Cours n°9 du 8 novembre 2012

3MCIn1 2012-2013

Protocols et délégation

Complément d'informations, éclaircissements

Pour aller à l'essentiel...

Un protocole, c'est:

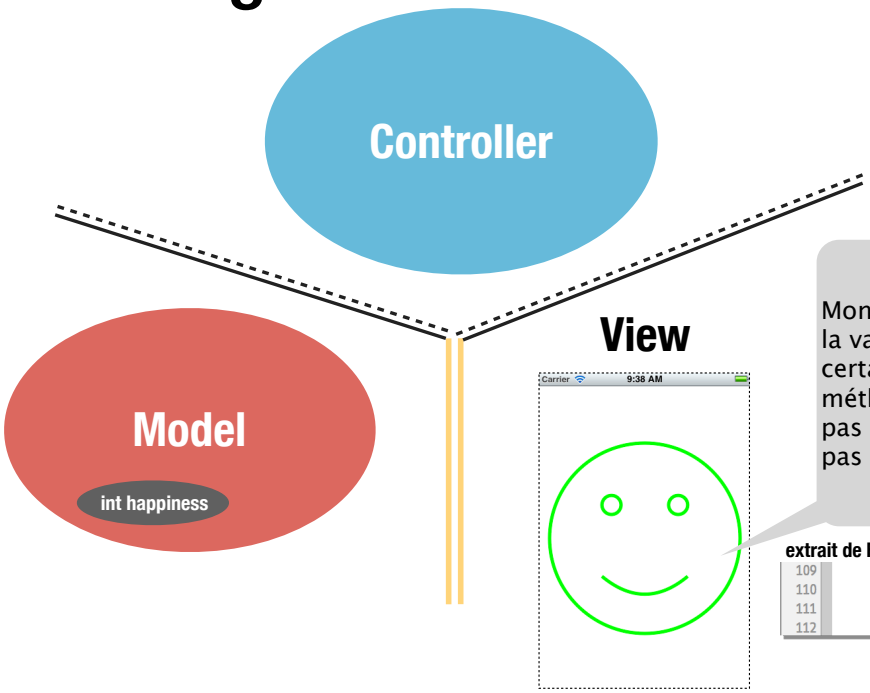
- une liste de déclaration de méthodes
- un langage commun qu'une collection d'objets acceptent de parler

Un protocole permet de communiquer de manière flexible avec un objet sans du tout connaître son type ou sa classe. Tout ce que l'on sait, c'est que l'objet implémente un certain protocole.

Protocols

Délégation, dataSource

La délégation: 1

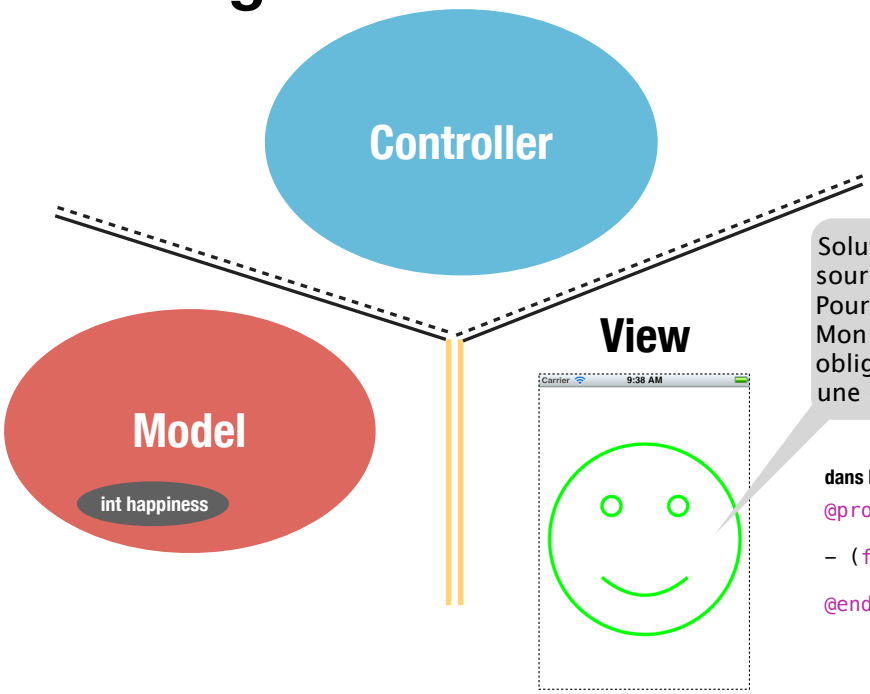


Mon sourire est figé. Ma «quantité de sourire», la variable `float smile`, est fixée à une certaine valeur dans la description de la méthode drawRect qui me dessine. Elle n'est pas liée à mon modèle et ne représente donc pas mon «bonheur» (int happiness)

