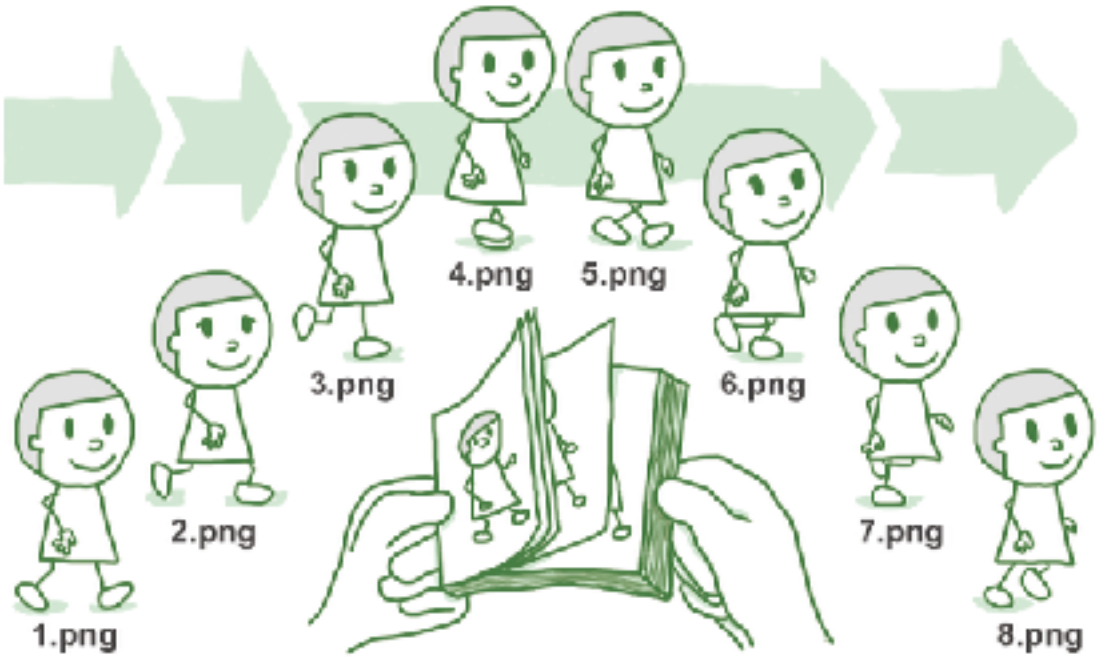
CHAPTER 動畫

逐格動畫



[問題]如何播放多張圖像，做出如卡通一樣逐格動畫的效果
[解答]使用UIImageView的播放逐格動畫功能
[範例程式碼]HelloFrameByFrameAnimation
[過程解說]
1.請先開一個新的Single View Application專案，選擇左邊欄的Assets.xcassets資料夾，把所需的圖檔放進專案中。(本範例的圖檔，在本章附贈程式碼中的Walking AnimationImages資料夾中。)

2.在左邊欄選擇Main.Storyboard之後，在右下方的搜尋框搜尋UIImageView。

把UIIImageView拉到畫面上。Autolayout設定的話，把這個 UIImageView的寬度固定在100、高度則固定在200，水平及垂直方向都設定在畫面的中央。最後打開Assistant Editor，把這個UIImageView連結到程式碼中，命名成animationView。

3.選擇ViewController.swift，把ViewController類別裡的程式碼改動如下：

```
class ViewController: UIViewController {

    @IBOutlet weak var animationView: UIImageView!//播放動畫的ImageView

    override func viewDidLoad() {
        super.viewDidLoad()
        animationView.animationImages = [
            UIImage(named: "1")!,
            UIImage(named: "2")!,
            UIImage(named: "3")!,
            UIImage(named: "4")!,
            UIImage(named: "5")!,
            UIImage(named: "6")!,
            UIImage(named: "7")!,
            UIImage(named: "8")!                    //放入要做動畫的圖片
        ]
        animationView.animationDuration = 0.8       //設定動畫播放速度
        animationView.animationRepeatCount = 0      //設定動畫播放次數
        animationView.startAnimating()              //開始播放動畫

        //animationView.stopAnimating()             //停止播放動畫
    }
}
```

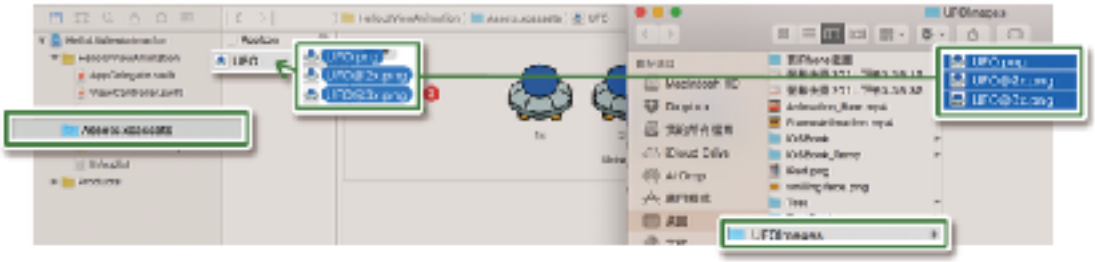
4.上面程式碼中，先把所有的圖片讀到程式中，把這些圖片放到一個陣列裡。每個UIImage後面都加上了驚嘆號，確保一定有這些圖像可供製作動畫。

5.要讓畫面上的animationView播放各種圖片、做出逐格動畫的效果，只需把上

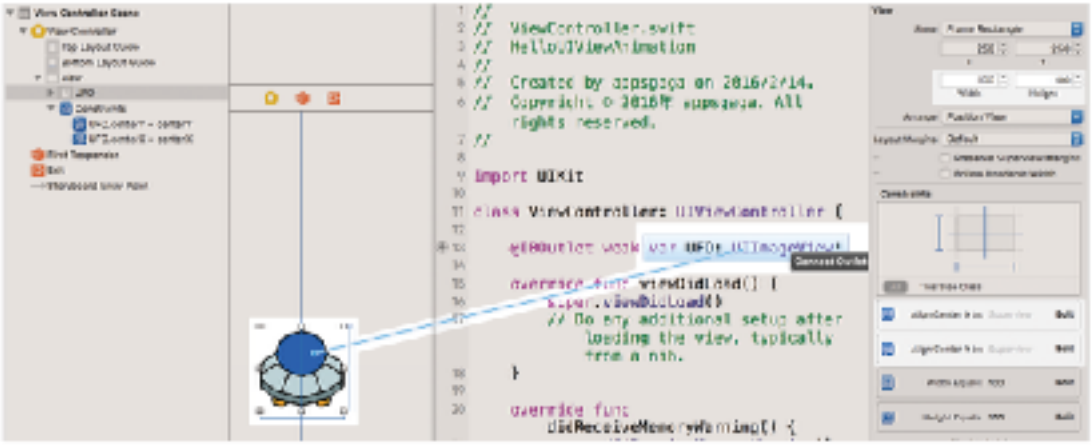
個步驟顯示圖片的陣列，設定給animationView的animationImages屬性。
6.呼叫animationView.startAnimating方法，程式就會開始播放動畫了。
7.設定animationView.animationDuration可以控制動畫播放的速度。
8.設定 animationView.animationRepeatCount可以控制動畫播放的次數，設成1播放一次，設成2播放兩次，設成3播放三次...如果設成0的話，會持續不停地播放。
9.如果要停止動畫的播放，就呼叫animationImages.stopAnimating方法。

