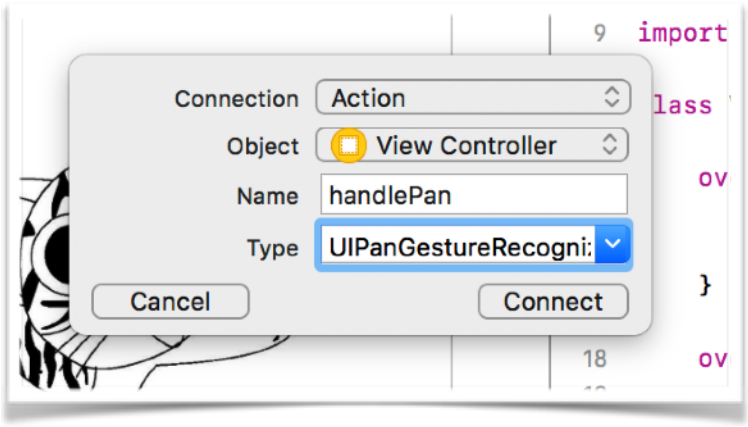


Maintenant que vous avez créé le système de reconnaissance du mouvement Pan et que vous l'avez associé à l'objet `UIImageView`, vous devez écrire la fonction de rappel afin que quelque chose se passe réellement lorsque le geste se produit.

Ouvrez l'assistant d 'édition pour mettre en parallèle le fichier `Main.Storyboard` et la fichier `ViewController.swift`. Créez une connection de type IBAction à partir de votre objet de type `UIPanGestureRecognizer`:



Appelez cette IBAction `handlePan` et faites en sorte que son type soit « `UIPanGestureRecognizer` » et non « any ».



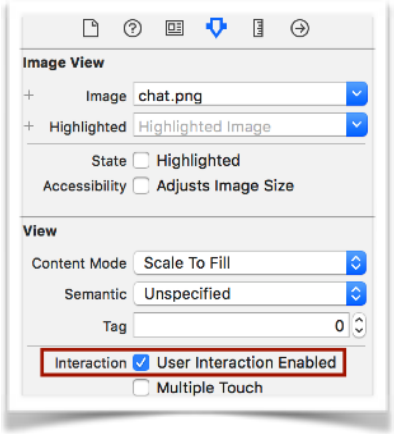
Insérez les instructions suivantes dans le corps de votre fonction `handlePan` :

```
let translation = sender.translation(in: self.view)
if let view = sender.view {
    view.center = CGPoint(x:view.center.x + translation.x,
                          y:view.center.y + translation.y)
}
sender.setTranslation(CGPoint.zero, in: self.view)
```

Votre objet `UIPanGestureRecognizer` appelle cette fonction lorsqu'un geste de type Pan est détecté, puis continuellement pendant que l'utilisateur continue de faire le geste, et une dernière fois lorsque le geste se termine (c'est-à-dire lorsque l'utilisateur soulève son doigt).

`UIPanGestureRecognizer` étant un argument de cette fonction, vous pouvez récupérer lea distance sur laquelle l'utilisateur a déplacé son doigt en appelant la fonction `translation(in:)`. Ici, on utilise cette distance pour déplacer le centre de l'image du chat. Il est important de remettre la valeur de la translation à zéro une fois que le geste est terminé. Sinon, les distances de translations continueront à s'additionner les unes au autres à chaque fois que le geste est effectué, et vous verrez le chat sortir rapidement de l'écran ! Notez qu'au lieu de coder en dur l'objet de type `UIImageView` dans cette fonction, nous utilisons une référence à cet objet en utilisant l'instruction `sender.view` (c'est-à-dire la vue contenant l'objet de reconnaissance de geste passé en paramètre). Cela rend votre code plus générique, de sorte que vous pouvez réutiliser cette même fonction pour la vue contenant l'image de la bouteille de lait.

