Gestures

Gestures são uma maneira de interagir com *views* em um nível mais baixo do que aquele provido pelos eventos de um componente (como *touch down*, *moved*, *up*, etc.), permitindo assim um maior controle sobre a maneira com que o usuário interage com elas. Os *gestures* são reconhecidos baseados em uma classe "abstrata" chamada **UIGestureRecognizer**. Uma série de gestos padrão já definidos são disponibilizados como classes filhas destas.

Gestures padrão

Tap

O mais básico dos gestos é o *tap*, um pressionar rápido na tela ou em um elemento visual. Este gesto pode ser feito com um ou mais dedos.

Este gesto é utilizado, em geral, para disparar a ação primária de um componente, a seleção de uma célula em uma **TableView** ou **CollectionView** ou o ato de "pressionar" o item (como um botão). Este gesto é reconhecido pela classe **UITapGestureRecognizer**, que pode definir o número de touches (número de dedos) e de taps (numero de vezes consecutivas que a tela deve ser tocada) para que este seja identificado. Estas informações são dadas pelas propriedades *numberOfTouchesRequired* e *numberOfTapsRequires* respectivamente.

No exemplo de aula, incluiremos uma ação de tap na view principal. Para isto selecione o storyboard e na **Object Library**(na área inferior direita do XCode), o componente "**Tap Gesture Recognizer**". Em seguida, arraste-o para a *view* principal (cuidado para não adicioná-la na *view* do baiacu ou da imagem de fundo). Isto define sobre qual *view* o gesto pode ser reconhecido. O gesto deve ser visível também na barra superior da cena onde foi incluido. A partir do ícone que foi adicionado, com a tela dividida (Selecione "*Show the Assistant Editor*") e com a classe **SillyBlowFishViewController** aparecendo (em modo Automático), clique no ícone do gesto que foi incluído e, mantendo pressionado a tecla control e o trackpad ou mouse arraste-o para o código. Isto deve criar a *action* que será disparada quando o tap for detectado. Nomeie o método como tap e altere o **Type** para **UITapGestureRecognizer**.

Com isto o gesto de *tap* em toda a área da *view* principal já pode ser detectada e neste caso o método **tap** será disparado. Preencheremos a ação a ser tomada neste caso:

```
@IBAction func tap(_ sender: UITapGestureRecognizer) {
    let position = sender.location(in: self.fish)
    fish.lookPosition(position: position)
}
```

Este código fará com que o baiacu olhe para a posição clicada na tela. A função **lookPosition** fornecida é responsável por isto, mas espera uma posição (de acordo com as coordenadas do baiacu) para tal. O **UITapGestureRecognizer** nos fornece o método **locationInView** que retorna a posição de um *tap* de acordo com o sistema de coordenadas da *view* fornecida como parâmetro.

O *tap* suporta um número de toques (uma sequência rápida de toques na tela ou em um componente específico). Vamos alterar isto no *storyboard*. Nas propriedades do componente de reconhecimento do *tap* (selecione-o pelo *Document Outline* ou na parte superior da cena) vá para a aba *Attribute Inspector* e altere a propriedade **taps** para 2. Com isto, nosso herói só olhará para a posição clicada se dois toques consecutivos forem aplicados.

O *tap* suporta também um gesto *multitouch*. Vamos criar programaticamente um reconhecedor de *tap* que trate isto. Altere o método **viewDidLoad** da classe **SillyBlowFishViewController** para:

```
override func viewDidLoad() {
   super.viewDidLoad()

   self.fish.startAnimating()

   let twoFingersTap = UITapGestureRecognizer(target: self,
        action:#selector(SillyBlowfishViewController.tapTwoFingers(_:)))
   twoFingersTap.numberOfTouchesRequired = 2
   self.view.addGestureRecognizer(twoFingersTap)

}
```

Definimos o número de toques simultâneos com a propriedade **numberOfTouchesRequired**. Quando este gesto ocorrer o método disparado será o **tapTwoFingers** (definido pelo comando **#selector**. Vamos definir este método:

```
func tapTwoFingers(_ tap:UITapGestureRecognizer) {
   let position1 = tap.location(ofTouch: 0, in: self.fish)
   let position2 = tap.location(ofTouch: 1, in: self.fish)
   fish.lookPosition(first: position1, second: position2)
}
```

Long Press

O gesto de *long press *é similar ao *tap*, porém repousando o dedo por um tempo mais longo sobre a tela. Este gesto é utilizado para ações tais quais selecionar um texto ou entrar em modo de edição (com uma lupa aumentando a região para uma seleção mais precisa). Porém vale ressaltar que por ser tão similar ao *tap* este gesto nem sempre é óbvio ao usuário e pode ser pouco utilizado ou mesmo induzir ao erro. Portanto utilize-o com cuidado.