UIView動畫(補間動畫)

[問題]如何做出簡單的UIView動畫
[解答]使用UIView.animateWithDuration方法，可以做出簡單動畫
[範例程式碼]HelloUIViewAnimation
[最簡單的UIView動畫]
1.請先開一個新的Single View Application專案。選擇左邊欄的Assets.xcassets資料夾，把所需的圖檔放進專案中。(本範例的圖檔，在本章附贈程式碼中的UFOImages資料夾中。)



2.在左邊欄選擇Main.Storyboard之後，在右下方的搜尋框搜尋UIImageView。把UIIImageView拉到畫面上，調整其顯示圖像為步驟1匯入專案的UFO圖像。之後，設定Autolayout的寬度是100、高度100，水平及垂直都在畫面的中央。最後打開Assistant Editor，把這個UIImageView連結到程式碼中，命名成UFO。



3.選擇ViewController.swift，把ViewController類別裡的程式碼改動如下：

```
class ViewController: UIViewController {

    @IBOutlet weak var UFO: UIImageView!

    override func viewDidAppear(_ animated: Bool) {
        UIView.animate(withDuration: 1.0, animations: {
            //把要做的動畫放在這個 Closure 裡面
            self.UFO.frame.origin.y = self.UFO.frame.origin.y - 100
        })
    }
```

動畫時間

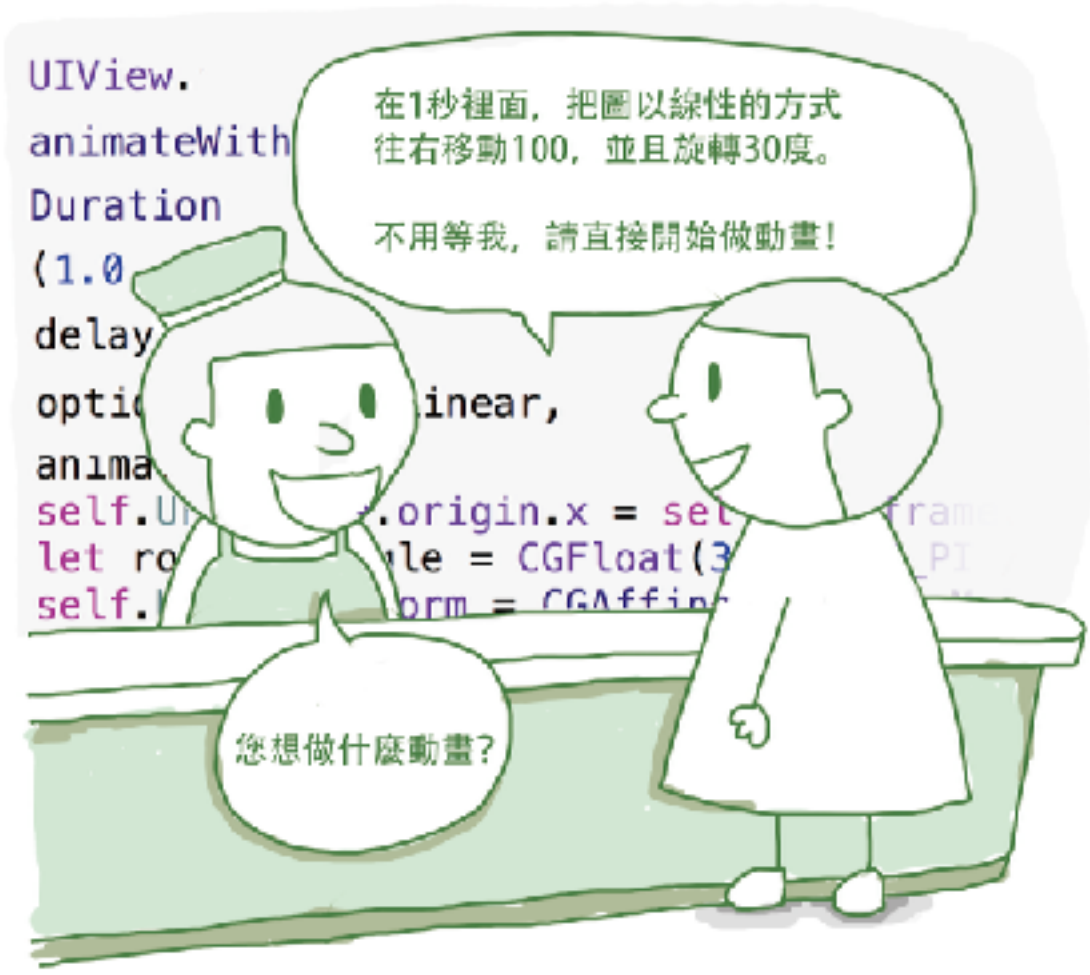
要做的動畫

4.使用UIView.animate方法，就可以移動畫面上的UIImageView圖像或是任何UIView的子類別。這個方法有很多種變形，上面的範例是最簡單的一個。
請注意，這個方法並不是寫在viewDidLoad的方法，而是寫在viewDidAppear的方法中。因為程式在呼叫viewDidLoad的方法時，圖片在畫面上的位置尚未完全確定。所以改在viewDidAppear的方法呼叫。您可以依情況使用UIView.animateWithDuration方法。比方說在使用者按下按鈕或是使用者觸碰畫面的過程中開始動畫。
5.最簡單的UIView.animateWithDuration方法接受兩個參數：第一個是動畫的時間，第二個參數是一個Closure、把要做的動畫全部寫在第二個參數這個閉包裡面。裡面設定的，是動畫結束時，圖像的屬性狀態。範例中會做的動畫效果是在1.0秒的時間裡面，把圖像從原來的位置，移動到距離原來位置上方100的地方。
6.由於是在Closure裡面設定動畫的程式碼，於是UIImageView圖像的前面要加上self。
7.使用self.UFO.frame.origin.y就可以得到目前圖像的y座標，減掉100之後重新存回self.UFO.frame.origin.y，就可以讓圖像往原本的位置上方移動100。
8.在設定動畫結束圖像的屬性時，可以設定多項屬性，比方說在原來的方法中加入兩行程式碼：

```
class ViewController: UIViewController {

    @IBOutlet weak var UFO: UIImageView!

    override func viewDidAppear(_ animated: Bool) {
        UIView.animate(withDuration: 1.0, animations: {
            //把要做的動畫放在這個 Closure 裡面
            self.UFO.frame.origin.y = self.UFO.frame.origin.y - 100

            //加入下面兩行程式碼：設定透明度為0，設定放大兩倍
            self.UFO.alpha = 0
            self.UFO.transform = CGAffineTransform(scaleX: 2.0, y: 2.0)
        })
    }
}
```