Si vous compilez et exécutez votre application, et essayez de faire glisser le chat, cela ne fonctionnera pas encore. La raison en est que la détection de gestes est désactivée par défaut pour les vues qui n'acceptent normalement pas les gestes, comme les objets de type `UIImageView`. Pour corriger cela, sélectionnez votre objet de type `UIImageView` contenant l'image du chat, ouvrez l'inspecteur d'attributs et cochez la case « User Interaction Enabled ».



Compilez et exécutez encore, et cette fois vous devriez être capable de faire glisser le chat dans l'écran de votre application.

Notez que vous ne pouvez pas faire glisser la bouteille de lait. C'est parce que les objets de reconnaissance de gestes doivent être liés à une (et une seule) vue. Cependant, si vous ajoutez un objet de type `UIPanGestureRecognizer` à l'objet de type `UIImageView` contenant la bouteille de lait et que vous reliez ce système de detection de geste à la méthode `handlePan` deja écrite, vous pouvez également déplacer votre bouteille de lait (veillez toutefois que la case « User Interaction Enabled » soit cochée également pour cet élément).

Partie II : Décélération de mouvement

Dans beaucoup d'applications Apple, lorsque l'utilisateur stoppe son geste, il y a une petite décélération à la fin du mouvement. Pensez au défilement d'une page Web, par exemple. Il est courant de vouloir avoir ce type de comportement dans vos applications.

Il y a plusieurs façons de mettre en place cette décélération, je vais vous présenter ici une implémentation très simple. L'idée est de détecter quand le geste se termine, comprendre à quelle vitesse le doigt se déplaçait, et animer l'objet se déplaçant vers une destination finale basée sur la vitesse de contact.

1. **Pour détecter quand le geste se termine :** La fonction de rappel associée à l'outil de reconnaissance de gestes est appelée à chaque fois que l'outil de reconnaissance de gestes change d'état, c'est-à-dire lorsque le geste commence, lorsque le geste change ou lorsque le geste se termine. Vous pouvez connaitre l'état dans lequel se trouve le système de reconnaissance de gestes simplement en regardant sa propriété `state`.
2. **Pour détecter la vitesse de déplacement du doigt:** Certains outils de reconnaissance de gestes renvoient des informations supplémentaires (reportez vous au guide API pour plus de détails). L'objet UIPanGestureRecognizer dispose d'une fonction `velocity(in:)` permettant de récupérer les informations liées à la vitesse de déplacement du doigt.

Ajoutez les instructions suivantes au bas de la fonction `handlePan(sender:)` dans le fichier `ViewController.swift` :

```
if sender.state == UIGestureRecognizerState.ended {
    // 1
    let velocity = sender.velocity(in: self.view)
    let magnitude = sqrt((velocity.x * velocity.x) +
                         (velocity.y * velocity.y))
    let slideMultiplier = magnitude / 200

    // 2
    let slideFactor = 0.1 * slideMultiplier

    // 3
    var finalPoint = CGPoint(x:sender.view!.center.x +
                             (velocity.x * slideFactor),
                             y:sender.view!.center.y +
                             (velocity.y * slideFactor))

    // 4
    finalPoint.x = min(max(finalPoint.x, 0),
                                         self.view.bounds.size.width)
    finalPoint.y = min(max(finalPoint.y, 0),
                                        self.view.bounds.size.height)

    // 5
    UIView.animate(withDuration: Double(slideFactor * 2), delay: 0,
                // 6
                options: UIViewAnimationOptions.curveEaseOut,
                animations: {sender.view!.center = finalPoint },
                completion: nil)
}
```

Cette fonction de décélération simple utilise la stratégie suivante:

1. Déterminez la longueur du vecteur de vitesse (c'est-à-dire la magnitude)
2. Si la longueur est <200, diminuez la vitesse de base, sinon augmentez-la.
3. Calculer un point final basé sur la vitesse de déplacement du doigt et le facteur de glissement.
4. Assurez-vous que le point final est dans les limites de la vue
5. Animez la vue jusqu'au lieu d'arrêt final.
6. Utilisez l'option d'animation `curveEaseOut` pour ralentir le mouvement au fil du temps.

