

本PDFは、2016年10月25日に発売された、書籍「これからつくる iPhoneアプリ開発入門 ～Swiftではじめるプログラミングの第一歩～」の**応用編**です。

## 本PDFで学習する前に

iOSアプリ開発が初めての方は、書籍「これからつくる iPhoneアプリ開発入門 ～Swiftではじめるプログラミングの第一歩～」を学習されたのちに、本PDFに取り組むと効果的です！

書籍は、次のネットショップから購入することができます。

Amazonでの購入：

<http://amzn.to/2dKYSMk>

楽天ブックスでの購入：

<http://bit.ly/2e42nDh>

SBクリエイティブでの購入：

<http://bit.ly/2edAN4i>

## 本PDFに関するお問い合わせ

お問い合わせは、次の公式サポートサイトよりお願いいたします。

公式サポートサイト：お問い合わせ

<https://swiftbg.github.io/swiftbook/contact/>

誤字脱字や改善要望がございましたらご連絡ください。ご要望の内容に関して検討し、PDFへ順次、反映していきたいと思っております。

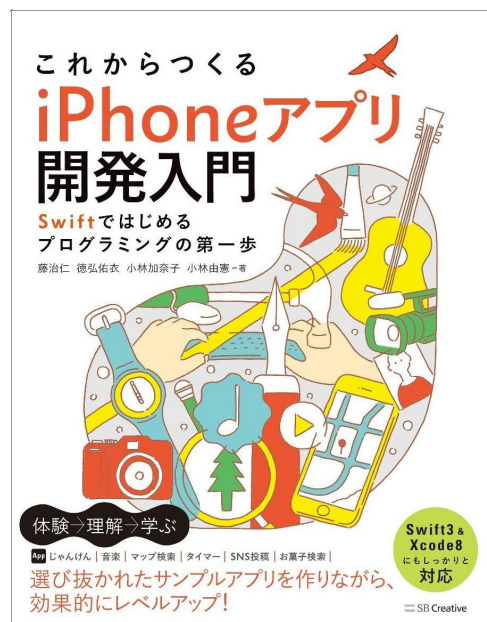
PDFが更新されたときは、公式サポートサイト、SNS等で告知させていただきます。

公式サイトSNS（フォローして頂けると、情報の入手が早くなります。）

Facebookページ：<https://www.facebook.com/swiftbgbook/>

Twitterアカウント：[https://twitter.com/swift\\_bg](https://twitter.com/swift_bg)

YouTubeで、「Swiftビギナーズ倶楽部」と検索して、チャンネル登録！



# 応用編 Lesson 1 対戦型じゃんけんアプリ

---

## このレッスンでできるようになること

「1日目Lesson 3 じゃんけんアプリを作ろう」の応用編として対戦型じゃんけんアプリに改造していきます。

これまでのじゃんけんアプリにSwiftの列挙型であるenumを導入しソースコードの可視化を図ります。

また、enum導入後に新しいAutolayoutを用いてUIボタンを均等の幅となるようにします。

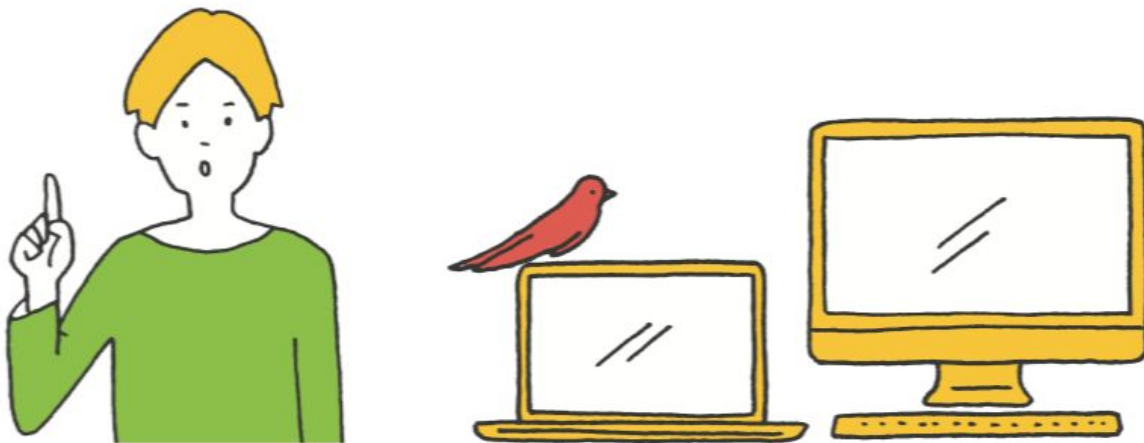
---

Lesson 1-1 完成イメージを確認しよう

Lesson 1-2 enumでスッキリ

Lesson 1-3 画面を作成しよう

Lesson 1-4 対戦型に進化しよう



# Lesson 1-1 完成イメージを確認しよう

【このレッスンで学ぶこと】	【できるようになること】
じゃんけんアプリをどのように改造するのかを確認します。 対戦型に変更するときの変更ポイントを確認します。	対戦型じゃんけんの処理のイメージをできるようになります。 また、工夫のポイントについて理解できます。

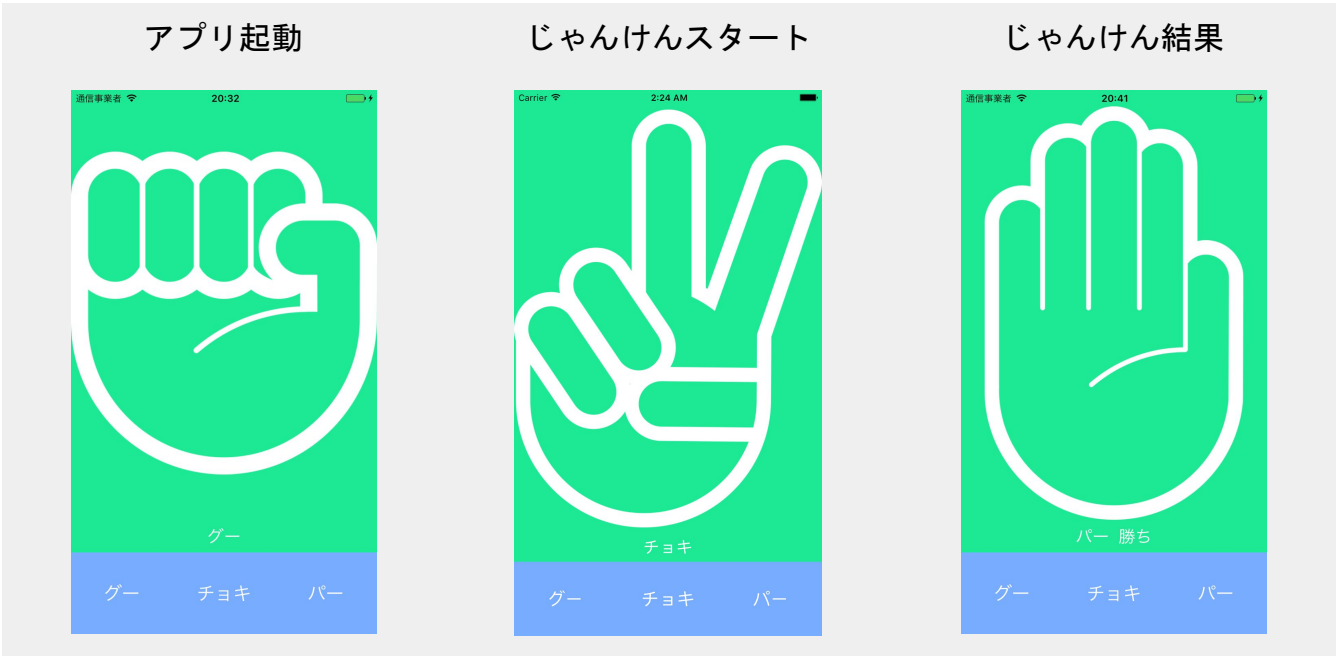
## 1:対戦型じゃんけんアプリとは

この章では、「1日目Leeson 3 じゃんけんアプリを作ろう」で学んだことを更にカスタマイズして、対戦型じゃんけんアプリに改造していきます。  
アプリが起動すると今までどおり初期画面となります。