extrait de FaceView.m

```
109
110         // on veut que smile soit fixé entre -1.0 et 1.0
111     float smile = 1;
112
```


La délégation: 2



Solution: je vais déléguer ma «quantité de sourire» à qui voudra bien la fixer à ma place. Pour ce faire, je vais utiliser un protocole. Mon protocole contient une méthode obligatoire, smileForFaceView:, qui retourne une «quantité de sourire».

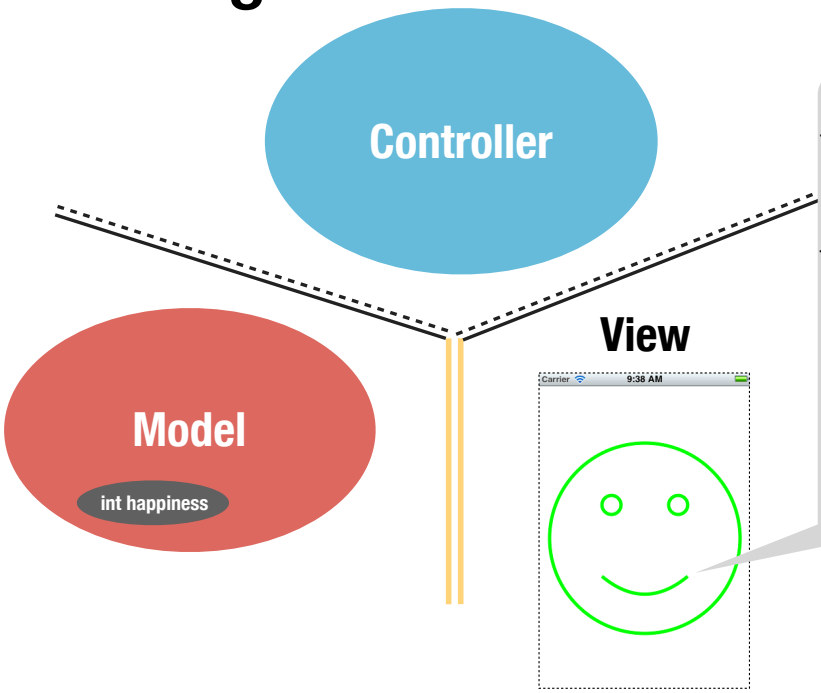
dans FaceView.h

```
@protocol FaceViewDataSource

- (float)smileForFaceView:(FaceView *)sender;

@end
```

La délégation: 3



Pour pouvoir communiquer avec celui à qui je vais déléguer ma «quantité de sourire», je vais ajouter une propriété à ma classe qui est un pointeur vers ce délégué.

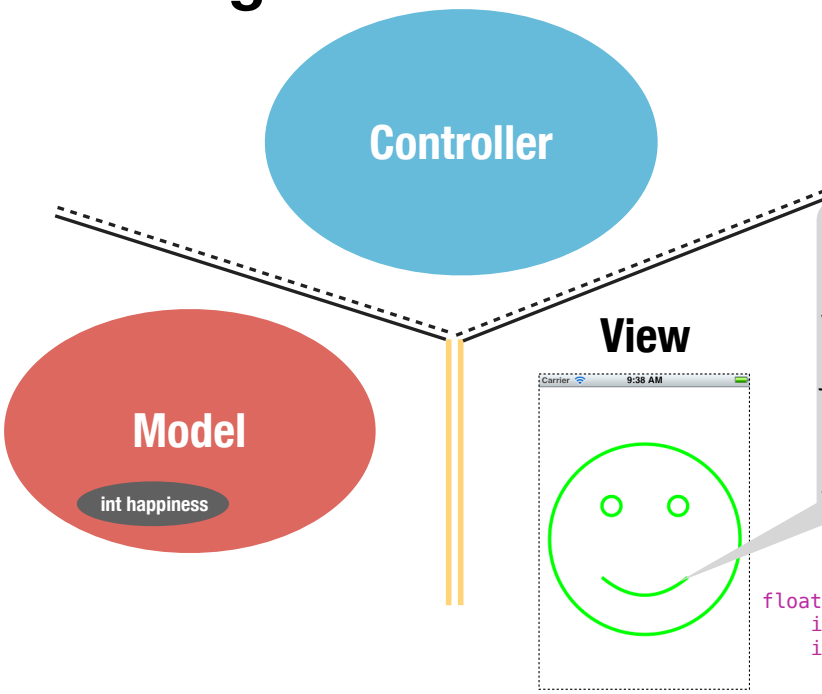
Je n'ai pas à connaître la classe de l'objet à qui je délègue ma «quantité de sourire», la seule chose qui m'importe c'est qu'il implémente mon protocole FaceViewDataSource.