Este gesto é identificado pela classe **UILongPressGestureRecognizer**. Além das propriedades para *taps* e *touches* similar ao **UITapGestureRecognizer** esta classe também possui as propriedades **allowableMovement** e **minimumPressDuration**, que definem, respectivamente, a quantidade de movimento dos dedos aceitável e o tempo mínimo que os dedos devem repousar na tela para que o gesto seja reconhecido.

Vamos adicionar programaticamente o reconhecimento do gesto (embora isto possa ser feito elo *Interface Builder* de forma similar ao que fizemos anteriormente). Vamos incluí-lo no final do **viewDidLoad**:

```
override func viewDidLoad() {
...
    let longPress = UILongPressGestureRecognizer(target: self,
            action: #selector(SillyBlowfishViewController.longPress(_:)))
     self.fish.addGestureRecognizer(longPress)

}
```

Veja que estamos adicionando este gesto apenas a view que desenha o baiacu. E o método que trata o gesto:

```
func longPress(_ gesture: UILongPressGestureRecognizer) {
   fish.pullMyFinger()
}
```

Faremos um segundo reconhecedor, para o caso de um long press com dois dedos. Para tal também devemos adicioná-lo e configurá-lo no viewDidLoad:

```
override func viewDidLoad() {
...
    let twoFingersLongPress = UILongPressGestureRecognizer(target: self, action:
#selector(SillyBlowfishViewController.twoFingersLongPress(_:)))
    twoFingersLongPress.numberOfTouchesRequired = 2
    self.view.addGestureRecognizer(twoFingersLongPress)
}
```

E temos que criar o método que trata este gesture:

```
func twoFingersLongPress(_ gesture: UILongPressGestureRecognizer) {
   fish.easterEgg()
}
```

Nota: Veja que utilizamos este gesto como um *easter egg*. Lembre-se que gestos pouco padrão, como

este, serão de difícil acesso e identificação pelo usuário! Devemos sempre que possível tentar utilizar gestos facilmente reconhecíveis e usuais, pois não existe um indicativo explícito na *app* de que esta interação existe, como ocorre com elementos visuais.

Pinch

O gesto de "pinçar" (*pinch*) é aquele de colocar dois dedos na tela e afastá-los ou aproximá-los, arrastando-os por ela. Este gesto é normalmente associado a ações de zoom in e zoom out, com o primeiro sendo feito ao afastarmos os dedos e o segundo ao aproximá-los.

Este gesto é tratado pela classe **UIPinchGestureRecognizer** que possui as propriedades **scale** e **velocity**, a escala de zoom atual e a velocidade com que os dedos estão se afastando ou aproximando.

Adicionaremos este gesto ao exemplo, também no final do **viewDidLoad**:

```
override func viewDidLoad() {
...
   let pinch = UIPinchGestureRecognizer(target: self, action: #selector(SillyBlow
fishViewController.pinch(_:)))
   self.view.addGestureRecognizer(pinch)
}
```

E o método que o tratará:

```
func pinch(_ gesture:UIPinchGestureRecognizer) {
   self.fish.inflate(scale: gesture.scale)
}
```

Este método fará com que o baiacu infle ou esvazie de acordo com o gesto. Veja o que acontece se o gesto for feito com os dedos mais próximos ou mais afastados no momento do toque.

Rotate

O gesto de rotação é feito com o toque de dois dedos na tela e a rotação destes em torno do ponto central entre ambos. Este gesto é utilizado em geral para rotacionar elementos na tela ou o ponto de vista destes (como em um mapa). Este gesto é tratado pela classe **UIRotationGestureRecognizer** que fornece a rotação realizada(**rotation**) e a sua velocidade (**velocity**). Adicionaremos este evento a nossa *view* principal no exemplo:

```
override func viewDidLoad() {
...
   let rotate = UIRotationGestureRecognizer(target: self,
                action: #selector(SillyBlowfishViewController.rotate(_:)))
   self.view.addGestureRecognizer(rotate)
}
```

Faremos com que a concha que aparece na tela rotacione de acordo com o gesto:

```
func rotate (_ gesture:UIRotationGestureRecognizer) {
    let rotation = gesture.rotation
    self.shell.transform = self.shell.transform.rotated(by: rotation)
}
```

Estamos neste método utilizando o grau de rotação para alterar a matriz de transformação associada com a **ImageView** que desenha a concha. Porém podemos notar que ela gira de forma rápida demais em relação ao nosso movimento. Isto se deve ao fato que a rotação é cumulativa ao longo do gesto e este método é disparado diversas vezes durante sua realização. Para evitar isto podemos zerar a rotação já tratada de forma a tratarmos apenas o "delta" entre duas chamadas consecutivas do método **rotate**:

```
func rotate (_ gesture:UIRotationGestureRecognizer) {
    let rotation = gesture.rotation
    self.shell.transform = self.shell.transform.rotated(by: rotation)
    gesture.rotation = 0
}
```

Com isto geramos um movimento muito mais suave e sincronizado com o gesto.