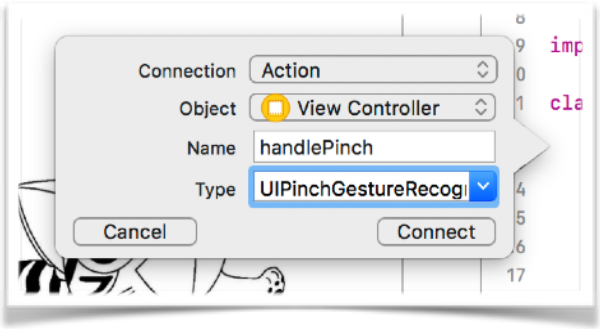
Compilez et exécutez votre application, vous devriez maintenant avoir une décélération basique de vos objets lorsque vous levez votre doigt.

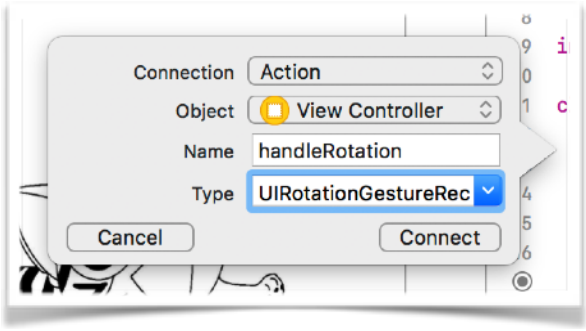
Partie III : Pincement et gestes de rotation

Dans cette troisième partie, nous verrons comment redimensionner un objet et le faire pivoter en utilisant des gestes de pincement et de rotation.

Sur le modèle de ce qui vous a été présenté dans la partie I, ajoutez des objets de reconnaissance de gestes de type `UIPinchGestureRecognizer` et `UIRotationGestureRecognizer` à l'objet de type `UIImageView` contenant l'image de chat.

Ouvrez l'assistant d'édition pour créer deux IBActions `handlePinch` et `handleRotation` dont les types sont respectivement `UIPinchGestureRecognizer` et `UIRotationGestureRecognizer` à partir des gestes associés à votre image de chat.





Ouvrez le fichier `ViewController.swift` et ajoutez ce qui suit à la méthode `handlePinch(sender:)`

```
if let view = sender.view {
    view.transform = view.transform.scaledBy(x: sender.scale,
                                             y: sender.scale)
    sender.scale = 1
}
```

Ensuite, ajoutez ce qui suit à `handleRotation(sender:)`:

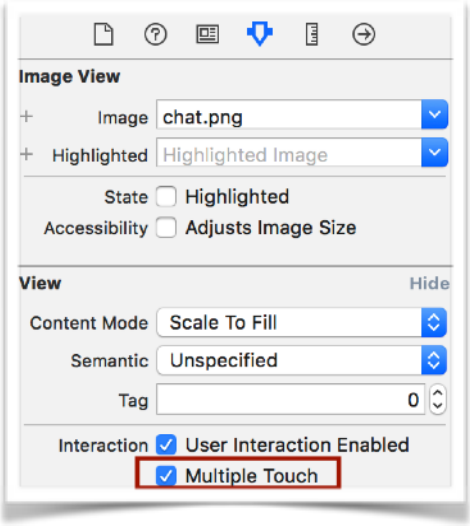
```
if let view = sender.view {
  view.transform = view.transform.rotated(by: sender.rotation)
  sender.rotation = 0
}
```

Comme avec l'objet de detection de geste Pan (`UIPanGestureRecognizer`) grâce auquel vous pouvez effectuer une translation, vous pouvez modifier l'échelle d'un objet ou effectuer sa rotation à partir des objets de détection de gestes `UIPinchGestureRecognizer` et de `UIRotationGestureRecognizer`.