La propriété que je m'ajoute est donc de type `id <FaceViewDataSource>`, c'est-à-dire un pointeur vers un objet de n'importe quelle classe qui implémente le protocole FaceViewDataSource.

dans FaceView.h

```
@property (weak, nonatomic) IBOutlet
id <FaceViewDataSource> dataSource;
```


La délégation: 4



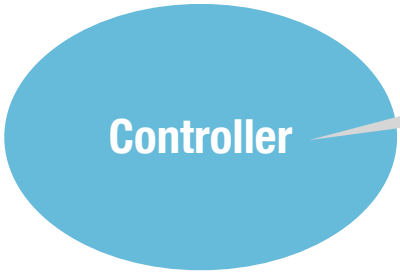
Maintenant, je vais utiliser mon délégué en remplaçant la valeur figée de ma «quantité de sourire», la variable `float smile`, par la valeur que me retourne la méthode `smileForFaceView:` .
J'ai choisi de gérer l'affichage d'une «quantité de sourire» entre -1 et 1, je vais donc me protéger des cas où mon délégué me retournerait des valeurs qui sortent de cette fourchette.

dans FaceView.m

```
float smile = [self.dataSource smileForFaceView:self];
    if (smile > 1) smile = 1;
    if (smile < -1) smile = -1;
```

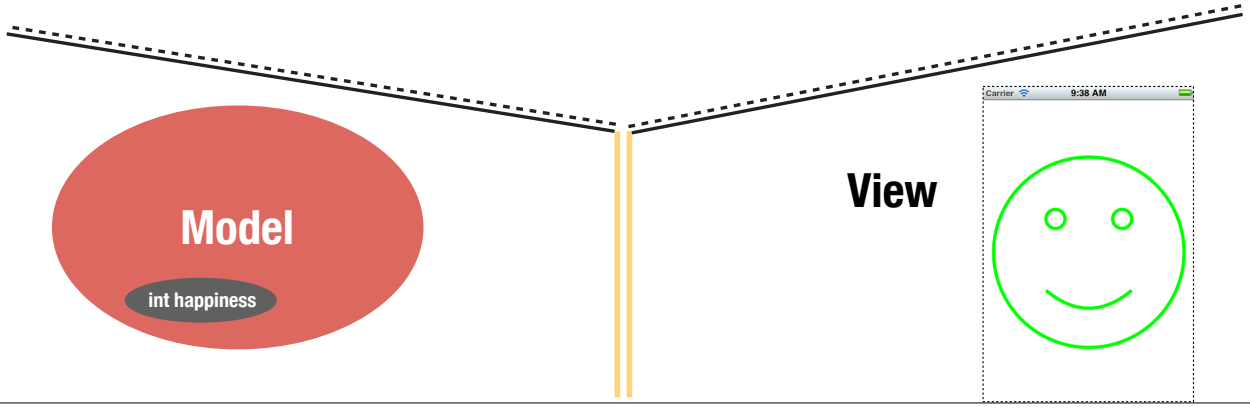
La délégation: 5

Je vais faire en sorte de prendre en charge la «quantité de sourire» de ma View puisqu'elle souhaite déléguer cela à quelqu'un (comme c'est ma servante, c'est normal que je m'en charge...).
Je vais commencer par déclarer que j'implémente le protocole FaceViewDataSource.