次に「グー」「チョキ」「パー」のいずれかのボタンをタップするとアニメーションを開始します。  
また、「グー」「チョキ」「パー」のアニメーションが切り替わる対戦中に、いずれかのボタンをタップするとアニメーションが停止し対戦結果が表示されます。

またアニメーション停止するとき最後に表示した結果のまま停止すると見て何を出せば勝てるのかがわかってしまいます。必ずじゃんけん結果表示するときは最後にもう1度表示更新してから結果を決定し対戦結果を表示します。

### ◎対戦型じゃんけんアプリの完成イメージ



# Lesson 1-2 enumでスッキリ

「このレッスンで学ぶこと」	「できるようになること」
enumを利用しソースコードを整理します。 Swiftならではのenumの活用方法を紹介しま す。	enumを使ってわかりやすいソースコードを作 ることができるようになります。

## 1:enumとは

enumことを列挙型と一般的に呼ばれています。  
「1日目Leeson 3じゃんけんアプリを作ろう」では、answerNumberを0なら「グー」、0なら「チョ  
キ」、2なら「パー」と数値と「グー・チョキ・パー」を指定していました。

グー	チョキ	パー
0	1	2

しかし、ソースコードだけを見ると、どの値がどのじゃんけん結果となるのかがわかりづらいです。  
ソースコードを見た時に、都度、0はグー、1はチョキ、2はパーとプログラマーが頭のなかで変換する  
必要があります。

この作業はとても、プログラマー自身に負荷がかかります。  
このサンプルコードは短いコードですし、簡単な3択としてすぐに思い出せるかもしれませんが。  
でも、実際は、このサンプルコードよりはるかに多くのコードを書いてアプリケーションを開発して  
いきます。  
どんな記憶力が良いプログラマーでも、時間が経てば人間は数字の意味を忘れてしまいます。  
さらに、コードを修正する人が変われば、「この数字の意味は何だろう？」と疑問に思います。  
プログラマーの世界では、一般的にこのような数字を「マジックナンバー」と言ったりもします。  
プログラミングの世界の「マジックナンバー」は、プログラムを書いた本人以外は、分からない数字  
のことを指します。

そこでenumを利用することによって、じゃんけん結果と値を紐付けて定義し、わかりやすくするこ  
とができます。  
enumを利用する際には、変数の型（例：String型）と同様にenum定義名を型として指定します。  
また、値を参照するときは先頭に「.（ドット）」を記述する必要があります。

### ◎enumの記述例 1

```
// jankenという名前のenum定義
enum janken {
  // 値：グー
  case gu
  // 値：チョキ
  case choki
  // 値：パー
  case pa
}

// 変数answerはenum定義jankenで定義し、初期値はグーとする
var answer : janken = .gu
```

また各値の詳細を定義することも可能です。

以下の例ときは、「.gu」なら0、「.choki」なら1という風に値をつけることも可能です。

値はString型など文字列にすることも可能です。

enumの値を取得するときは、「.rawValue」メソッドを利用します。

### ◎enumの記述例 2

```
// jankenという名前のenum定義
// 値はUInt32の値で保持する
enum janken : UInt32{
  // 値：グー(値は0)
  case gu = 0
  // 値：チョキ(値は1)
  case choki = 1
  // 値：パー(値は2)
  case pa = 2
}

// 変数answerはenum定義jankenで定義し、初期値はグーとする
var answer : janken = .gu

// 変数answerの値を表示する
print(answer.rawValue) → 0が取得できる
```

最後にenum内部にメソッドを記述することができます。

メソッドを記述することによって値を表現方法を変化することが可能です。

以下の例では、UIに表現するためにユーザーがわかる文字に変換しています。

### ◎enumの記述例 3

```
// jankenという名前のenum定義
// 値はUInt32の値で保持する
enum janken : UInt32{
  // 値：グー(値は0)
  case gu = 0
  // 値：チョキ(値は1)
  case choki = 1
  // 値：パー(値は2)
  case pa = 2

  // 値を文字に変換するメソッド
}
```

```
func string() -> String {
  switch self {
    case .gu:
      return “グー”
    case .choki:
      return “チョキ”
    case .pa:
      return “パー”
    }
  return “不明”
}

// 変数answerはenum定義jankenで定義し、初期値はグーとする
var answer : janken = .gu

// 変数answerの値を文字に変換するメソッドを使って表示する
print(answer.string()) →グーが取得できる
```

## 2:enumを使ってコードを整理してみよう

### (2-1) enumを定義します

◎enumを定義したソースコード

```

25  @IBOutlet weak var answerLabel: UILabel!
26
27  // じゃんけんのenum
28  enum janken : UInt32 {
29    // グーの定義
30    case gu = 0
31    // チョキの定義
32    case choki = 1
33    // パーの定義
34    case pa = 2
35  }
36
37  // じゃんけん (数字)
38  var answerNumber:UInt32 = 0

```

追加

まずは、enumを定義します。

「gu」は「0」、「choki」は「1」、「pa」は2を代入します。

定義をした以降は「.gu」のように、ドットを付けて参照することができます。



## (2-2) enumを利用します

### ◎enumを利用するソースコード

```
37 // じゃんけん (数字)
38 var answerNumber: janken = .gu ①
39
40 @IBAction func shuffleAction(_ sender: AnyObject) {
41
42     // 新しいじゃんけんの結果を一時的に格納する変数を設ける
43     // arc4random_uniform()の戻り値がUInt32なので明示的に型を指定
44     var newAnswerNumber: UInt32 = 0
45
46     // ランダムに結果を出す、前回の結果と異なる場合のみ採用
47     // repeat は繰り返しを意味する
48     repeat {
49
50         // 0,1,2の数値をランダムに算出 (乱数)
51         newAnswerNumber = arc4random_uniform(3)
52
53         // 前回と同じ結果のときは、再度、ランダムに数値をだす
54         // 異なる結果のときは、repeat を抜ける
55     } while answerNumber.rawValue == newAnswerNumber ②
56
57     // 新しいじゃんけんの結果を格納 ③
58     if (newAnswerNumber == janken.gu.rawValue) {
59         answerNumber = .gu
60     } else if (newAnswerNumber == janken.choki.rawValue) {
61         answerNumber = .choki
62     } else if (newAnswerNumber == janken.pa.rawValue) {
63         answerNumber = .pa
64     }
65
66     if answerNumber == .gu {
67         // グー
68         answerLabel.text = "グー"
69         answerImageView.image = UIImage(named: "gu")
70     } else if answerNumber == .choki {
71         // チョキ
72         answerLabel.text = "チョキ"
73         answerImageView.image = UIImage(named: "choki")
74     } else if answerNumber == .pa {
75         // パー
76         answerLabel.text = "パー"
77         answerImageView.image = UIImage(named: "pa")
78     }
79
80 }
81
82 }
83 }
```

- ①じゃんけん結果を格納している変数「answerNumber」をUInt32からenum定義したjanken型に置き換えます。
- ②「answerNumber」の値（数値）は「.rawValue」メソッドを利用して取得します。
- ③新しいじゃんけんの結果をif文を利用して格納します。
- ④今まで数値で「0」「1」「2」と指定していましたが、「.gu」「.choki」「.pa」とenumに置き換えます。

じゃんけんの結果を数値でコードに記述しているより遥かに読みやすく（可読性）なっていることがわかります。

### (2-3) enumにメソッドを定義して使ってさらに整理します

ソースを確認するとさらに画面に表示するための「グー」と変換する箇所と、画像を表示するために「gu」に変換している箇所があります。

enumにメソッド定義することによってenum定義した値とプログラムの中にif文分岐の処理内で書かれているをenum定義に集約することによってenumの値を追加した時の影響箇所を少なくすることにつながります。不具合の発生防止に役立ちます。