Chaque vue possède une propriété `transform` qui précise la transformation (translation, rotation, changement d'échelle) appliquée sur cette vue. Apple propose un certain nombre de fonctions intégrées pour rendre le travail de transformation facile, comme la méthode `scaledBy(x:,y:)` permettant une mise à l'échelle ou la méthode `rotated(by:)` permettant d'effectuer une rotation. Ce sont les deux méthodes utilisées dans les exemples précédents.

Par ailleurs, comme précédemment, puisque vous mettez à jour la vue chaque fois que le geste se met à jour, il est très important de réinitialiser l'échelle et la valeur de rotation à leur valeur par défaut afin de ne pas avoir de comportement aberrant.

Vérifiez que la case « Multiple Touch » est cochée pour l'objet de type `UIImageView` contenant une image de chat.



Vous pouvez maintenant compiler et tester votre application.
Dans le simulateur, maintenez la touche d'option enfoncée et faites-glisser votre souris pour simuler deux doigts, et maintenez la touche Maj et l'option enfoncées simultanément pour déplacer les doigts simulés dans une position différente[2].

[2] Il est plus facile de simuler des gestes nécessitant plusieurs doigts directement dans des appareils de test. Vous aurez l'occasion de le faire lors de votre prochain TP.

Effectuez les manipulations nécessaires (ajout des objets de reconnaissance de gestes, création des liens avec les méthodes `handlePinch` et `handleRotation` et selection de la case « Multiple Touch ») pour que les gestes de pincement et de rotation soient détectés également sur l'objet de type `UIImageView` contenant la bouteille de lait.

Partie IV : Reconnaissance de gestes simultanés

En testant votre application, vous pouvez remarquer que, si vous mettez un doigt sur le chat et un doigt sur la bouteille de lait, vous pouvez les faire glisser en même temps. Cependant, vous remarquerez que si vous essayez de faire glisser le chat et que vous le faites glisser au milieu d'un deuxième doigt pour tenter de le pincer, cela ne fonctionnera pas. Par défaut, une fois qu'un objet de reconnaissance de gestes sur une vue «réclame» le geste, personne d'autre ne peut reconnaître un geste à partir de ce point. Un seul geste peut donc être reconnu à la fois pour un objet donné.

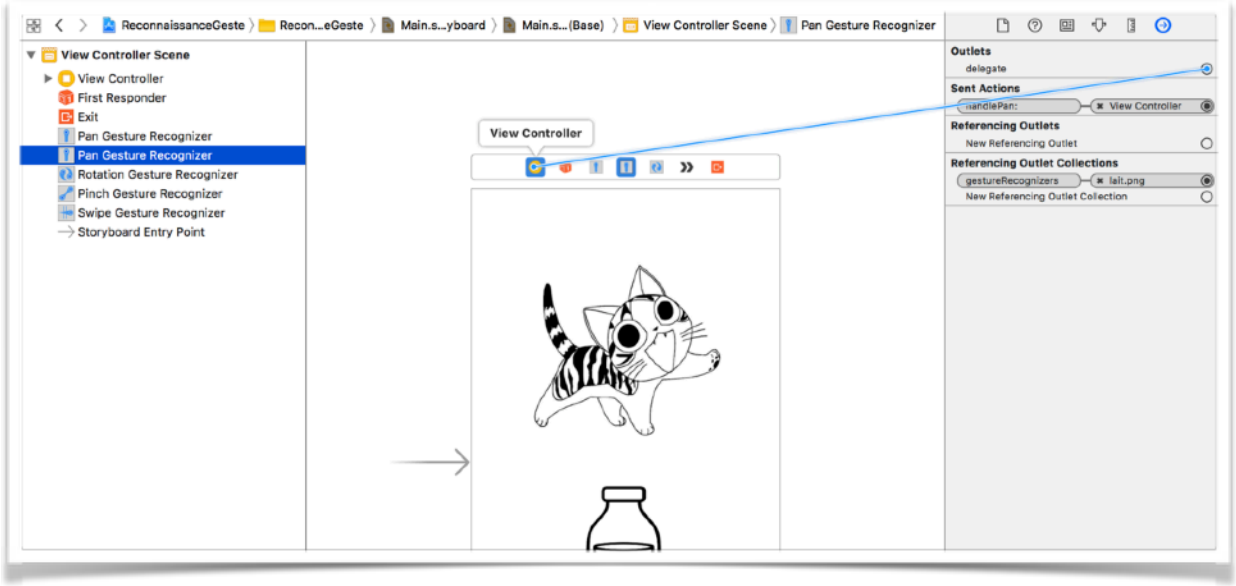
Il est cependant possible de modifier cela en utilisant le protocole `UIGestureRecognizerDelegate`.