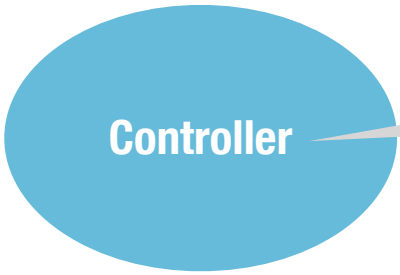
dans HappinessViewController.m

```
@interface HappinessViewController () <FaceViewDataSource>
```



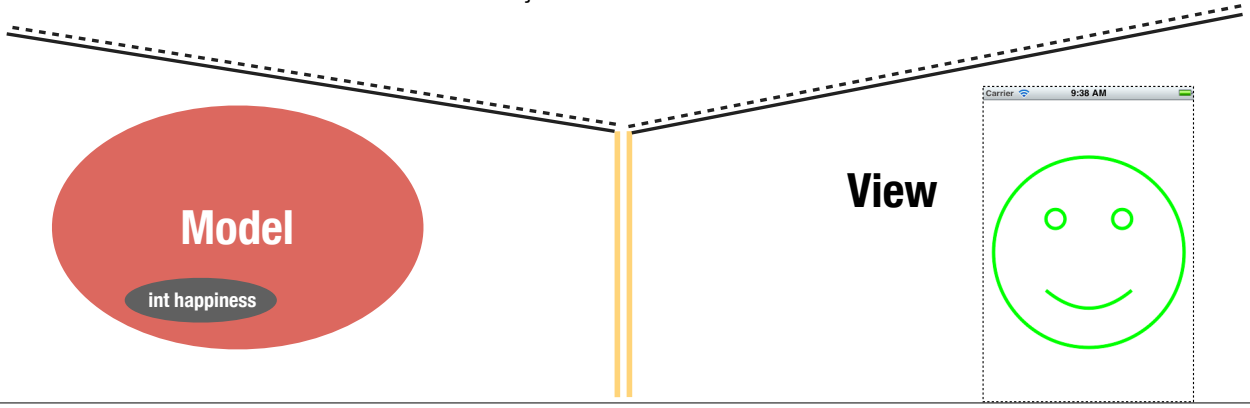
La délégation: 6

Maintenant que j'ai déclaré que j'implémentais le protocole FaceViewDataSource, alors je suis contraint d'implémenter toutes les méthodes obligatoires de ce protocole.
J'ajoute donc dans mon implémentation la méthode – (float)smileForFaceView:(FaceView *)sender {…}



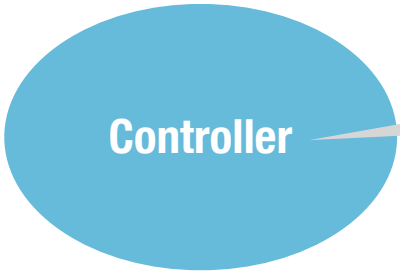
dans HappinessViewController.m

```
- (float)smileForFaceView:(FaceView *)sender
{
    ...
}
```



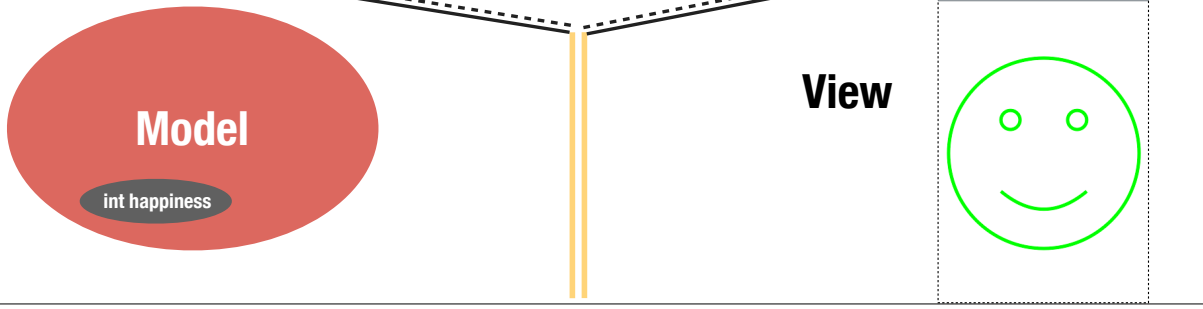
La délégation: 7

Comment implémenter cette méthode smileForFaceView? J'ai un problème... mon modèle est un `int` qui vaut entre 0 et 100, alors que la «quantité de sourire» que la méthode doit retourner est un `float` qui vaut entre -1 et 1!
Il faut trouver une solution pour convertir de l'un à l'autre.