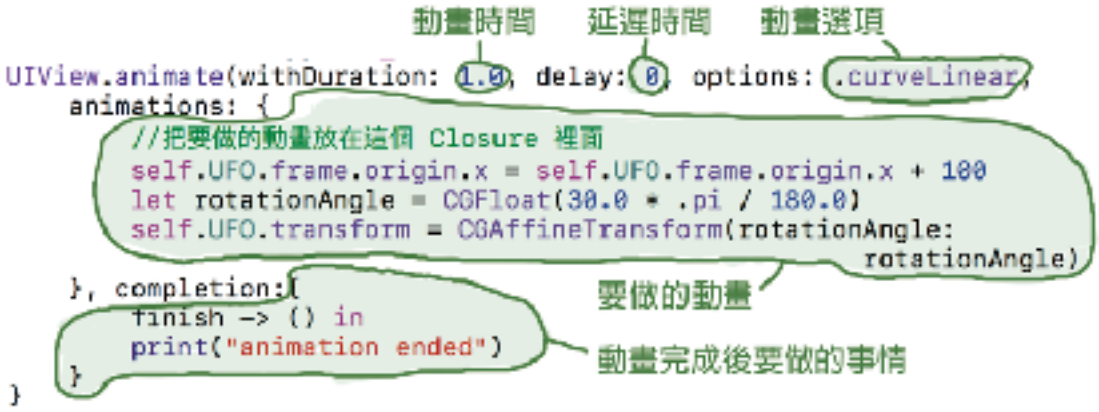
9.在UIView.animate方法的第二個參數多加入兩行程式碼，圖像就會除了往上移動100以外，同時也會一面變成完全透明、一面放大兩倍。以上是UIView.animate方法、最簡單版本的介紹。

[完整的UIView動畫]



除了上面的寫法以外，UIView.animate方法也有很多種變形，請把上面範例中、viewDidAppear方法裡的UIView.animationWithDuration方法改成下面的程

式碼：



1.以上的這種變形，加入了多個參數，參數的數目一舉從兩個變成五個。其中第一個參數還是動畫的時間、第二個參數則是設定延遲多久開始做出動畫的效果、第三個參數設定如何做動畫、第四個參數設定要做的動畫，第五個參數設定動畫完成後要做的事情。
2.範例中，動畫時間設定為1.0，所以會在一秒裡面做動畫。延遲時間設定為0，所以呼叫方法時即刻就開始做動畫。
3.繼續解釋上面的範例程式碼，動畫選項設定成「.curveLinear」，所以動畫會以線性的方式呈現。動畫的選項除了線性以外，還有「.autoreverse(自動倒退播放)」、「.repeat(重複)」、「.curveEaseIn(動畫開始時速度放慢)」、「.curveEaseOut(動畫結束時速度放慢)」，與「.curveEaseInOut(動畫開始與結束速度放慢)」等選項...。
4.範例中把要做的動畫放到第四個參數的Closure裡面。設定動畫做完後，x方向要往右移動100，並且用CGGAffineTransform方法，將圖片旋轉30度。所以程式碼執行的結果，就會在1秒內一面往右邊移動100，一面旋轉30度。
5.上面範例的第五個參數、設定要做的事情、目前只有印出「animation ended」。除了可以印出訊息以外，其實還可以在這個參數裡面，寫入新的一套UIView.animate方法，讓圖片在做完動畫時，再做另外一套新的動畫。

[結合各種動畫選項]

之前範例的程式碼，在動畫選項設定成「.curveLinear」，所以動畫會以線性的方式呈現。動畫選項除了可以代換成其他不同的選項以外，還可以把各種動畫選項結合起來：

結合三種選項

```
UIView.animate(withDuration: 1.0, delay: 0,
               options: [.repeat, .autoreverse, .curveEaseIn],
    animations: {
        //把要做的動畫放在這個 Closure 裡面
        self.UFO.frame.origin.x = self.UFO.frame.origin.x + 100
        let rotationAngle = CGFloat(30.0 * .pi / 180.0)
        self.UFO.transform = CGAffineTransform(
                                        rotationAngle: rotationAngle)
    }, completion:{
        finish -> () in
        print("animation ended")
    }
)
```

如上程式碼，由於更改了動畫的選項，所以動畫會重複、以開始速度放慢的方式來播放動畫，除此以外，也會做出自動倒退的效果。

[彈性的動畫]

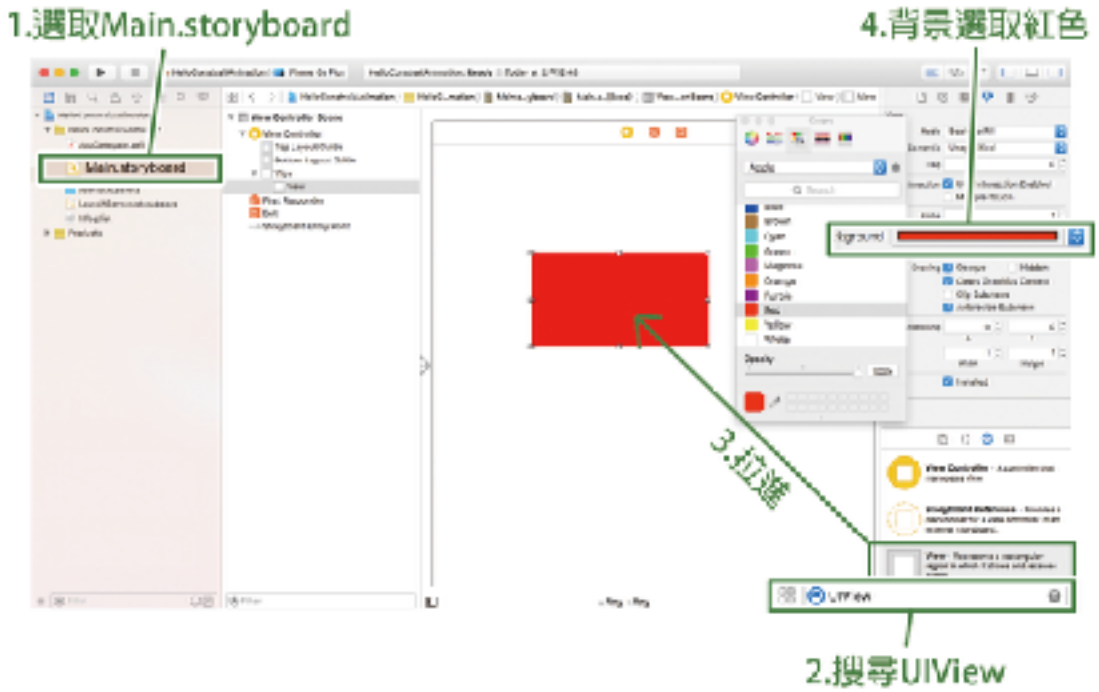
另外一種變形的UIView.animate方法，可以讓動畫有像彈簧一樣的效果。請再把上面範例中、viewDidAppear方法裡的UIView.animate方法改成下面的程式碼：

```swift
UIView.animate(withDuration: 1.0, delay: 0,
               usingSpringWithDamping: 0.3, initialSpringVelocity: 0.5,
                        options: [.repeat, .autoreverse], animations: {
    //同時設定x跟y座標
    let xFinal = self.UFO.frame.origin.x + 100
    let yFinal = self.UFO.frame.origin.y + 100
    let finalPoint = CGPoint(x: xFinal, y: yFinal)
    self.UFO.frame.origin = finalPoint

    //同時放大又旋轉
    let scale = CGAffineTransform(scaleX: 2.0, y: 2.0)
    let rotationAngle = CGFloat(30.0 * .pi / 180.0)
    let rotation = CGAffineTransform(rotationAngle: rotationAngle)
    self.UFO.transform = scale.concatenating(rotation)
}, completion: nil)
```