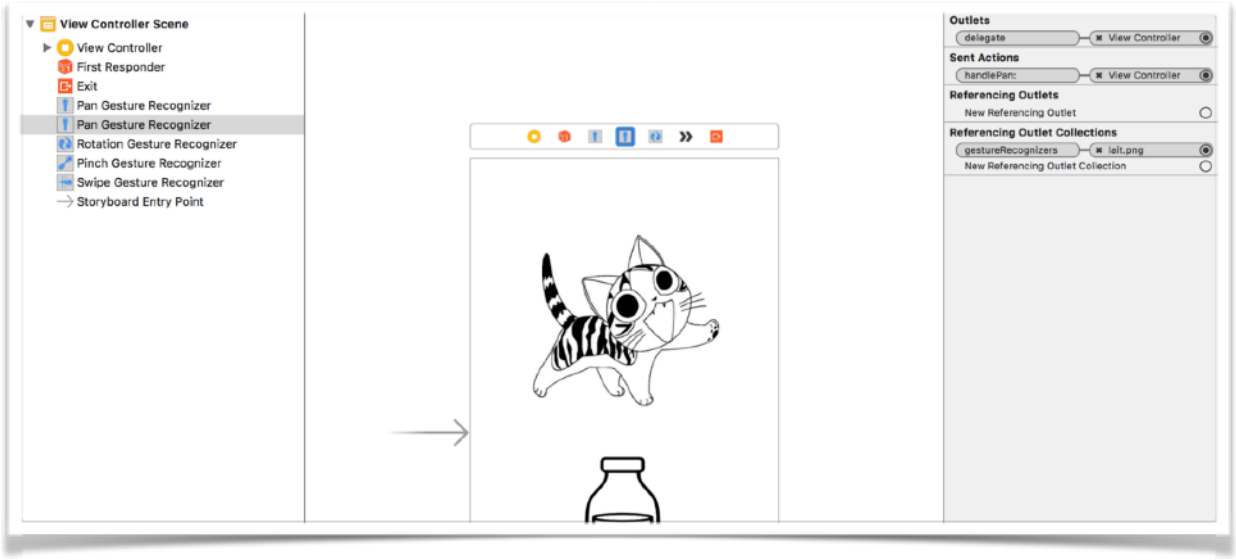
Ouvrez votre fichier `ViewController.swift`. et faire en sorte qu'il se conforme au protocole `UIGestureRecognizerDelegate` et implémentez ensuite l'une des fonctions optionnelles du délégué:

```
func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
          shouldRecognizeSimultaneouslyWith
          otherGestureRecognizer: UIGestureRecognizer) -> Bool {
    if gestureRecognizer is UIPanGestureRecognizer ||
       gestureRecognizer is UIPinchGestureRecognizer {
       return true
    } else {
       return false
    }
}
```

Cette fonction indique à l'outil de reconnaissance de gestes s'il est correct de reconnaître un autre geste (paramètre `otherGestureRecognizer`) si un autre dispositif de reconnaissance de geste donné (paramètre `gestureRecognizer`) a déjà détecté un geste. L'implémentation par défaut renvoie toujours `false`. Dans cet exemple, l'implementation renverra `true` si les deux gestes concernés sont un geste de changement d'échelle (`Pinch`) et un geste `Pan`.