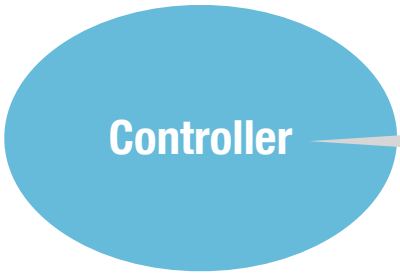
dans HappinessViewController.m

```
- (float)smileForFaceView:(FaceView *)sender
{
    return (self.happiness - 50) / 50.0;
}
```



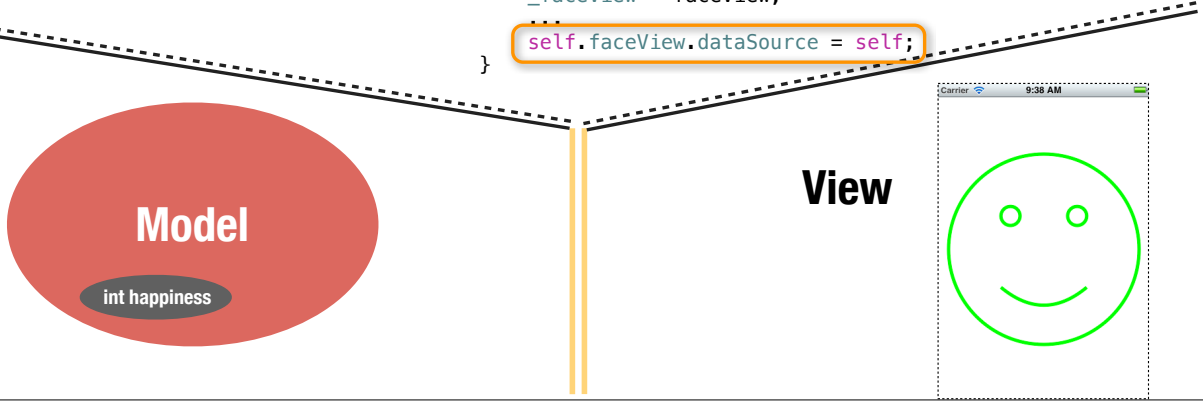
La délégation: 8

J'ai déclaré que j'implémentais le protocole FaceViewDataSource, j'ai implémenté ses méthodes obligatoires, il faut maintenant que je m'annonce comme étant le délégué de ma View!
L'endroit idéal pour le faire, c'est dans le setter de ma propriété faceView (ce setter est utilisé par iOS au chargement du storyboard pour instancier le Controller et la View qui lui est attachée)