#### ◎enumにメソッドを定義したソースコード

```
27 // じゃんけんのenum
28 enum janken : UInt32 {
29     // グーの定義
30     case gu = 0
31     // チョキの定義
32     case choki = 1
33     // パーの定義
34     case pa = 2
35
36     // enumから文字列に変換
37     func string() -> String{
38         // じゃんけんの結果の文字を配列として定義
39         let text = ["グー", "チョキ", "パー"]
40
41         // enumの値をInt型に型変換
42         let index = Int(self.rawValue)
43
44         // 文字列を返す
45         return text[index]
46     }
47
48     // enumから画像に変換
49     func imageName() -> String{
50         // じゃんけんの結果の画像名を配列として定義
51         let named = ["gu", "choki", "pa"]
52
53         // enumの値をInt型に型変換
54         let index = Int(self.rawValue)
55
56         // 画像名を返す
57         return named[index]
58     }
59 }
```

追加

enum定義内にメソッドを追加します。

「self」を利用すると、自分自身のインスタンスを参照しますので、enumに代入している値を取得できます。

ここでは変換後の文字を配列定義し、「self.rawValue」とすることでenumの値を取得しています。じゃんけん結果の文字を返却するメソッドと、画像名を返却するメソッドを追加しています。それぞれ、最後に配列の添字として利用しenumに代入している値と対応する文字を返しています。



## ◎enumメソッドを利用したソースコード

```
61 // じゃんけん (数字)
62 var answerNumber:janken = .gu
63
64 @IBAction func shuffleAction(_ sender: AnyObject) {
65
66     // 新しいじゃんけんの結果を一時的に格納する変数を設ける
67     // arc4random_uniform()の戻り値がUInt32なので明示的に型を指定
68     var newAnswerNumber:UInt32 = 0
69
70     // ランダムに結果を出す、前回の結果と異なる場合のみ採用
71     // repeat は繰り返しを意味する
72     repeat {
73
74         // 0,1,2の数値をランダムに算出 (乱数)
75         newAnswerNumber = arc4random_uniform(3)
76
77         // 前回と同じ結果のときは、再度、ランダムに数値をだす
78         // 異なる結果のときは、repeat を抜ける
79     } while answerNumber.rawValue == newAnswerNumber
80
81     // 新しいじゃんけんの結果を格納
82     if (newAnswerNumber == janken.gu.rawValue) {
83         answerNumber = .gu
84     } else if (newAnswerNumber == janken.choki.rawValue) {
85         answerNumber = .choki
86     } else if (newAnswerNumber == janken.pa.rawValue) {
87         answerNumber = .pa
88     }
89
90     // じゃんけんから文字列を取り出す
91     answerLabel.text = answerNumber.string()
92     // じゃんけんから画像を取り出す
93     answerImageView.image = UIImage(named: answerNumber.imageName()) 修正
94 }
95 }
```

じゃんけんの結果を画面に表示する箇所のif文分岐を削除します。

先程、janken型に定義した「string」「imageName」メソッドを利用して、じゃんけん結果の文字列と画像名を取得するコードに修正します。

enumのメソッドで取得した結果を代入するシンプルなプログラムになりました。



## Lesson 1-3 画面を作成しよう

[このレッスンで学ぶこと]	[できるようになること]
新しいAutoLayoutの設定方法を学びます。	均等なサイズのUIパーツのAutoLayoutの設定ができるようになります。

### 1: 「じゃんけんをする！」ボタンを削除しよう

「じゃんけんをする！」ボタンの箇所に「グー」、「チョキ」、「パー」ボタンを配置します。  
「じゃんけんをする！」ボタンを削除します。

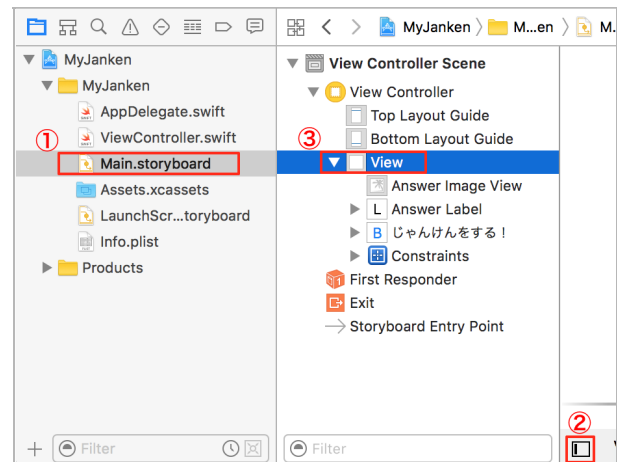
#### (1-1) Main.storyboardを選択します

- ①  Main.storyboardを選択します。
- ② もし、[Document Outline] が表示されていないときは、 [Document Outline] ボタンをクリックします。

[Document Outline] で、 [View Controller Scene] → [View Controller] と▼をクリックしてツリーを展開します。

- ③ [View] を選択します。

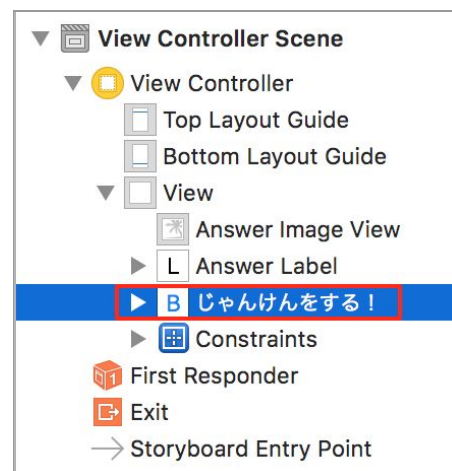
◎Main.storyboard選択操作画面



#### (1-2) 「じゃんけんをする！」ボタンを削除します

[Document Outline] から「じゃんけんをする！」を選択します。  
delete キーを押して削除します。

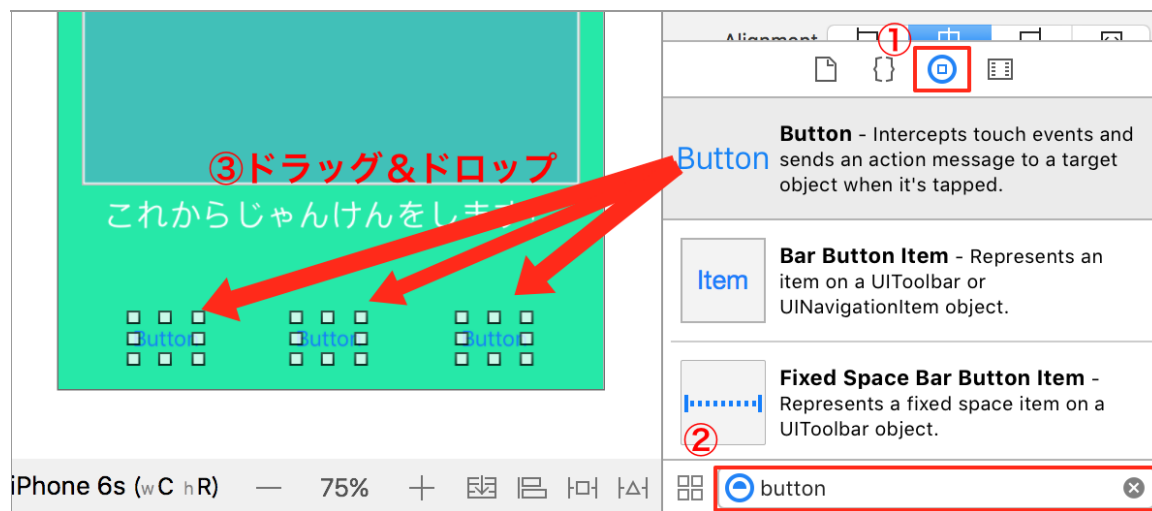
◎「じゃんけんをする！」を削除する操作画面



## 2: 「グー」、「チョキ」、「パー」 ボタンを配置しよう

### (2-1) Buttonを配置します

◎Button追加操作画面



- ① [Object Library] を選択してください。
- ② 検索窓に「button」と入力後に「enter」キーを実行しパーツを検索します。
- ③ 表示されたButtonを、Storyboardの「じゃんけんをする！」ボタンがあった箇所にドラッグ&ドロップし、ボタンを横一行に3個配置します。