Pan

O gesto de *pan* é aquele de pressionar a tela e arrastar o dedo sobre ela. Este gesto é utilizado usualmente para *scrolling*, para mover o ponto de vista em uma *view* maior que a tela (*pan*, como em um mapa ou imagem grande) ou para eventos de *drag & drop*. Implementaremos este gesto para mover o baiacu pela tela. Inicialmente adicionando o gesto ao **viewDidLoad**:

```
override func viewDidLoad() {
...
   let pan = UIPanGestureRecognizer(target: self,
                action: #selector(SillyBlowfishViewController.pan(_:)))
   self.view.addGestureRecognizer(pan)
}
```

E temos que tratar o gesto:

```
func pan (_ gesture:UIPanGestureRecognizer) {
   let translation = gesture.translation(in: self.view)

   self.fish.transform = self.fish.transform.translatedBy(x: translation.x,
                                                          y: translation.y)
}
```

O método **translation** nos retorna o quanto houve de deslocamento do gesto de *pan* em relação a view dada como referência. Utilizamos esta informação para aplicá-la a matriz de transformação do peixe.

Ao executarmos o código veja que o deslocamento é muito rápido. Da mesma forma que com o método de rotação a translação é cumulativa. Precisamos "resetá-la" entre as chamadas do método.

```
func pan (_ gesture:UIPanGestureRecognizer) {
    let translation = gesture.translation(in: self.view)

    self.fish.transform = self.fish.transform.translatedBy(x: translation.x,
                                                           y: translation.y)
    gesture.setTranslation(CGPoint.zero, in: self.view)
}
```

O método **setTranslation** nos permite alterar o valor da translação associada ao gesto até este ponto (em relação a uma dada *view*).

Swipe

O gesto de *swipe* é um breve pan feito na tela em uma direção. Este gesto é em geral utilizado para mover entre conteúdos diferentes (como em uma paginação) ou para revelar elementos escondidos (como na *home screen* do iOS ou em uma célula de uma *table view*). Este gesto é identificado pela classe **UISwipeGestureRecognizer** e a direção do *swipe* pode ser definido pela propriedade **direction**.