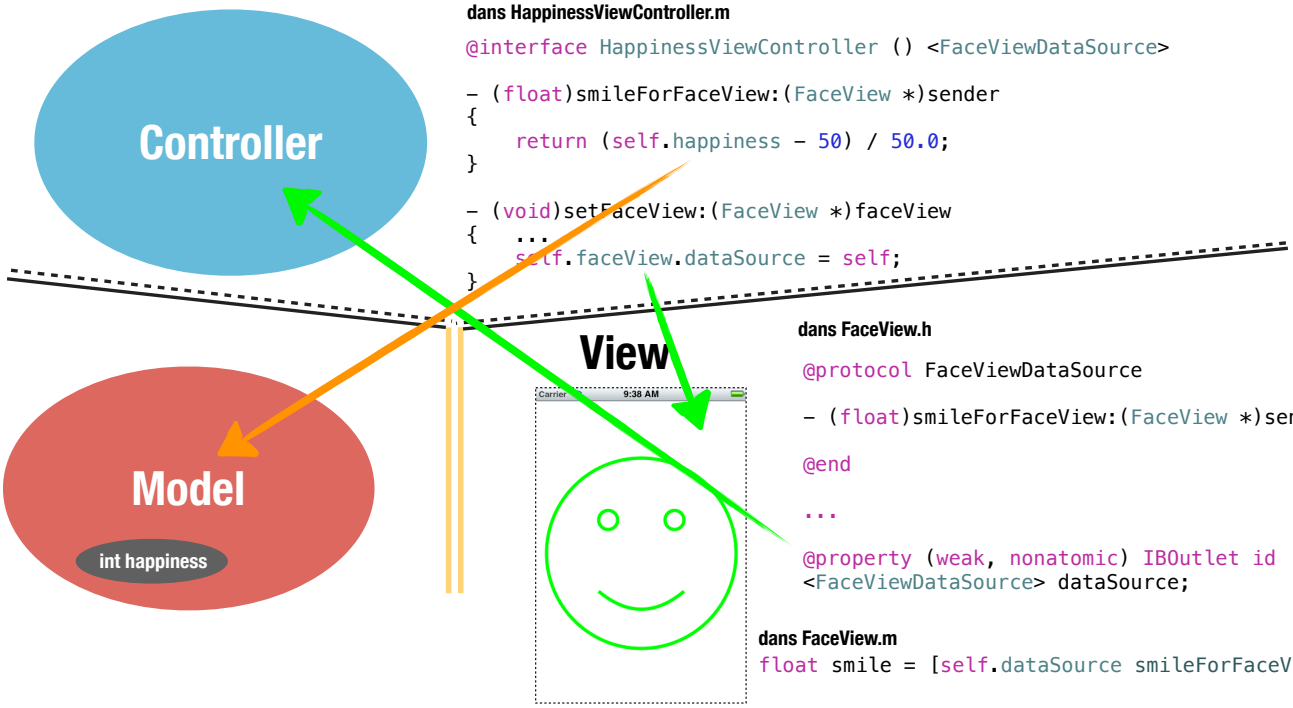


dans HappinessViewController.m

```
- (void)setFaceView:(FaceView *)faceView
{
    _faceView = faceView;
    ...
    self.faceView.dataSource = self;
}
```



La délégation: résumé



dans HappinessViewController.m

```
@interface HappinessViewController () <FaceViewDataSource>

- (float)smileForFaceView:(FaceView *)sender
{
    return (self.happiness - 50) / 50.0;
}

- (void)setFaceView:(FaceView *)faceView
{   ...
    self.faceView.dataSource = self;
}
```

dans FaceView.h

```
@protocol FaceViewDataSource

- (float)smileForFaceView:(FaceView *)sender;

@end

...

@property (weak, nonatomic) IBOutlet id
<FaceViewDataSource> dataSource;
```

dans FaceView.m

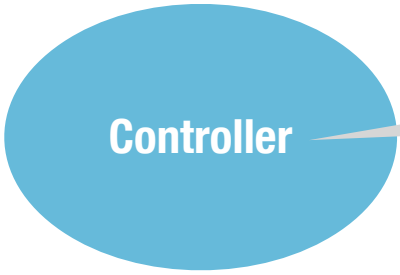
```
float smile = [self.dataSource smileForFaceView:self];
```


Mettre à jour l'affichage avec une Gesture qui modifie le modèle

On veut implémenter une Pan Gesture (lent glissement du doigt) qui modifie le modèle, notre «bonheur», la propriété `int happiness`, ce qui du coup mettra à jour l'affichage du sourire dans la View

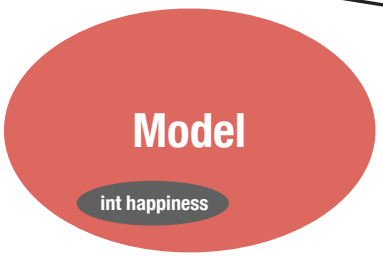
Pan Gesture: 1

Je vais permettre à ma View de réagir aux Pan Gestures.
J'y ajoute donc un `UIPanGestureRecognizer` et le meilleur endroit pour le faire, c'est dans le setter de ma propriété faceView (ce setter est utilisé par iOS au chargement du storyboard pour instancier le Controller et la View qui lui est attachée).