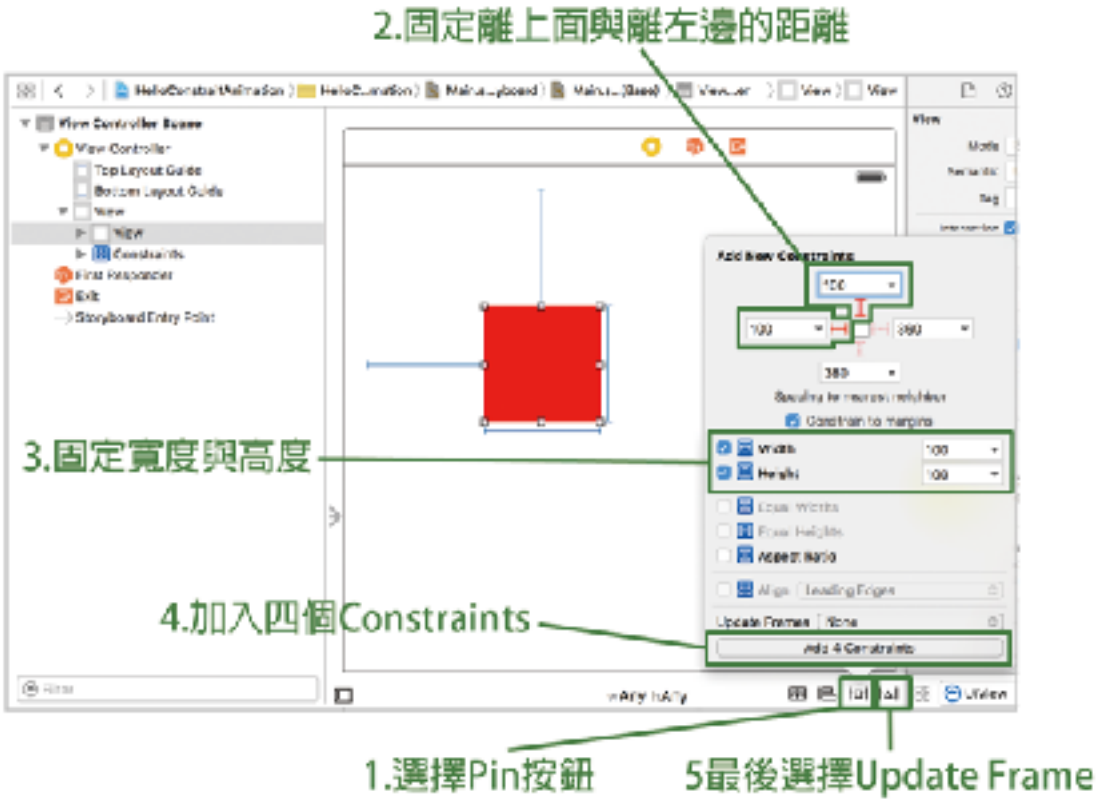
1.這個版本的UIView.animate方法，一共有七個參數：第一個參數還是動畫的時間、第二個參數則是設定延遲多久開始做出動畫的效果。第三個參數設定彈簧動畫的阻尼係數，第四個參數則是設定彈簧動畫初始的速度。第五個參數設定如何做動畫、第六個參數設定要做的動畫，最後第七個參數則是設定動畫完成後要做的事情。這樣設定，動畫就會以彈跳的方式呈現。
2.之前的範例都只是示範單單往x方向或是單單往y方向移動。如果同時要移動x跟y方向的話，請參考本範例，把希望移動的點，用CGPoint初始化方法產生出一個CGPoint座標，用範例的方法同時移動x跟y的方向。
3.之前的範例在變形的過程中，都單單只有示範放大或是單單只有旋轉圖像。如果在動畫的過程中，想要一面放大又一面旋轉圖像的話，請參考上面的程式碼，分別設定放大與縮小的數值，最後用CGAffineTransform的concat方法結合兩種效果。

Autolayout Constraint動畫

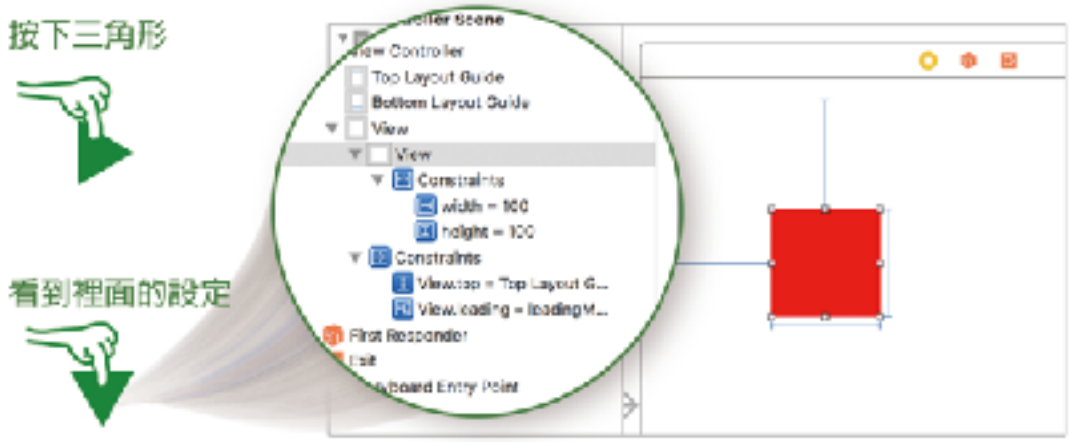
[問題]AutoLayout的Constraint也可以做動畫嗎？
[解答]使用UIView.animate方法，也可以讓AutoLayout的Constraint隨著時間變動
[範例程式碼]HelloConstraintAnimation
[過程解說]
1.請先開一個新的Single View Application專案。在左邊欄選擇Main.Storyboard之後，在右下方搜尋UIView。把這個UIView拉到畫面上，背景選成紅色。



2.在選擇UIView的情況下，幫這個紅色的四方形加上Autolayout的Constraint：固定寬度與高度都為100，且讓這個四方形距離畫面上面與左邊的距離為100。

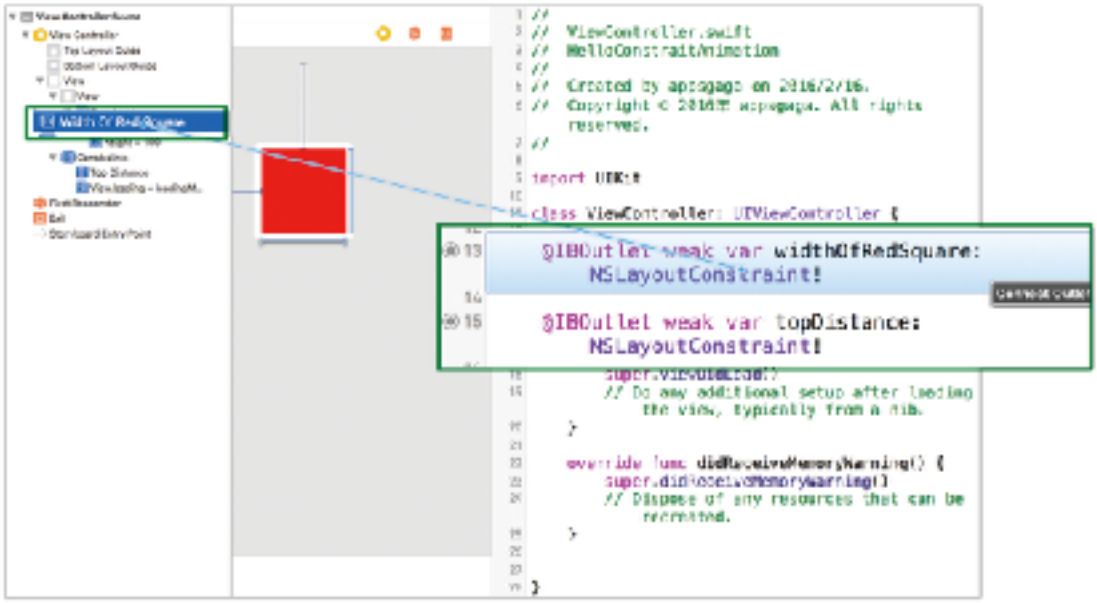


3.請看到Document Outline這欄，把所有的三角形都按開。裡面有四個藍色標示的、剛剛設定的Autolayout的Constraints，分別是固定寬度100、固定高度100，固定離上方距離是100，固定離左邊的距離是100。UIView用來做動畫的方法，也可以用來變化這些數值。這個範例要讓寬度與距離上面的距離做出動畫，於是要連結這兩個Constraint。