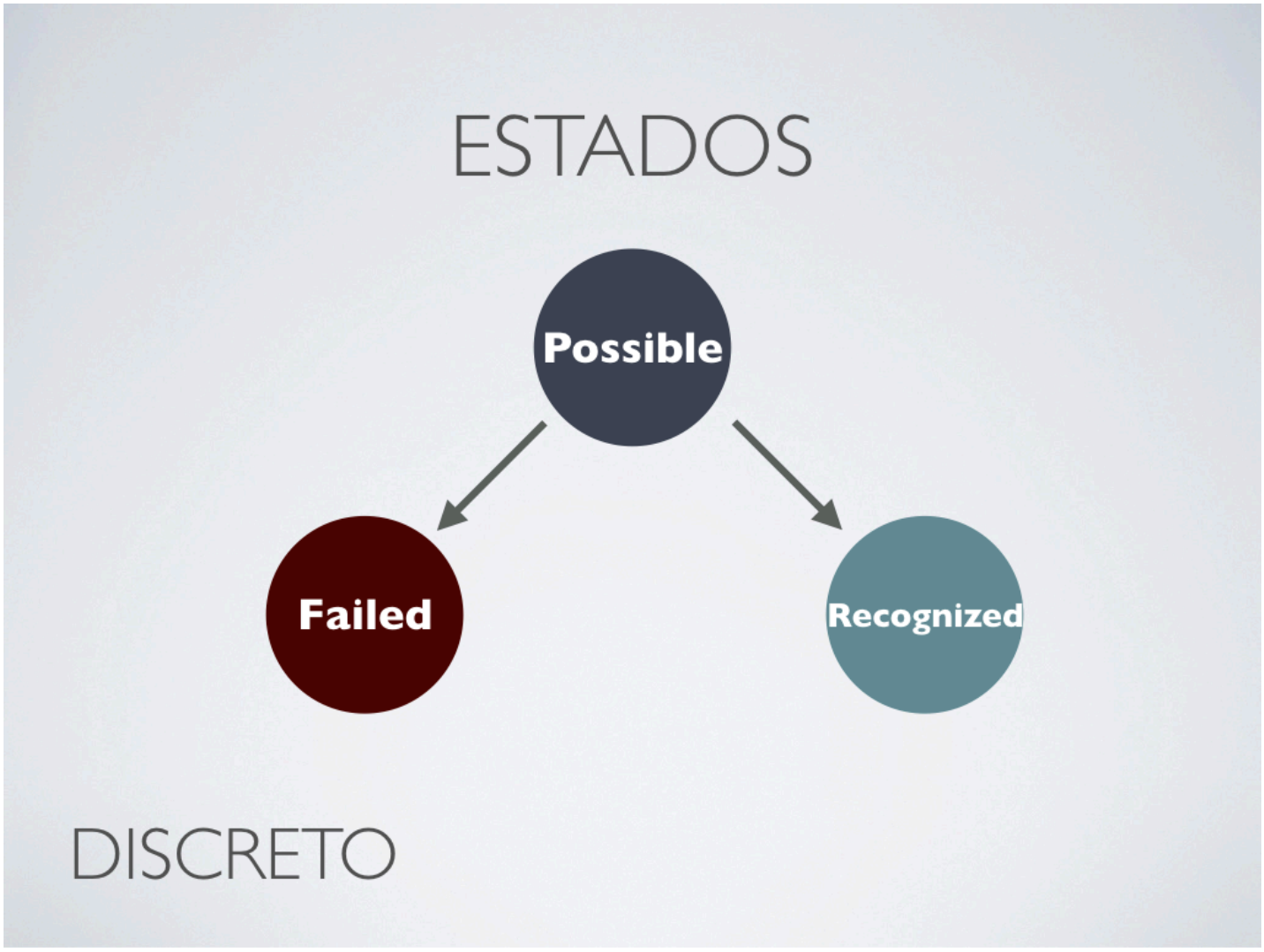
Estado de um gesto

Os gestos como *tap* e *long press* são eventos curtos e tratados de um só vez. Outros, como o *pan*, rotação e *pinch* são longos e tratados de forma progressiva. Os primeiros são considerados gestos discretos e os segundos gestos contínuos. Os reconhecedores mantém uma máquina de estados para tratar o processo de reconhecimento de um gesto. Esta máquina de estados difere bastante no caso de um gesto discreto ou contínuo.

Um gesto discreto se inicia quando um dado conjunto de toques é candidato a ser reconhecido como tal gesto. Isto inicia o processo de reconhecimento no estado **Possible**. Caso o reconhecedor perceba que o

gesto não era aquele pelo qual é responsável o reconhecimento é terminado e o estado alterado para **Failed**. Caso contrário ele é bem sucedido e o estado alterado para **Recognized** e seu método de tratamento é disparado.