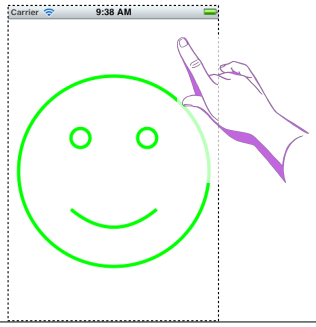


dans HappinessViewController.m

```
- (void)setFaceView:(FaceView *)faceView
{
    _faceView = faceView;
    ...
    self.faceView.dataSource = self;
}
```

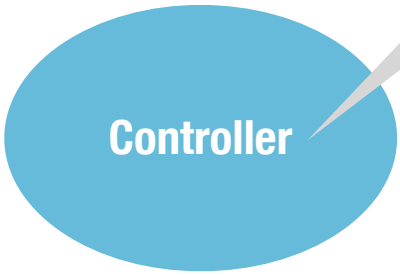


View



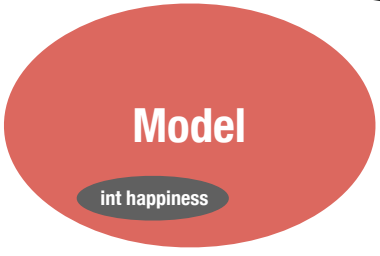
Pan Gesture: 2

Le target de ce `UIPanGestureRecognizer`, celui qui prendra en charge les conséquences de ce mouvement, est moi-même, le Controller.
En effet, ma View ne peut pas se charger de cela elle-même car cette Gesture va modifier le modèle (mon «bonheur», happiness).



dans HappinessViewController.m

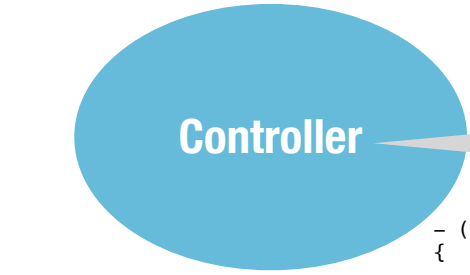
```
- (void)setFaceView:(FaceView *)faceView
{
    _faceView = faceView;
    ...
    [self.faceView addGestureRecognizer:[[UIPanGestureRecognizer
alloc] initWithTarget:self action:@selector(gererHappinessGesture:)]];
    self.faceView.dataSource = self;
}
```



View

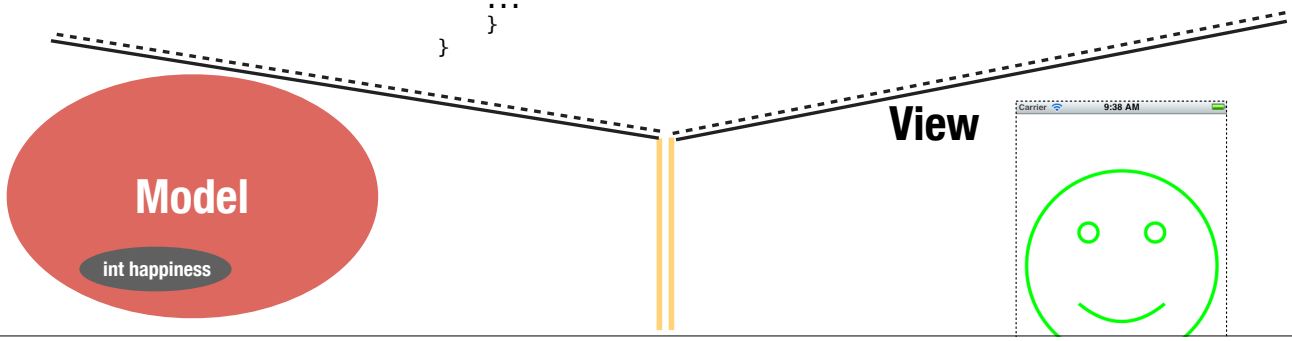
Pan Gesture: 3

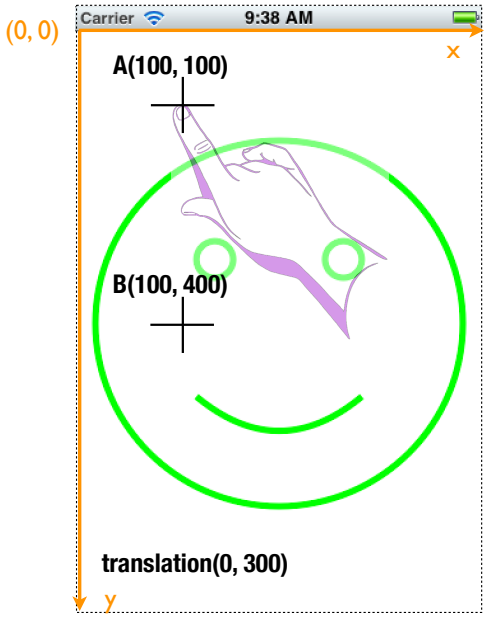
Je vais implémenter la méthode `gererHappinessGesture:`. Elle prend en paramètre le UIPanGestureRecognizer qui en est à l'origine. Elle doit s'activer lorsque l'état de mon GestureRecognizer passe à *changed* ou à *ended*, donc je vérifie que l'une de ces conditions est présente.
Je définis une variable de type CGPoint nommée `translation` et je place dedans ce qui m'est retourné par la méthode `translationInView:` , à savoir la différence de coordonnées x et y par rapport au point précédemment mesuré dans le déplacement.