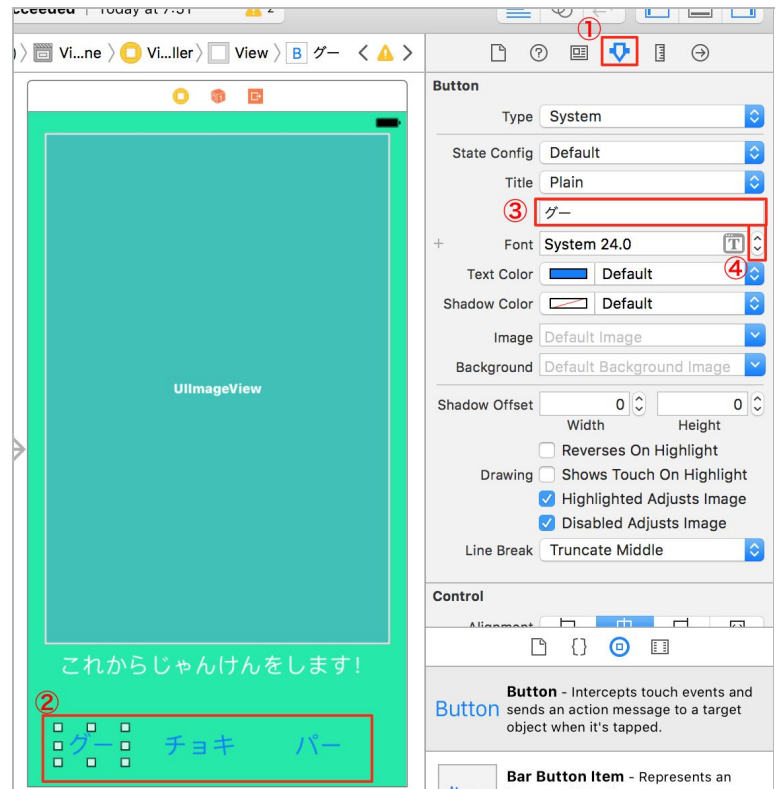
## (2-2) タイトルを設定します

ボタンのタイトルをデフォルトの「Button」から「ゲー」、「チョキ」、「パー」に変更します。

- ① [Attributes inspector] を選択してください。
- ② Buttonを選択します
- ③ [Title] にそれぞれ「ゲー」、「チョキ」、「パー」と入力します。
- ④ フォントサイズを24に変更します。

フォントサイズを大きくしたことで「ゲー」「チョキ」「パー」の文字が「...」と表示された場合は、Buttonパーツの表示エリアを広げて文字全体が見えるようにしてください。

◎タイトル設定操作画面

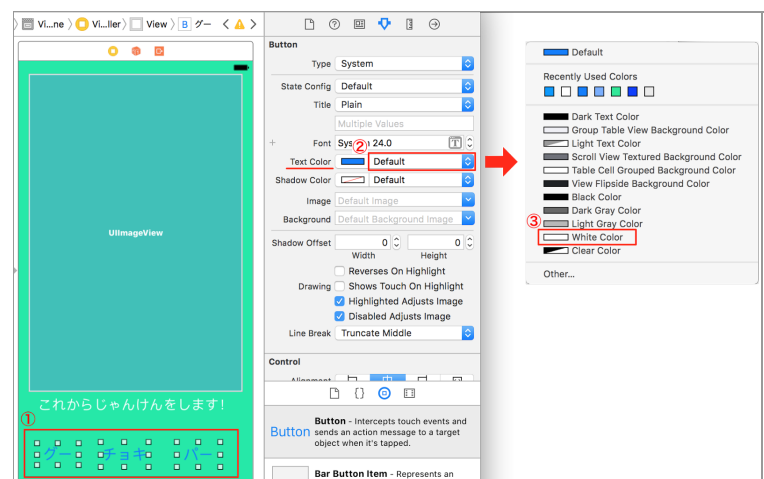


## (2-3) 配色を変更します

まずは、ボタンのタイトル色を変更します。

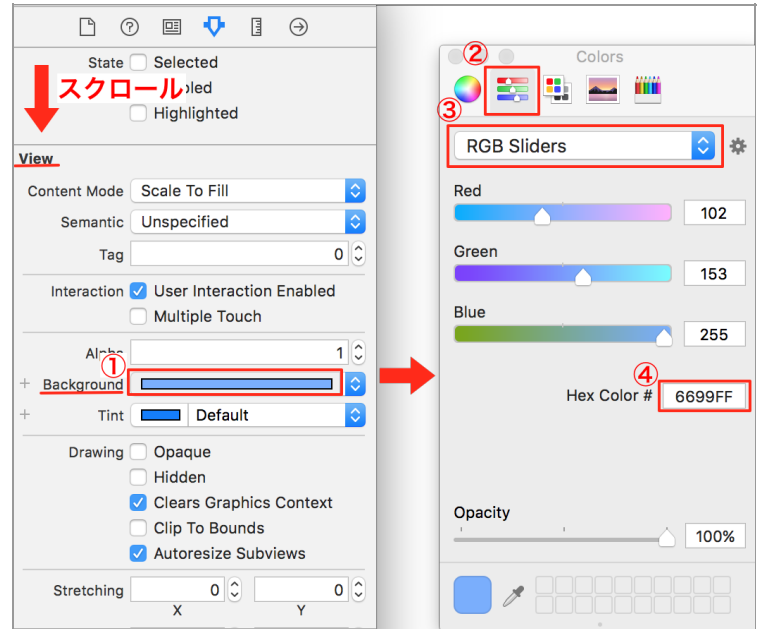
- ① Buttonを、「command ⌘」キーを押しながら、「ゲー」、「チョキ」、「パー」を選択します。
- ② [Text Color] の「Default」を選択してください。
- ③ ポップアップした一覧より「White Color」を選択してください。

◎配色変更操作画面 1



次に、ボタンの背景色を変更します。

- ① [View] → [Background] 横の選択色部分を選択します。
- ② [Color Palettes] タブを選択します。
- ③ リストより「RGB Sliders」を選択します。
- ④ 最後に [Hex Color #] 「6699FF」と入力します。



### 3:AutoLayoutで、レイアウトを整えよう

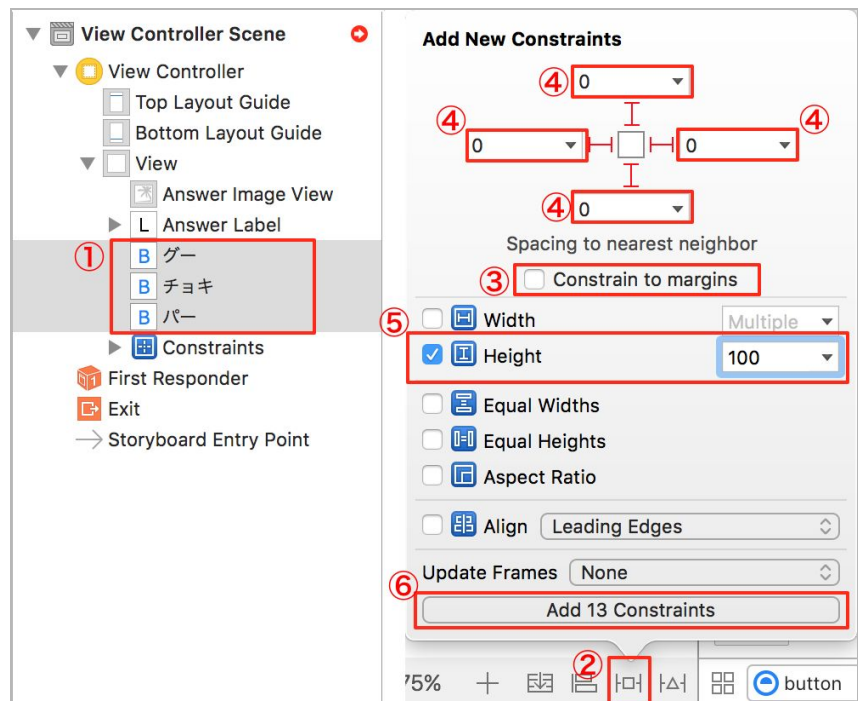
AutoLayoutとは、画面の大きさに合わせて、配置したUIパーツを自動的にレイアウトする機能です。