Ensuite, ouvrez votre fichier `Main.storyboard`, et pour chaque objet de reconnaissance de gestes, connectez sa sortie de délégué au contrôleur de vue.





Compilez et lancez l'application, vous êtes maintenant capable de faire glisser le chat, le pincer pour l'agrandir, et continuer à le faire glisser après.

Partie V : Implementer la detection d'un geste swipe

Dans cette dernière partie du tutoriel, vous allez gérer la reconnaissance d'un geste de type Swipe. Lors de la reconnaissance d'un geste de type Swipe dont la direction est vers la droite, on souhaite qu'un objet de type `UIAlertController` s'affiche à l'écran.

Pour cela, dans votre application `ReconnaissanceGeste` :

- ajoutez un objet de type `Swipe Gesture Recognizer` à la `UIView` de votre interface graphique,
- créez la méthode `handleSwipe` correspondante,
- implémentez la méthode `handleSwipe` de façon à ce qu'un objet de type `UIAlertController` soit créé et s'affiche à l'écran lorsque la direction du geste (attribut `direction`) est égale à `.right`.

Compilez et testez votre application.

Sources :

- https://www.raywenderlich.com/162745/uigesturerecognizer-tutorial-getting-started

Partie VI : Détection de gestes et multimédia

Partie VI.1 : Détection de gestes

Question 1 : Créez un nouveau projet de type « single view ».

Question 2 : Ajoutez un objet de type `UIView` à votre storyboard. Créez le IBOutlet correspondant et initialisez sa couleur de fond avec la couleur de votre choix dans la méthode `viewDidLoad`.

Question 3 : Ajoutez un geste « tap » à votre objet de type `UIView`. Ajoutez les IBOutlet et IBAction correspondants. Paramétrez le geste dans la méthode `viewDidLoad` pour préciser le nombre de tap nécessaires à se détection `(numberOfTapsRequired)` ainsi que le nombre de doigts `(numberOfTouchesRequired)`. Un geste « tap » doit être détecté lorsque l'utilisateur tape successivement l'écran 2 fois de suite avec un doigt. Implémentez l'IBAction afin que la couleur de fond de `UIView` change lorsque le geste est détecté.

L'instruction suivante vous permet de créer un objet de type CGFloat d'une valeur aléatoire comprise entre 0 et 1 :

```
let b = CGFloat(arc4random() % 255) / 255.0
```

Vous pouvez ensuite créer une couleur à partir de trois composantes RGB grâce à l'instruction suivante (avec `r`, `g` et `b` compris entre 0 et 1). Le paramètre `alpha` correspond à l'information d'opacité de la couleur et est également compris entre 0 et 1:

```
let c = UIColor(red: r, green: g, blue: b, alpha: 1.0)
```

Question 4 : Lancez votre application sur un iPad et testez la détection du geste.

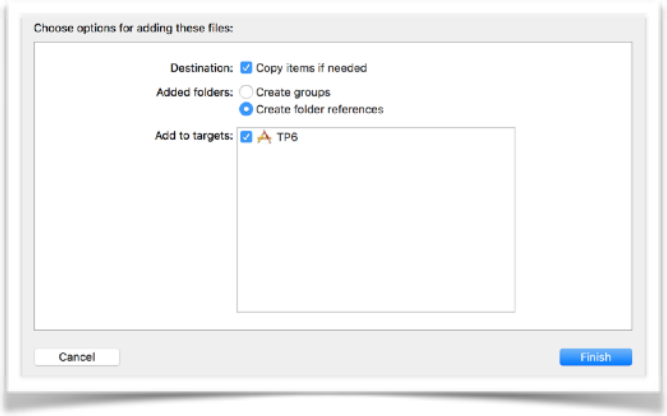
Question 5 : Ajoutez un geste « long press » à votre objet de type `UIView`. Ajoutez les IBOutlet et IBAction correspondants. Paramétrez le geste dans la méthode `viewDidLoad` afin que le geste soit détecté lorsque l'utilisateur appuie sur l'écran avec 2 doigts pendant au moins une seconde (attribut `minimumPressDuration`).