4.分別選取[寬度]與[距離上面距離]兩個Constraint。一面按著鍵盤上面的

Control鍵，一面把兩個Constraint連結到程式碼中。變數名稱分別命名成widthOfRedSquare與topDistance。

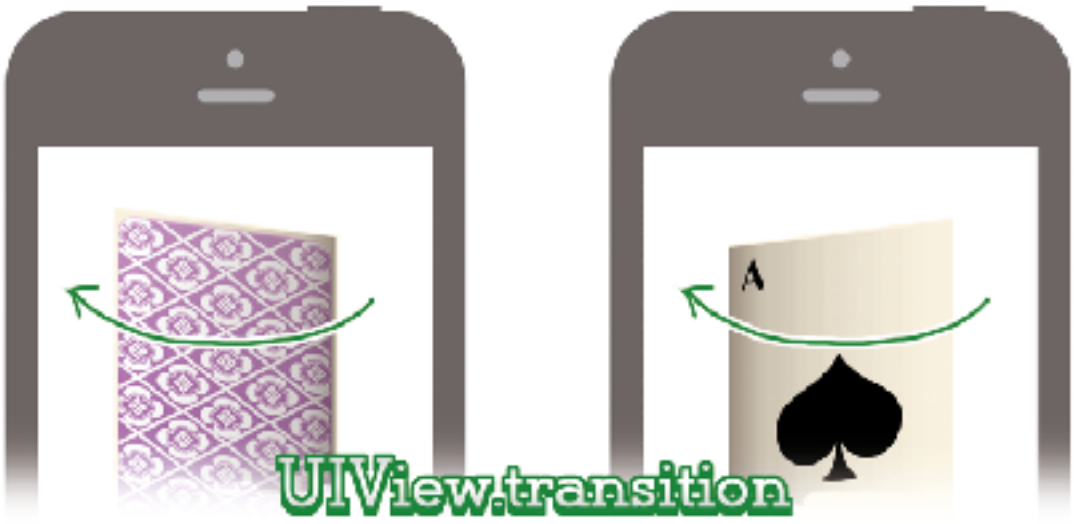


5.回到程式碼，把ViewController類別裡的程式碼改成下面的樣子：

```
class ViewController: UIViewController {

    @IBOutlet weak var widthOfRedSquare: NSLayoutConstraint!
    @IBOutlet weak var topDistance: NSLayoutConstraint!

    override func viewDidAppear(_ animated: Bool) {
        UIView.animate(withDuration: 1.0, animations:{
            //動畫設定
            self.widthOfRedSquare.constant = 200
            self.topDistance.constant = 400
            self.view.layoutIfNeeded()        //記得要加這行程式碼
        })
    }

}
```

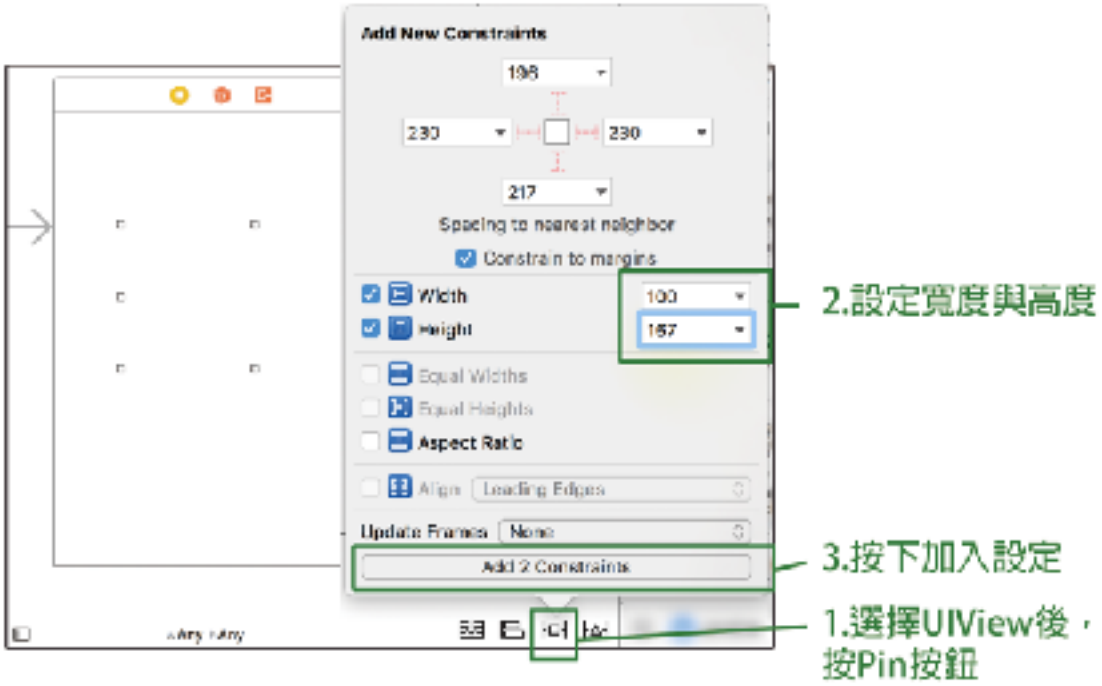
6.和前幾個問題的解答一樣，本範例在viewDidAppear方法中設定動畫效果。設定寬度在動畫結束後變成200、離畫面上方的的距離為400。最後也是最重要的，就是要呼叫self.view.layoutIfNeeded方法，就會發現紅色的四方形寬度變寬、而距離畫面上方的距離也變長了。