#### (3-1) Pinを設定します

#### ◎ 3 つボタン AutoLayout設定操作画面

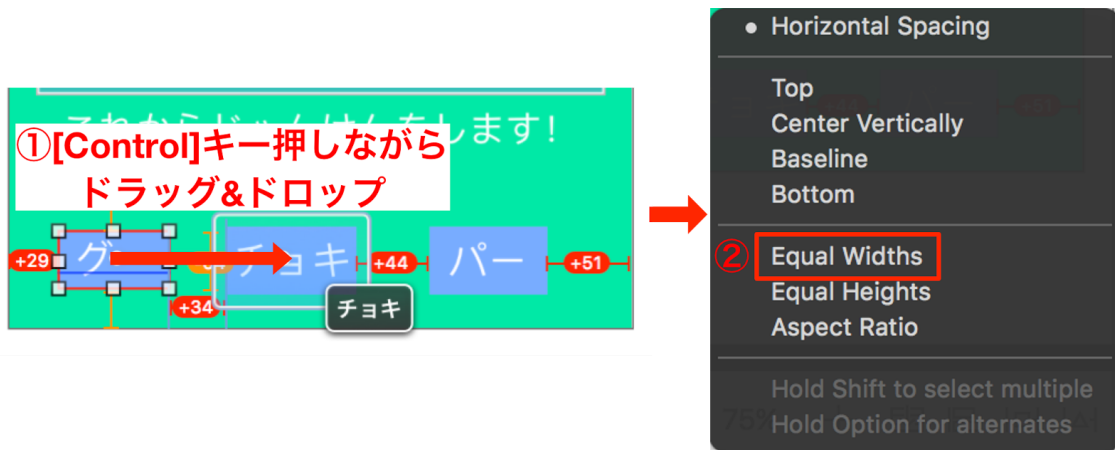
「ゲー」、「チョキ」、「パー」の3つのボタンを画面下部に均等に配置するための余白と高さを設定します。

- ① [Document Outline] の「ゲー」、「チョキ」、「パー」を「command ⌘」キーを押しながら、同時に選択します。
- ② [Pin] をクリックします。  
[Pin] では、UIパーツのサイズや、他のUIパーツとの距離を設定できます。  
[Add New Constraints] というメニューが表示されます。
- ③ [Constrain to margins] のチェックを外します。
- ④ [上下左右の余白] に「0」と入力します。
- ⑤ [Height] をチェック入れて、「100」と入力します。
- ⑥ [Add 13 Constraints] を選択して、制約を追加します。



### (3-2) 3つのボタンの幅を均等になるように設定します

◎ 3つのボタン幅を均等になる AutoLayout設定操作画面



①「グー」ボタンから「チョキ」ボタンへ [control] キーを押しながらドラッグ&ドロップ操作をします。


②ポップアップした画面より幅を均等になるように「Equal Widths」を選択します。  
同じ操作を「グー」ボタンから「パー」ボタンも行ってください。

### (3-3) 「ビューの誤配置」(viewmis placement)を解消します

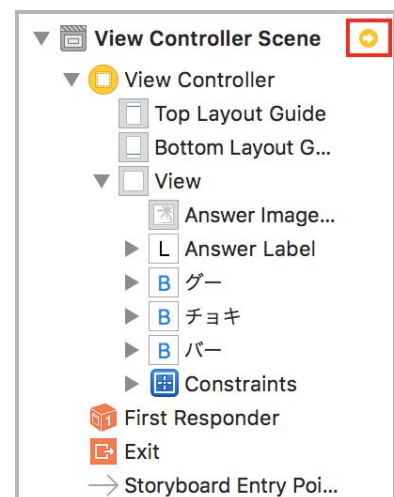
[Document Outline]に  が表示されます。

これは、AutoLayout(制約)として設定した位置とパーツの実際の位置が一致していない状況を表しています。

これを、「ビューの誤配置」(viewmis placement)と言います。


 マークをクリックして、「ビューの誤配置」を解消します。

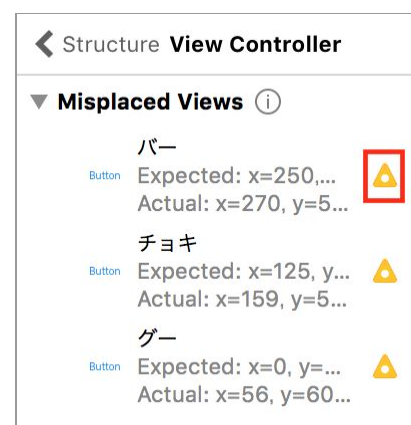
#### ◎警告の表示



#### ◎Misplaced Views

[Document Outline]がスライドされて、「Misplaced Views」が表示されます。

さらに、 をクリックします。

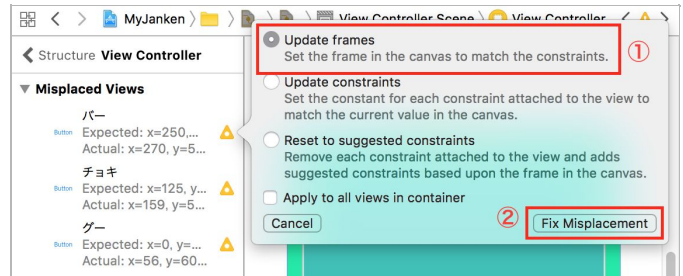




## ◎位置のずれ解消ダイアログ

ラジオボタン「Update freams」が選択されています。

その下にある文言「Set the frame in the canvas to match the constraints.」は、「画面の表示を設定した制約に合致します」という意味です。



「Fix Misplacement」をクリックして、位置のずれを解消します。  
ひとつめの警告が解消できたら、残る2つの警告も同じ手順で解消しておきましょう。

## ◎位置のズレ解消後のレイアウト

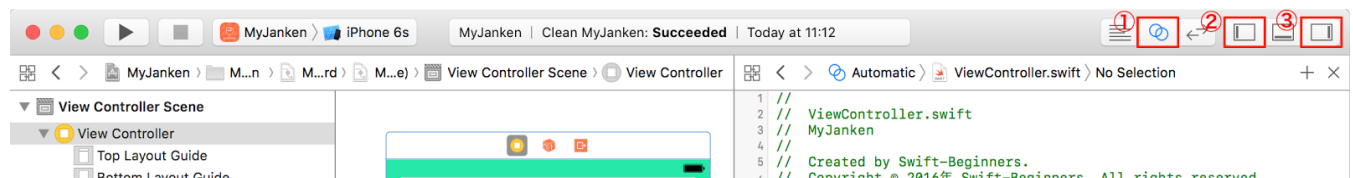
3つの警告が解消できたら、「ゲー」「チョコキ」「パー」の高さ「100」ポイント、横幅いっぱいに背景色が設定されるようになります。

右のキャプチャとレイアウトが一緒になっているか、確認してください。



## 4:関連付けをしよう

### ◎Assistant Editor切り替え操作画面



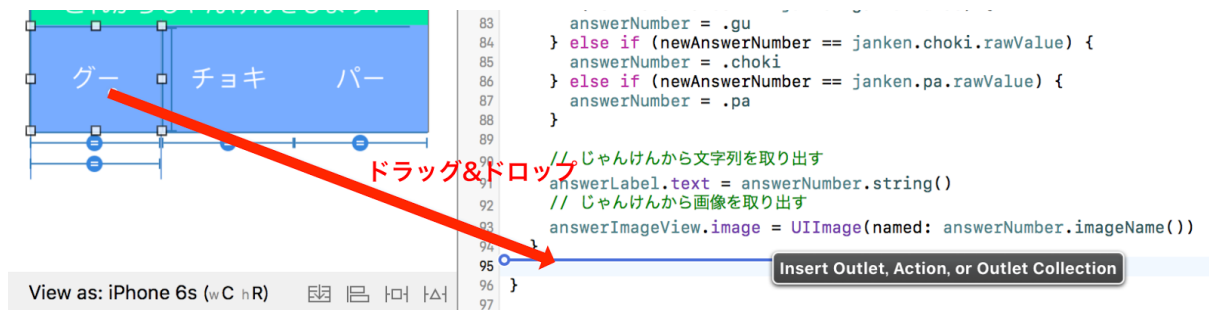
① [Assistant Editor] ボタンをクリックして、エディタを表示します。