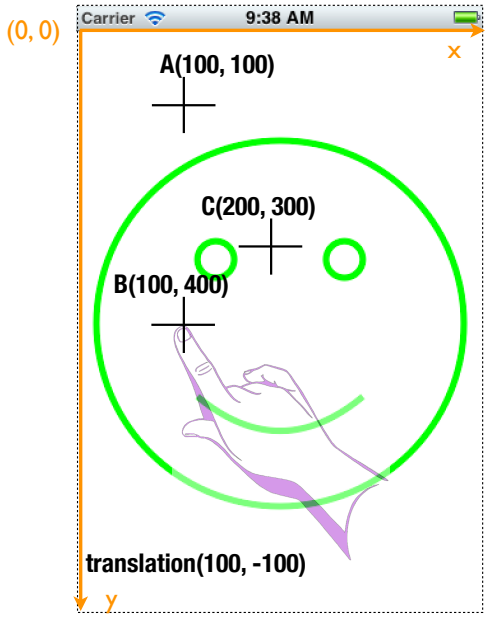


dans HappinessViewController.m

```
- (void)gererHappinessGesture:(UIPanGestureRecognizer *)gesture
{
    if ((gesture.state == UIGestureRecognizerStateChanged || gesture.state
== UIGestureRecognizerStateEnded)) {
        CGPoint translation = [gesture translationInView:self.faceView];
    ...
    }
}
```

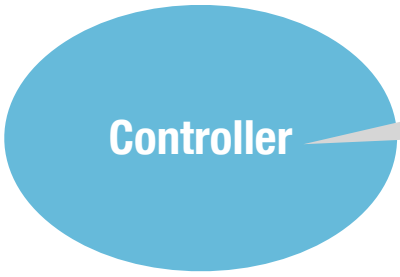






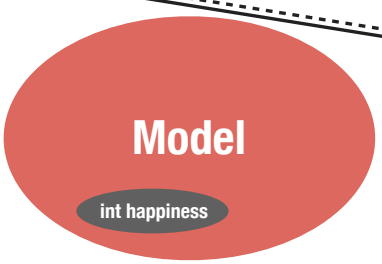
Pan Gesture: 4

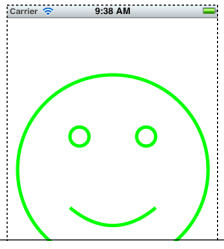
Finalement, j'ai toutes les informations pour pouvoir modifier le modèle. Je vais chercher la valeur de ma propriété happiness (mon modèle), je lui soustrais la valeur du déplacement sur l'axe des y et je lui réattribue cette valeur. Pour ne pas que le mouvement soit trop sensible, je choisis de diviser par un facteur 2. C'est optionnel. Comme je n'ai pas envie de mesurer un déplacement incrémental, mais uniquement toujours le déplacement par rapport au dernier point mesuré, je remets la variable translation à zéro.



dans HappinessViewController.m

```
- (void)gererHappinessGesture:(UIPanGestureRecognizer *)gesture
{
    if ((gesture.state == UIGestureRecognizerStateChanged ||
gesture.state == UIGestureRecognizerStateEnded)) {
        CGPoint translation = [gesture translationInView:self.faceView];
        self.happiness -= translation.y / 2;
        [gesture setTranslation:CGPointZero inView:self.faceView];
    }
}
```





Tout est en place pour que la gesture fonctionne

Nous avons:

- un recognizer

- une target qui prend en charge les conséquences du mouvement

- une méthode qui est implémentée et qui modifie le modèle selon le déplacement relevé dans la View