翻頁動畫

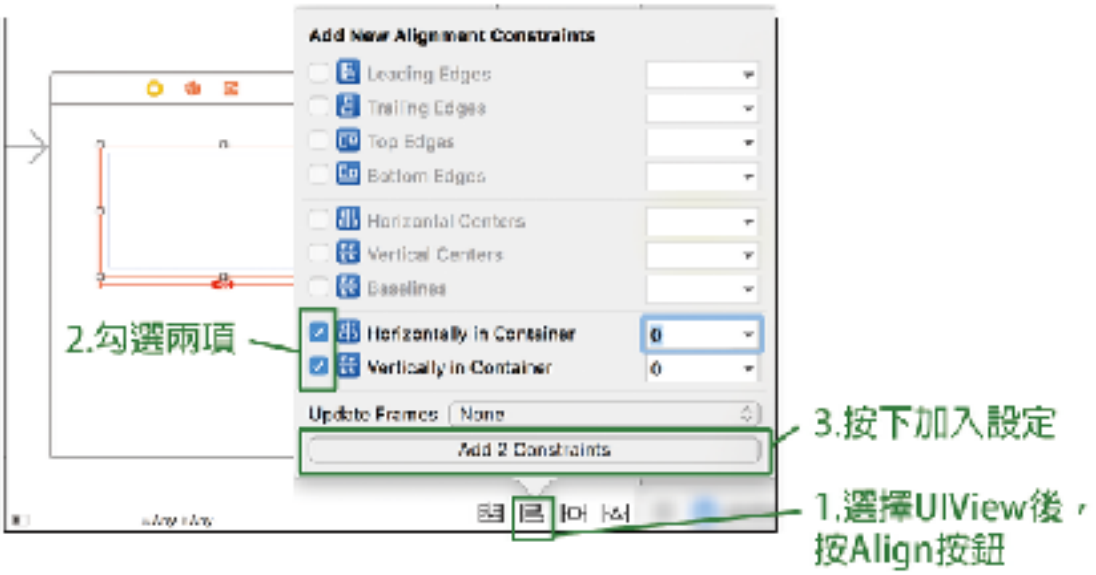


[問題]如何做出翻轉卡片或是翻書的效果
[解答]使用UIView.transition方法，可以做出翻轉卡片或是翻書的效果
[範例程式碼]HelloFlipCard
[過程解說]
1.請先開一個新的Single View Application專案。選擇左邊欄的Assets.xcassets資料夾，把所需的圖檔放進專案中。(本範例的圖檔，在本章附贈程式碼中的MyCards資料夾中。)

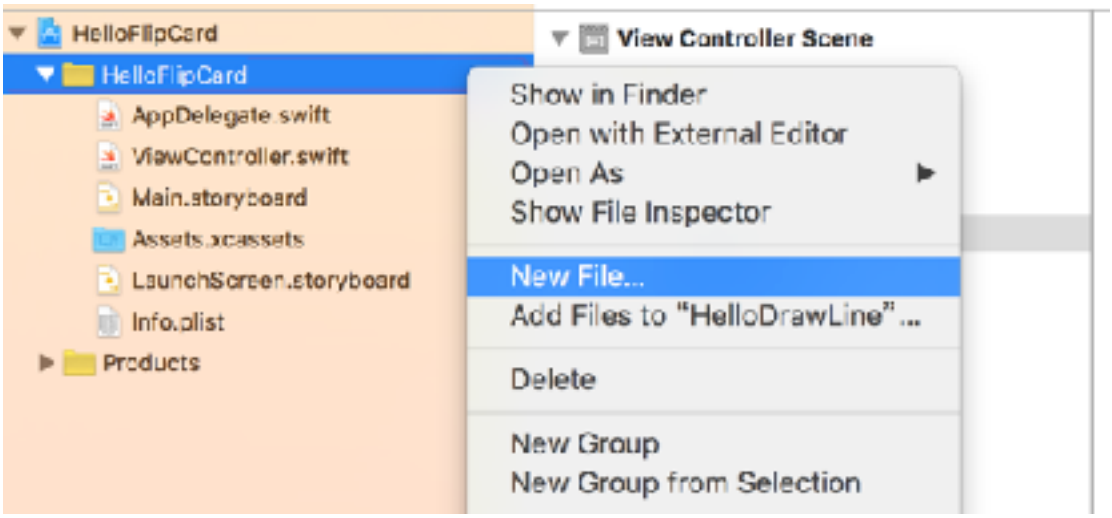
2.設定固定這個UIView的寬度為100，高度在167點的大小。



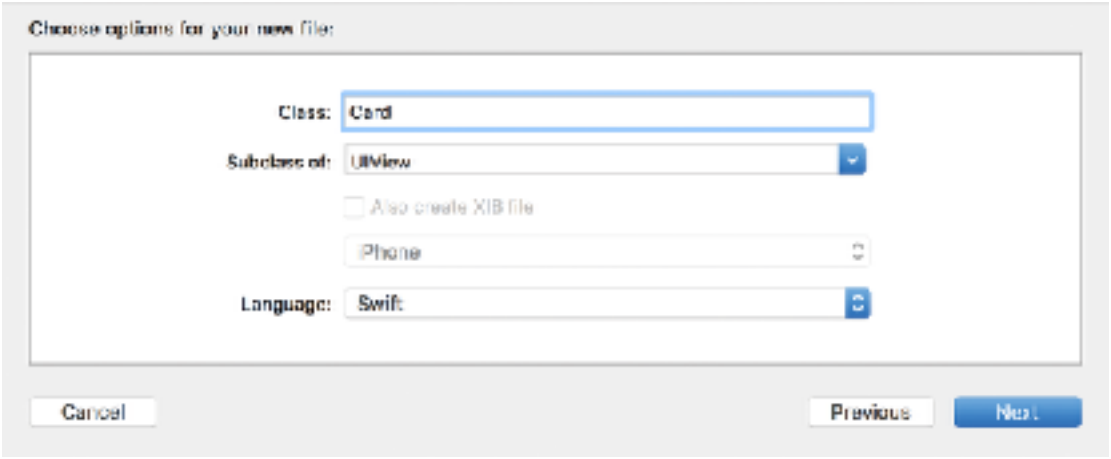
3.設定讓這個UIView置中。接下來要在這個UIView上繪圖。



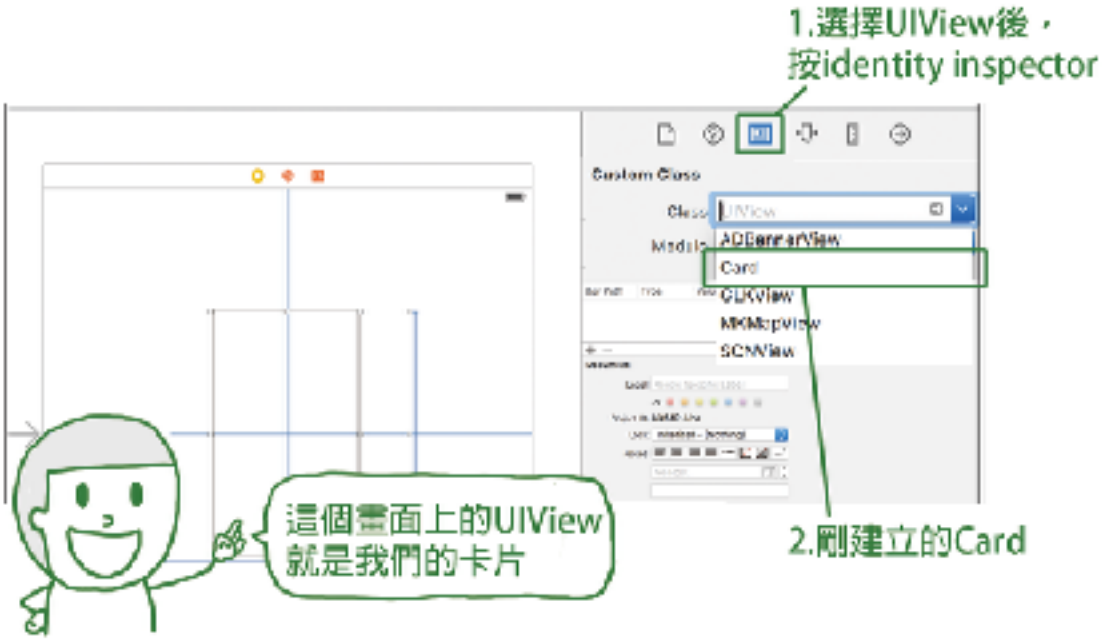
4.選擇左邊欄檔案資料夾，按右鍵新增檔案。



5.選擇iOS > Source > Cocoa Touch Class。
6.新增一個UIView的子類別，範例中命名成Card。
(請注意，子類別的選項選取UIView)