② [Navigator] ボタンをクリックして [Navigator] を閉じます。

③ [Utilities] ボタンをクリックして [Utilities] を閉じます。

ここから、関連付けを行います。

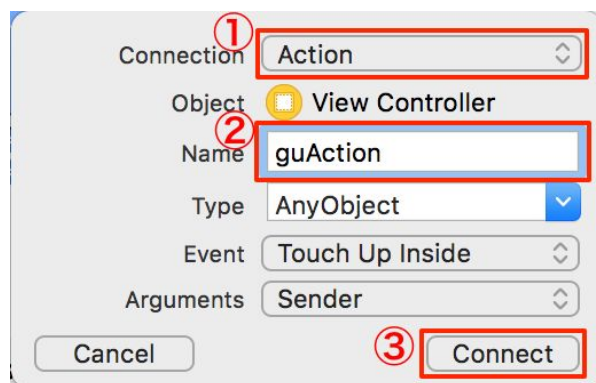
## ◎「グー」ボタン関連付け操作画面 1



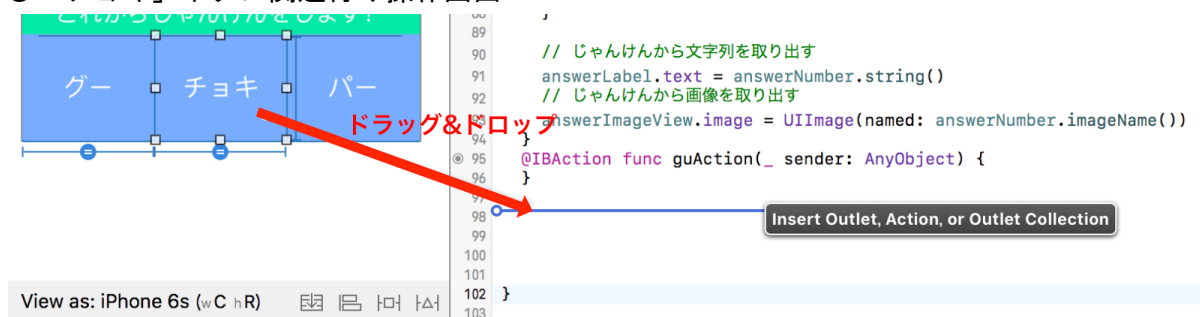
「グー」ボタンを選択して、[control] キー を押しながら、右側のコードにドラッグ & ドロップします。  
一番下の「}」の上でドラックを離して配置してください。

## ◎「グー」ボタン関連付け操作画面 2

画面のようなポップアップが表示されます。  
① [Connection] を「Action」を選択します。  
② [Name] を「guAction」と入力します。  
③ [Connect] を選択します。



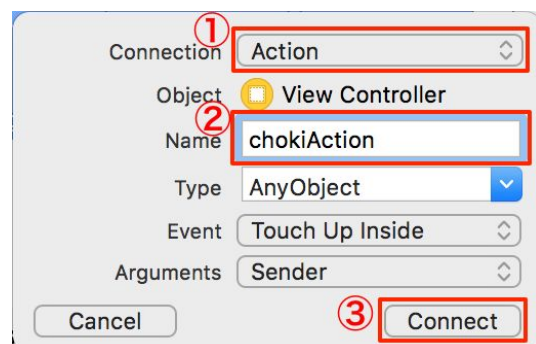
## ◎「チョキ」ボタン関連付け操作画面 1



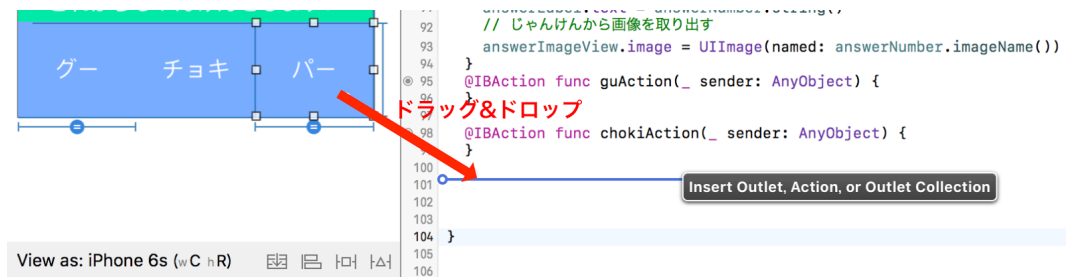
同じく「チョキ」ボタンを選択して、[control] キー を押しながら、右側のコードにドラッグ & ドロップします。

## ◎「チョキ」ボタン関連付け操作画面 2

画面のようなポップアップが表示されます。  
① [Connection] を「Action」を選択します。  
② [Name] を「chokiAction」と入力します。  
③ [Connect] を選択します。



## ◎「パー」ボタン関連付け操作画面 1

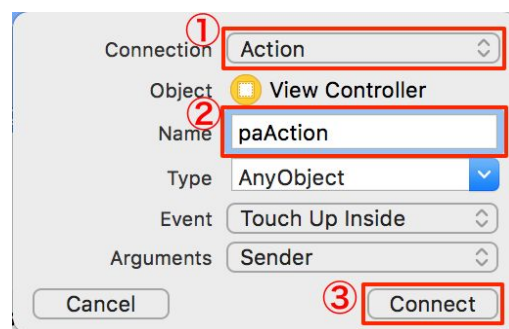


「パー」ボタンを選択して、[control] キーを押しながら、右側のコードにドラッグ&ドロップします。

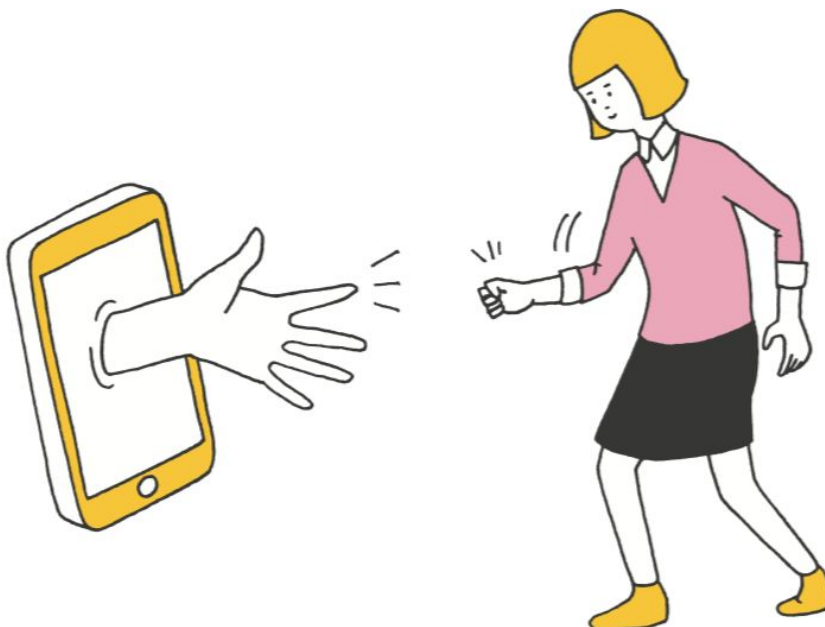
## ◎「パー」ボタン関連付け操作画面 2

画面のようなポップアップが表示されます。

- ① [Connection] を「Action」を選択します。
- ② [Name] を「paAction」と入力します。
- ③ [Connect] を選択します。



以上で、AutoLayoutの設定と、パーツとプログラムの関連付けは完了しました。  
お疲れ様でした！



## Lesson 1-4 対戦型に進化しよう

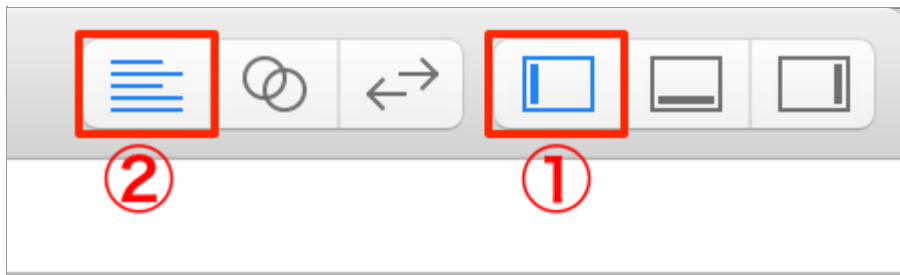
[このレッスンで学ぶこと]	[できるようになること]
タイマーを用いたアニメーションを学びます。 また、アニメーションの状態を管理するためにenumを利用します。	タイマー機能を応用してゲームを作れるようになります。 簡単な状態遷移について理解できるようになります。