Question 6 : Implémentez l'IBAction associée au geste. Elle doit permettre de redonner à votre `UIView` sa couleur initiale.

Partie VI.2 : Lecture d'un fichier vidéo

Question 5 : Ajoutez un bouton « lire une video » à votre interface graphique ainsi que l'IBAction correspondante.

Question 6 : Télécharger le fichier « video.mp4 » via Moodle et ajoutez la dans le dossier de votre application. Vérifier que le nom de votre projet dans le menu « Add to targets » est bien coché.



Question 7 : Implémenter la méthode permettant de lancer la lecture de la vidéo. Testez votre application.

Partie VI.3 : Prise de photos

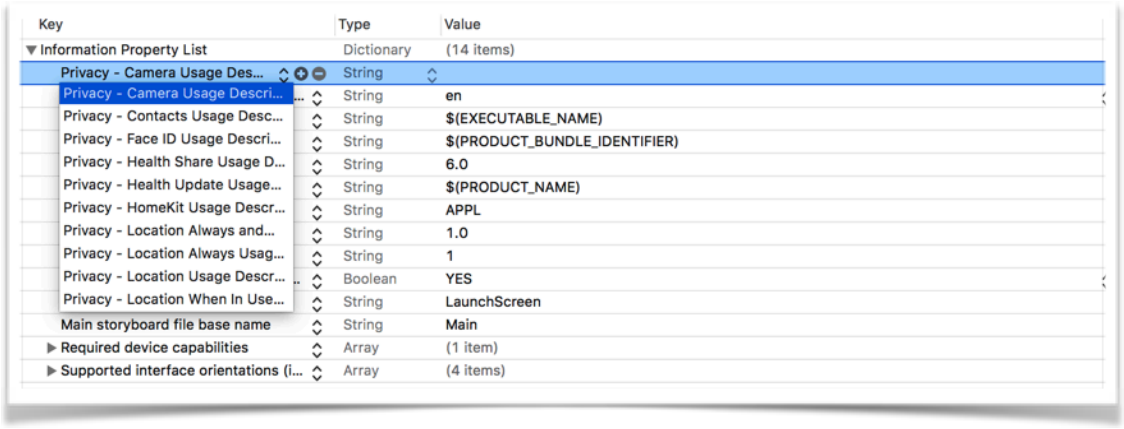
Question 8 : Ajoutez un bouton « prendre une photo » à votre interface graphique, et créez l'IBAction correspondante.

Question 9 : Implémenter la méthode permettant de paramétrer et de lancer la prise de vue.

Depuis iOS 10.0, vous avez besoin de préciser l'usage des photos prises dans votre application dans le fichier Info.plist.

Pour se faire, modifier le fichier Info.plist pour ajouter le champ « Privacy - Camera Usage Description » en appuyant sur le bouton + associé au dictionnaire à la racine de votre fichier

puis en sélectionnant le champ « Privacy - Camera Usage Description » dans la liste déroulante. Précisez ensuite l'usage fait des photos dans votre application (par exemple « photo use »).



Question 10 : Ajoutez un objet de type `UIImageView` dans votre interface graphique et créez le IBOutlet correspondant.

Question 11 : Implémenter les méthodes du protocole `UIImagePickerControllerDelegate` qui permettent d'annuler (`imagePickerControllerDidCancel:`) ou de traiter une prise de vue (`imagePickerController:didFinishPickingMediaWithInfo:`). Affichez la photo prise dans votre objet de type `UIImageView`.

Question 12 : Testez votre application.