7.回到Main.storyboard。選擇步驟1~3生成的UIView。將其類別設定為Card。



8.請打開Card.swift檔案，找到Card類別。把程式碼改成如下所示：

```
class Card: UIView {

    var isBack = true                    //記錄卡片正面還是反面，一開始是反面
    var front = UIImageView(image: UIImage(named: "Front")) //讀入圖片
    var back  = UIImageView(image: UIImage(named: "Back"))  //讀入圖片

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        self.addSubview(back) //生成這個UIView時，把背面的圖加入到UIView上

    }

    override func touchesEnded(_ touches: Set<UITouch>,
                                          with event: UIEvent?) {
        if isBack == true{
            //翻牌的動畫
            isBack = false
            UIView.transition(from: back, to: front, duration: 1,
                    options: .transitionFlipFromRight, completion: nil)
        }else{
            isBack = true
            UIView.transition(from: front, to: back, duration: 1,
                     options: .transitionFlipFromLeft, completion: nil)
        }
    }
}
```

9.如上程式碼，Card類別裡面，isBack屬性記錄目前卡片是正面還是反面，把撲克牌的正反面圖像都讀入，分別存在front跟back屬性中。
10.在生成Card類別、呼叫init方法時，把背面的圖像back，加入到Card這個UIView的畫面上。

11.覆寫touchesEnded方法，在使用者按完撲克牌之後，呼叫用UIView.transition方法，做出翻牌的動畫。這個方法接受五個參數；第一個參數是翻轉前的圖像、第二個參數是翻轉後的圖像、第三個參數是動畫執行的時間、第四個參數是如何做動畫，第五個參數是做完動畫之後要做什麼。
12.範例中前兩個分別是撲克牌的兩面，所以做的動畫會從撲克牌的某一面整個翻轉到另外一面。第三個參數設定為1，所以翻面的動畫會在1秒鐘的時間裡完成。
13.第四個參數設定動畫會以左右翻面的方式進行。而做完動畫之後第五個參數填nil，所以什麼都不會做。
14.第四個參數除了左右翻面以外，也有下面幾項選擇：

.transitionFlipFromBottom	由下到上翻面
.transitionFlipFromTop	由上到下翻面
.transitionCurlUp	像書本一樣往後翻頁
.transitionCurlDown	像書本一樣往前翻頁
.transitionCrossDissolve	融合效果

如果想要做出像書本一樣的翻頁效果，請把options這個參數，改成上面「.transitionCurlUp」或是「.transitionCurlDown」的選項。

Core Animation

[問題]如何使用CABasicAnimation來做出動畫
[解答]選擇不同的動畫效果之後，設定初始值、結束值，與動畫的時間，就可以做出基本的動畫。
[範例程式碼]HelloCoreAnimation
[過程解說]
1.請照著[UIView動畫]的前兩個步驟，開一個新的專案、匯入圖片，並且在Storyboard上加入UIImageView，固定飛碟圖像的寬度與高度，並且讓飛碟的

位置置中，準備製作動畫。設定完成之後，在viewDidAppear方法中，寫入下面的程式碼：

```
class ViewController: UIViewController {

    @IBOutlet weak var UFO: UIImageView!

    override func viewDidAppear(_ animated: Bool) {

        let animation = CABasicAnimation(keyPath: "opacity")//建立動畫
        animation.fromValue = 1.0    //設定動畫初始值
        animation.toValue = 0.0      //設定動畫結束值
        animation.duration = 1.0     //設定動畫時間

        self.UFO.layer.add(animation, forKey: nil) //加入動畫
        self.UFO.layer.opacity = 0.0//<--這句很重要，要設定動畫結束時的屬性
    }
}
```

2.製作CABasicAnimation動畫很簡單，只要先利用正確的keyPath建立動畫，設定初始值、結束值，與動畫時間。把動畫加入圖像的CALayer圖層，畫面就會出現動畫了。
3.範例中建立的透明度動畫。初始值完全不透明，結束值完全透明，動畫時間為1秒鐘，把動畫加入圖像的CALayer圖層，圖像就會在1秒鐘變透明了。
4.如果沒有寫最後一句程式碼的話，動畫做到最後會重新回復初始值。如果想要圖像在做完動畫之後持續保持在透明的狀態，就要設定self.UFO.layer.opacity = 0。圖像就會保持透明。
5.除了透明度，CABasicAnimation還可以做各種不同的動畫。下面是其中一些動畫的keyPath：

transform.scale	opacity	margin	zPosition
backgroundColor	cornerRadius	borderWidth	bounds
position	frame	shadowColor	shadowOpacity

6.請把上面範例中viewDidAppear方法改成下面的樣子，利用CABasicAnimation做出另外一種動畫：

```
override func viewDidAppear(_ animated: Bool) {

    let animation = CABasicAnimation(keyPath: "bounds.size")//建立動畫
    animation.fromValue = NSValue(cgSize:self.UFO.frame.size)//動畫初值
    animation.toValue = NSValue(cgSize:
                    CGSize(width: 200, height: 200))     //設定動畫結束值
    animation.duration = 1.0    //設定動畫時間

    self.UFO.layer.add(animation, forKey: nil) //加入動畫
    self.UFO.layer.bounds.size = CGSize(width: 200, height: 200)

}
```

7.改動的程式碼建立了的尺寸大小的動畫。初始值為原圖大小，結束值放大成寬度100 、高度100，動畫時間為1秒鐘，把動畫加入圖像的CALayer圖層，圖像就會在1秒鐘變大。
8.如果沒有寫最後一句程式碼的話，動畫做到最後會重新回復初始值。如果想要圖像在做完動畫之後持續保持在放大的狀態，就要設定最後圖像的大小，圖像才會保持在放大的狀態。

UIKit Dynamic動畫(幫視圖加上重力)

[問題]如何幫視圖加上重力與碰撞的效果
[解答]使用UIKitDynamic的方法可以幫視圖加上重力與碰撞等效果
[範例程式碼]HelloUIKitDynamic
[過程解說]
如上示意圖，如果要模擬UIView受重力影響的效果，需要...
1.生成UIDynamicAnimator來做動畫的效果。
2.生成UIDynamicBehavior代表要表現什麼樣的行為：比方說要表現重力行為、碰撞行為等等。
3.把需要表現行為的物件(UIDynamicItem)加到UIDynamicBehavior。
4.再把要表現的行為加到UIDyanmicAnimator，物件就會做出各種不同的效果。

[受重力影響的視圖]

1.請開啟一個新的Single View Application專案，在ViewController類別裡，把程式碼改成下面所示：

```
import UIKit

class ViewController: UIViewController {

    var animator:UIDynamicAnimator!

    override func viewDidLoad() {
        super.viewDidLoad()

        //設定及將受重力引響的方形區域
        let rectArea = CGRect(x: view.frame.width/2-50, y: 0,
                                                width: 100, height: 100)

        //用這個區域製作一個UIView
        let redBox = UIView(frame: rectArea)
        redBox.backgroundColor = UIColor.red
        view.addSubview(redBox)

        //產生UIDynamicAnimator
        animator = UIDynamicAnimator(referenceView: view)

        //產生UIDynamicBehavior
        let gravity = UIGravityBehavior()

        //把視圖(UIView)加入UIDynamicBehavior
        gravity.addItem(redBox)

        //把 UIDynamicBehavior 加入 UIDynamicAnimator
        animator.addBehavior(gravity)
    }
}
```