### 1:アニメーションをしてみよう

ボタンを押したらアニメーションを開始してもう1度ボタンを押したらアニメーションを停止するようにしてみましょう。

#### (1-1) プロジェクトナビゲータを表示します

©Standard Editor切り替え操作画面

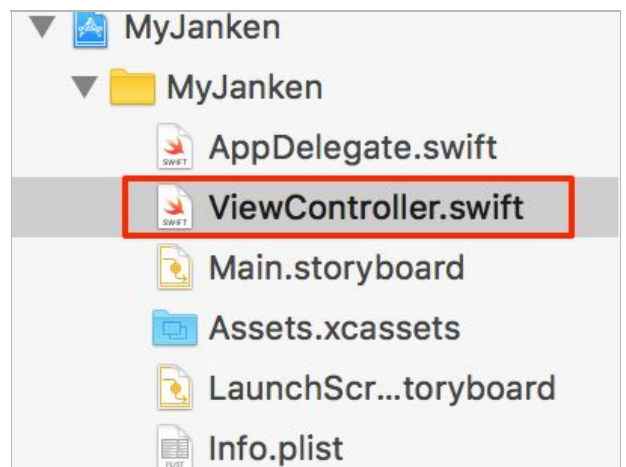


- ① [Navigator] ボタンをクリックして [Navigator] を開きます。
- ② [Standard Editor] ボタンをクリックして、エディタを表示します。

#### (1-2) ViewController.swiftを選択します

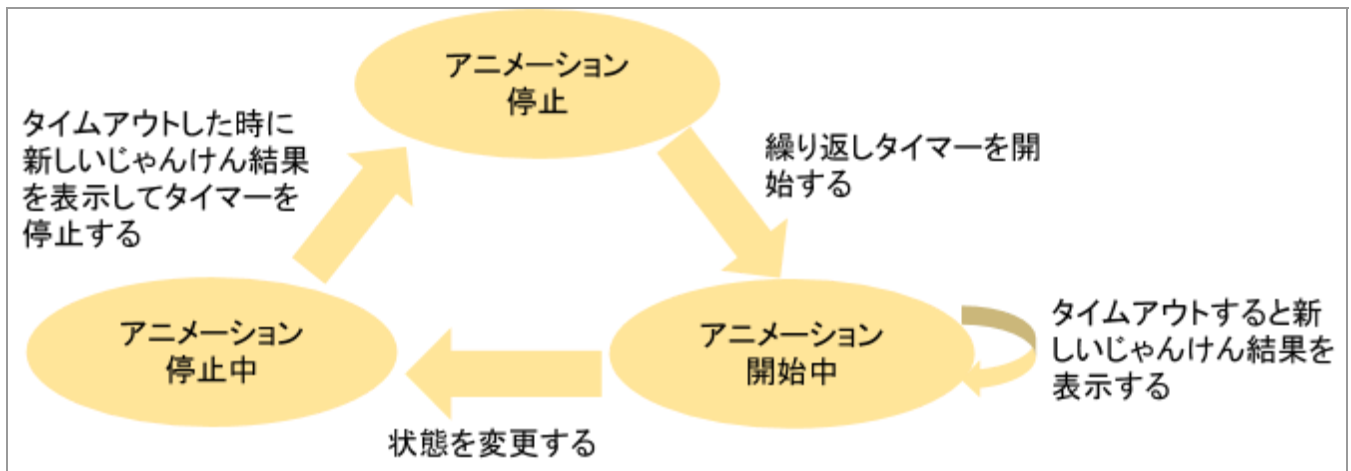
©ViewController.swift選択操作画面

[Project navigator] から   
[MessagesViewController.swift] を選択します



### (1-3) アニメーションの状態を管理するenumと変数を作成します

#### ◎enumで作成する状態と役割



アニメーションを管理するための状態を表す「enum」を作成します。

3つの状態「アニメーション停止」、「アニメーション開始中」「アニメーション停止中」を作成します。

アニメーションの停止をすぐに行わない理由は、プレイヤーが最後に表示したじゃんけん結果をもとに後出しじゃんけんをする可能性があるからです。

必ずランダムになるようにするためにタップ後に、もう1度じゃんけん結果を再表示する必要があります。

そのために「アニメーション停止中」の状態があります。

ここでも、もう一度「enum」の利用方法を復習していきましょう！

#### ◎enumを定義したソースコード

アニメーションの状態を管理するenum「jotai」を定義しています。

また、変数animationStateを、enum型jotaiを指定して作成します。

初期状態としては、アニメーション停止を表す「.stop」を指定しています。

```
101  @IBAction func paAction(_ sender: AnyObject) {
102  }
103
104  // アニメーションの状態を表すenum
105  enum jotai : Int {
106      // アニメーション中
107      case start
108      // アニメーション停止中
109      case stopping
110      // アニメーション停止
111      case stop
112  }
113
114  // アニメーションの状態を管理する変数
115  var animationState : jotai = .stop
116
117
118  }
```

追加

## (1-4) アニメーション行う処理を作成します

### ◎アニメーション開始と停止を受け付けるソースコード

```
114 // アニメーションの状態を管理する変数
115 var animationState : jotai = .stop
116
117 // タイマーの変数を作成
118 var timer : Timer!
119
120 // じゃんけんを次々変わるアニメーションを開始する
121 func startAnimation() {
122     // timerをアンラップしてnowTimerに代入
123     if let nowTimer = timer {
124         // もしタイマーが、実行中だったらスタートしない
125         if nowTimer.isValid == true {
126             // 何も処理しない
127             return
128         }
129     }
130
131     // タイマーをスタート
132     timer = Timer.scheduledTimer(timeInterval: 0.1,
133                                   target: self,
134                                   selector: #selector(self.timerInterrupt(_:)),
135                                   userInfo: nil,
136                                   repeats: true)
137
138     // 状態をアニメーション中に変更する
139     animationState = .start
140 }
141
142 // じゃんけんを次々変わるアニメーションを停止する
143 func stopAnimation() {
144     if animationState == .start {
145         // 状態を停止中に変更する
146         animationState = .stopping
147     }
148 }
149
150 }
```

追加

アニメーション開始と停止を受け付けるプログラムを記述しています。

アニメーション開始処理「startAnimation」メソッドは、まずタイマーが開始中かどうかをチェックしています。

アンラップに関しては、書籍の「1日目Lesson5 マップ検索アプリを作ろう ~UI パーツの扱いと delegate を学ぶ~」でも学びましたね。

タイマーが実行中の場合は処理をせずにプログラムを終了し、タイマーを実行していない時はタイマーを開始します。

そしてアニメーション状態の変数「animationState」を開始中「.start」に変更します。

ここでタイマースタートするコードがエラーとなっていますが、タイマー完了時のメソッド「timerInterrupt」が存在しないのでエラーとなっています。

後ほど作成しますのでここではエラーのまま先に進みましょう。

アニメーション停止処理「stopAnimation」メソッドは、アニメーション状態の変数「animationState」が開始中「.start」かをチェックしています。開始中の場合はアニメーション状態を停止中「.stopping」に変更しています。