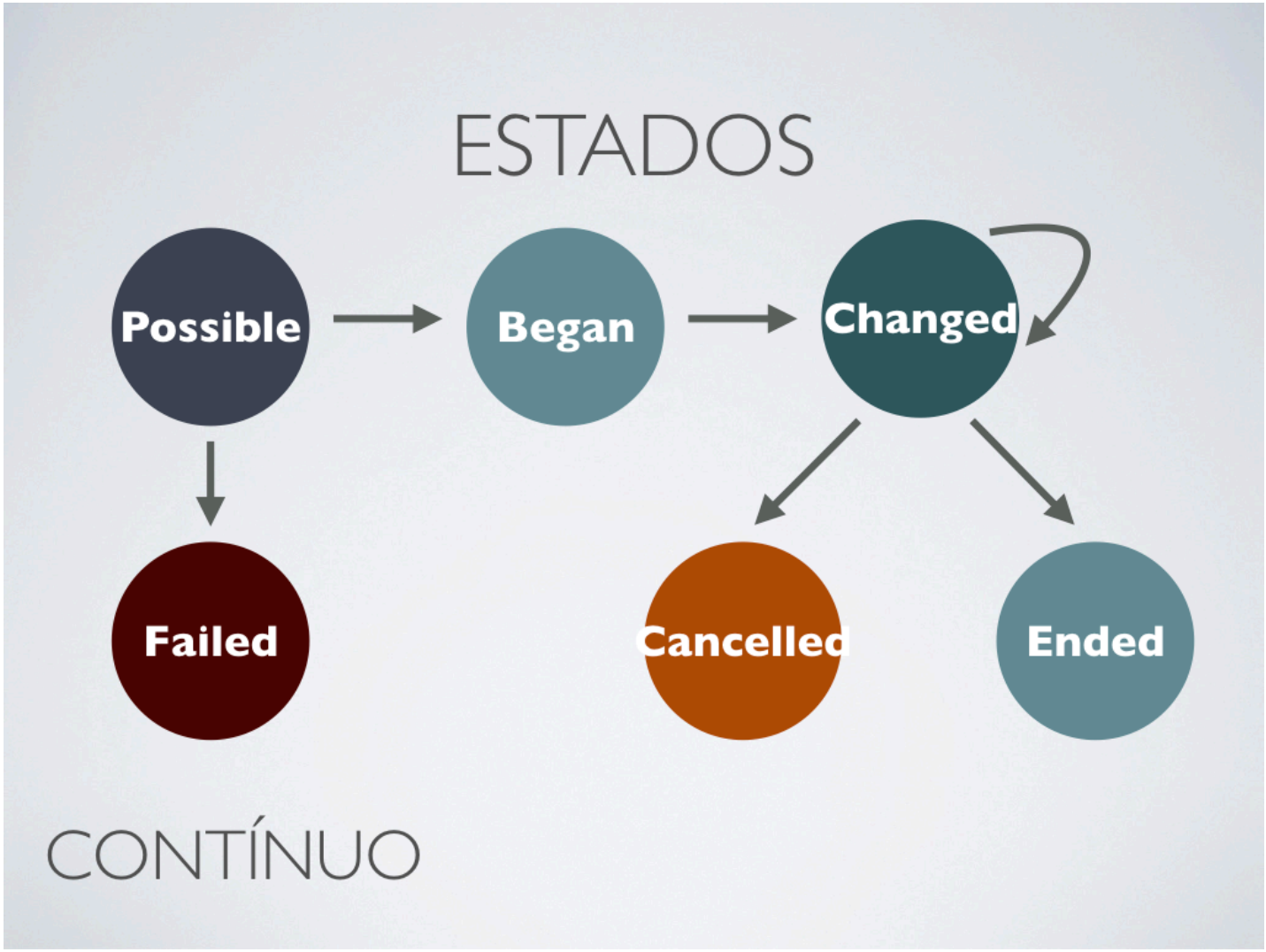
A imagem abaixo ilustra esta máquina de estados:



No caso de um gesto contínuo o início é similar para os estados **Possible** e **Failed**. Porém, quando o gesto é "reconhecido" o estado é alterado para **Began**, indicando o início de uma ação longa.

Enquanto o gesto se desenrola o estado é mantido como **Changed** que pode terminar com o cancelamento deste (no caso de notar-se que se trata de outro gesto ou por outro fator) e seu estado sendo alterado para **Cancelled** ou terminado de forma bem sucedida com o fim do (ou dos) toque(s) e seu estado alterado para **Ended**. Durante os estados afirmativos **Began**, **Changed** e **Ended** o método de tratamento é disparado.

A imagem abaixo ilustra esta máquina de estados:



Vamos alterar o método **pan** para tratar a mudança de estados de um gesto contínuo:

```
func pan (_ gesture:UIPanGestureRecognizer) {
   let translation = gesture.translation(in: self.view)

   self.fish.transform = self.fish.transform.translatedBy(x: translation.x, y: tr
anslation.y)
   gesture.setTranslation(CGPoint.zero, in: self.view)

   //Tratamento do estado do gesto

   if (gesture.state == .began) {
       self.fish.startFollowing()
   }

   if (gesture.state == .ended) {
       self.fish.stopFollowing()
   }
}
```

Veja que agora fazemos um tratamento diferente para quando o gesto se inicia e quanto termina.

Gestos Customizados

Caso seja necessário tratar uma interação não padrão, o iOS nos permite criar *custom gestures* em um nível mais baixo que o dos *gesture recognizers* disponíveis, considerando os toques dados na tela.

Isto é feito herdando da classe **UIGestureRecognizer** e de seus métodos **touchesBegan**, **touchesMoved** e **touchesEnded**. Para estes métodos temos disponíveis os toques feitos na tela (definidos por objetos da classe **UITouch**) e da sequência de posições destes (dados pela classe **UIEvent**).

Caso seja necessário um controle maior do gesto, a alteração de seus comportamentos ou mesmo tratar como ele interagirá com outros gestos podemos utilizar o seu delegate e implementar o protocolo **UIGestureRecognizerDelegate**.