2.上面的程式碼中，先用CGRect初始化方法設定UIView的範圍。這個方法接受四個參數，分別是範圍的 x座標、y座標、寬度，以及高度。

3.用步驟2產生的範圍製作一個UIView。把這個UIView存在常數redBox裡面，並且把這個UIView的背景色，設成紅色。之後再把redBox加到畫面上。

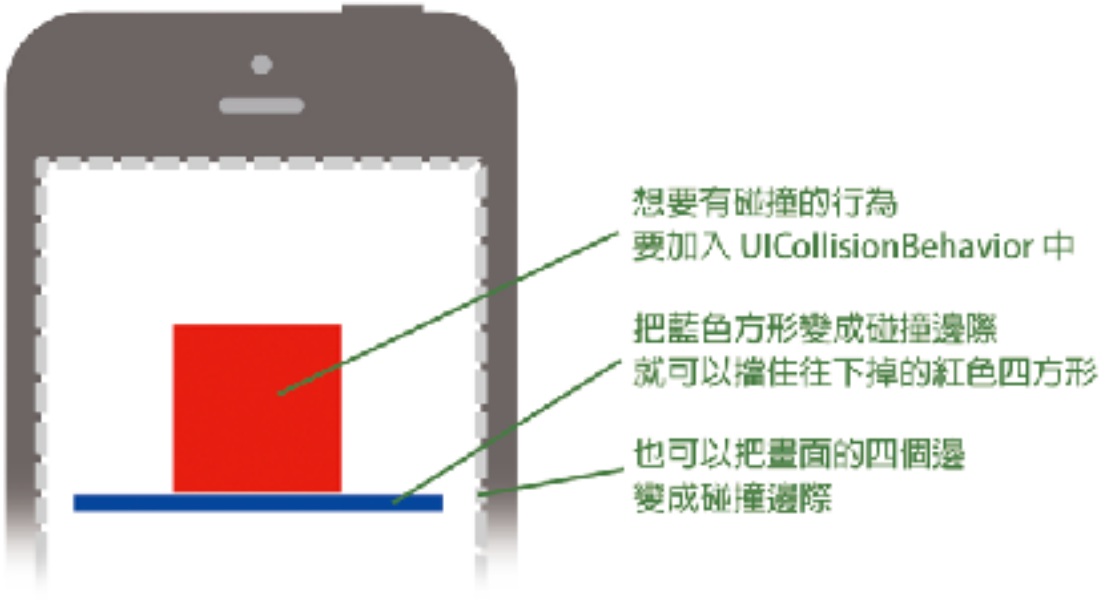
4.如果要讓程式裡紅色的四方形受重力影響的話，要先產生UIDynamicAnimator，於是先幫整個類別加入屬性animator，型別是UIDynamicAnimator。在viewDidLoad裡面，產生UIDynamicAnimator。產生UIDynamicAnimator的方法參數填入view，

意思是，產生的UIDynamicAnimator要控制所有UIViewController的view裡面所有的動畫。把這個UIDynamicAnimator存在屬性animator裡。

5.產生UIDynamicBehavior，決定表現的效果。因為需要的效果是重力，所以生成UIGravityBehavior，存在常數gravity。這個gravity要表現的效果是重力。

6.把紅色 redBox加入UIDynamicBehavior中，再把gravity加入UIDynamicAnimator，紅色四方形就會受重力影響掉下來了。

[碰撞]



程式做到這邊，紅色的四方形會模擬重力效果往下掉、直接掉落到螢幕的外面。如果想要阻擋紅色四方形掉落，請參考加入下面的程式碼：

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

    //設定及將受重力引擎的方形區域
    let rectArea = CGRect(x: view.frame.width/2-50, y: 0,
                                        width: 100, height: 100)

    animator.addBehavior(gravity) ←————————— 之前寫到這邊

    //設定擋住紅色四方形的區域
    let wallArea = CGRect(x: 50, y: 500, width: 300, height: 10)

    //用這個區域做一個 UIView
    let blueBar = UIView(frame: wallArea)
    blueBar.backgroundColor = UIColor.blue
    view.addSubview(blueBar)

    //產生另一個碰撞UIDynamicBehavior
    let collision = UICollisionBehavior()

    //把視圖(UIView)加入碰撞UIDynamicBehavior
    collision.addItem(redBox)

    //將藍色的blueBar的邊際轉換成碰撞邊界感應區
    collision.addBoundary(withIdentifier: "blueBar" as NSCopying,
                                for: UIBezierPath(rect: blueBar.frame))

    //將畫面的四個邊變成碰撞邊界感應區
    //collision.translatesReferenceBoundsIntoBoundary = true

    //把 UIDynamicBehavior 加入 UIDynamicAnimator
    animator.addBehavior(collision)
}
```

1.上面的程式碼產生了一個藍色的UIView阻擋掉落的紅色四方形。先用CGRect初始化的方法設定UIView的範圍。
2.用上個步驟產生的範圍做一個UIView。把這個UIView存在常數blueBar裡面，並且把這個UIView的背景色，設成藍色。之後再把blueBar加到畫面上。
3.如果要讓程式有碰撞的行為，要先產生UICollisionBehavior，存在常數collision。
4.把紅色 redBox加入UIDynamicBehavior中，再把collision加入UIDyanmicAnimator，紅色四方形就除了會模擬重力影響以外，還會有碰撞的效果。
5.不過這樣設定之後，紅色四方形碰到藍色的blueBar還是會直接穿越。要使用collision.addBoundary方法，把blueBar轉換成碰撞邊界感應區，藍色四方形才會阻擋紅色四方形掉落。collision.addBoundary方法接受兩個參數，第一個參數是幫藍色的blueBar取的標示ID，第二個參數是藍色四方形的邊界。
6.如果想讓畫面的四個邊際當作牆壁阻擋紅色四方形的話，請把藍色四方形的

程式碼全部註釋起來，使用collision.translateReferenceBoundsIntoBoundary = true。把畫面的四個邊際當作碰撞邊界感應區。

[吸附效果]

UIKitDynamic還可以做出好多特別的效果，請把之前的類別的程式碼改成下面的樣子，可以做出吸附的效果：

```
override func viewDidLoad() {
    super.viewDidLoad()
    let rectArea = CGRect(x: view.frame.width/2-50, y: 0,
                                              width: 100, height: 100)
    let redBox = UIView(frame: rectArea)
    redBox.backgroundColor = UIColor.red
    view.addSubview(redBox)
    animator = UIDynamicAnimator(referenceView: view)

    //生成吸附行為(UISnapBehavior)
    let snap = UISnapBehavior(item: redBox, snapTo: view.center)

    //把 UIDynamicBehavior 加入 UIDynamicAnimator
    animator.addBehavior(snap)
}
```

要做出吸附的效果，就生成UISnapBehavior。生成的方法裡面接受兩個參數：第一個是要做效果的物件，第二個參數是要吸附位置的座標。範例中的設定，讓紅色的四方形吸到螢幕的中間。