## ◎タイマータイムアウト時のプログラム

```
142 // じゃんけんを次々変わるアニメーションを停止する
143 func stopAnimation() {
144     if animationState == .start {
145         // 状態を停止中に変更する
146         animationState = .stopping
147     }
148 }
149
150 func timerInterrupt(_ timer:Timer) {
151     // 新しいじゃんけんの結果を一時駅に格納する変数を設ける
152     var newAnswerNumber:UInt32 = 0
153
154     // ランダムに結果を出す但前回の結果と異なる場合のみ採用する
155     // repeat は繰り返しを意味する
156     repeat {
157         // じゃんけん結果をランダムに算出 (乱数)
158         newAnswerNumber = arc4random_uniform(3)
159
160         // 前回と同じ結果ときは、再度、ランダムにじゃんけん結果をだす
161         // 異なる結果のときは、repeat を抜ける
162     } while answerNumber.rawValue == newAnswerNumber
163
164     // 新しいじゃんけんの結果を格納
165     if (newAnswerNumber == janken.gu.rawValue) {
166         answerNumber = .gu
167     } else if (newAnswerNumber == janken.choki.rawValue) {
168         answerNumber = .choki
169     } else if (newAnswerNumber == janken.pa.rawValue) {
170         answerNumber = .pa
171     }
172
173     // じゃんけんから文字列を取り出す
174     answerLabel.text = answerNumber.string()
175     // じゃんけんから画像を取り出す
176     answerImageView.image = UIImage(named: answerNumber.imageName())
177
178     if animationState == .stopping {
179         // タイマー停止
180         timer.invalidate()
181         // 状態を停止に変更する
182         animationState = .stop
183     }
184 }
185 }
```

追加

タイマー完了時のメソッドのプログラムを記述しています。

まず新しいじゃんけん結果を取得しています。

ここは、「じゃんけんをする！」ボタンのメソッド「shuffleAction」と同じ内容です。

最後に、アニメーション状態が停止中「.stopping」の時は、タイマーを停止してアニメーション状態を停止「.stop」に変更しています。

## (1-5) ボタンをタップしたらアニメーション開始と停止するようにしよう

### ◎ボタンをタップした時にアニメーション開始と停止するプログラム

各ボタンとも同じ処理です。  
アニメーション状態を確認して停止していたら開始するメソッド「startAnimation」を実行します。アニメーションが開始していたらアニメーション停止するメソッド「stopAnimation」を実行しています。

アニメーションが開始と停止が行われるようになります。

```
○ 96  @IBAction func guAction( sender: AnyObject) {
97      // アニメーション状態をチェック
98      if animationState == .stop {
99          // アニメーション停止しているので開始する
100         startAnimation()
101     } else if animationState == .start {
102         // アニメーション開始しているので停止する
103         stopAnimation()
104     }
105 }
106
○107 @IBAction func chokiAction( sender: AnyObject) {
108     // アニメーション状態をチェック
109     if animationState == .stop {
110         // アニメーション停止しているので開始する
111         startAnimation()
112     } else if animationState == .start {
113         // アニメーション開始しているので停止する
114         stopAnimation()
115     }
116 }
117
○118 @IBAction func paAction( sender: AnyObject) {
119     // アニメーション状態をチェック
120     if animationState == .stop {
121         // アニメーション停止しているので開始する
122         startAnimation()
123     } else if animationState == .start {
124         // アニメーション開始しているので停止する
125         stopAnimation()
126     }
127 }
```

追加

追加

追加

ここで実行してみましょう。ボタンをタップするとアニメーションが開始し、もう1度タップするとアニメーションが停止します。

## 2:アニメーション停止する時に対戦結果を表示してみよう

### (2-1) アニメーションする時にタップしたじゃんけんを覚えます

対戦結果を表示するためにはどのじゃんけんがタップされたか覚える必要があります。  
アニメーションを停止する時に表示中にじゃんけん結果とタップしたじゃんけんを比べて勝敗を表示します。

### ◎タップした時にじゃんけんを覚える変数を作成

タップした時にじゃんけんを覚える変数を作成します。こちらでもenum「janken」を使います。

```
142 // タイマーの変数を作成
143 var timer : Timer!
144
145 // タップしたじゃんけん
146 var tappedJanken:janken = .gu
147
148 // じゃんけんを次々変わるアニメーションを開始する
149 func startAnimation() {
150     // timerをアンラップしてnowTimerに代入
```

追加

## ◎タップした時にじゃんけんを覚えるプログラム

タップしアニメーション停止する前にタップされたじゃんけんを覚えていきます。

「グー」「チョキ」「パー」とそれぞれタップされたら、「tappedJanken」にenumを代入して保持しておきます。

後ほど、保持された「tappedJanken」の値を利用して勝負の判定を行います。

```
○ 96 @IBAction func guAction(_ sender: AnyObject) {
97     // アニメーション状態をチェック
98     if animationState == .stop {
99         // アニメーション停止しているので開始する
100         startAnimation()
101     } else if animationState == .start {
102         // タップしたときのじゃんけんを覚える
103         tappedJanken = .gu
104     }
105     // アニメーション開始しているので停止する
106     stopAnimation()
107 }
108 }
○ 110 @IBAction func chokiAction(_ sender: AnyObject) {
111     // アニメーション状態をチェック
112     if animationState == .stop {
113         // アニメーション停止しているので開始する
114         startAnimation()
115     } else if animationState == .start {
116         // タップしたときのじゃんけんを覚える
117         tappedJanken = .choki
118     }
119     // アニメーション開始しているので停止する
120     stopAnimation()
121 }
122 }
○ 124 @IBAction func paAction(_ sender: AnyObject) {
125     // アニメーション状態をチェック
126     if animationState == .stop {
127         // アニメーション停止しているので開始する
128         startAnimation()
129     } else if animationState == .start {
130         // タップしたときのじゃんけんを覚える
131         tappedJanken = .pa
132     }
133     // アニメーション開始しているので停止する
134     stopAnimation()
135 }
136 }
```

追加

追加

追加



## (2-2) アニメーションを停止する時に対戦結果を表示します

### ◎対戦結果を表示するプログラム

「アニメーション停止中」時にタイマー停止する処理の後に、対戦結果を判定しています。  
対戦結果の初期値は、「負け」にしています。

表示中のじゃんけん結果とタップしたじゃんけんが同じなら「あいこ」にしています。

次に対戦結果が「勝ち」条件を判定しています。

全ての条件分岐が完了したら対戦結果を表示しています。

```
187 func timerInterrupt(_ timer:Timer) {
188     // 新しいじゃんけんの結果を一時駅に格納する変数を設ける
189     var newAnswerNumber:UInt32 = 0
190
191     // ランダムに結果を出す。前回の結果と異なる場合のみ採用する
192     // repeat は繰り返しを意味する
193     repeat {
194         // じゃんけん結果をランダムに算出 (乱数)
195         newAnswerNumber = arc4random_uniform(3)
196
197         // 前回と同じ結果ときは、再度、ランダムにじゃんけん結果をだす
198         // 異なる結果のときは、repeat を抜ける
199     } while answerNumber.rawValue == newAnswerNumber
200
201     // 新しいじゃんけんの結果を格納
202     if (newAnswerNumber == janken.gu.rawValue) {
203         answerNumber = .gu
204     } else if (newAnswerNumber == janken.choki.rawValue) {
205         answerNumber = .choki
206     } else if (newAnswerNumber == janken.pa.rawValue) {
207         answerNumber = .pa
208     }
209
210     // じゃんけんから文字列を取り出す
211     answerLabel.text = answerNumber.string()
212     // じゃんけんから画像を取り出す
213     answerImageView.image = UIImage(named: answerNumber.imageName())
214
215     if animationState == .stopping {
216         // タイマー停止
217         timer.invalidate()
218         // 状態を停止に変更する
219         animationState = .stop
220
221         // 対戦結果の初期値は"負け"
222         var kekkaText : String = "負け"
223
224         if answerNumber == tappedJanken {
225             // 同じ内容なら"あいこ"
226             kekkaText = "あいこ"
227         } else if (answerNumber == .gu) {
228             if (tappedJanken == .pa) {
229                 kekkaText = "勝ち"
230             }
231         } else if (answerNumber == .choki) {
232             if (tappedJanken == .gu) {
233                 kekkaText = "勝ち"
234             }
235         } else if (answerNumber == .pa) {
236             if (tappedJanken == .choki) {
237                 kekkaText = "勝ち"
238             }
239         }
240         // 対戦結果を表示する
241         answerLabel.text = "\(answerNumber.string()) \(kekkaText)"
242     }
243 }
```

追加

ここで実行してみましょう。タップしてアニメーションが開始します。もう1度タップしたじゃんけんとアニメーション停止した時のじゃんけん結果で対戦結果が表示されます。

以上で、対戦型じゃんけんの開発は終了です。