

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。

公式サイトの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

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

本書の「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しょう。また、本書のじゃんけんアプリでは、「カスタマイズ編①～起動画面（LaunchScreen）を設定しよう～」、「カスタマイズ編②～アイコンを設定しよう～」がありますが、カスタマイズ編の学習を終えていなくても、応用編を実施することは可能です。

プログラムとユーザーでじゃんけんをして、結果が表示されるアプリを制作していきます。

これまでのじゃんけんアプリにSwiftの列挙型であるenum（イーナム）を使って、プログラムコードをわかりやすく整理していきます。enumの基本的な使い方を学習していきましょう。

また、新しくボタンを3つ追加し、AutoLayoutを用いてUIボタンが均等の幅になるように設定していきます。

- Lesson 1-1 完成イメージを確認しよう
- Lesson 1-2 enum（列挙型）で整理しよう
- Lesson 1-3 画面を作成しよう
- Lesson 1-4 対戦型へカスタマイズしよう



Lesson 1-1 完成の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]
対戦型じゃんけんアプリの完成イメージを確認します。完成をイメージすることでこれから行なう作業の見通しを立てます。 対戦型に変更するときのポイントも確認しましょう。	対戦型じゃんけんの使い方が理解できるようになります。完成イメージからどのような制作を行なうのか、イメージができるようになります。

1:対戦型じゃんけんアプリとは

この章では、「1日目Lesson 3 じゃんけんアプリを作ろう」のサンプルアプリを更にカスタマイズして、対戦型じゃんけんアプリを作っていきます。

まずは、完成イメージを確認してみましょう。

◎対戦型じゃんけんアプリの完成イメージ



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ランダムに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、目で見て狙い打ちすることができてしまい、必ず勝ててしまいます。じゃんけん結果を表示するとき、停止する前にもう1度表示更新してから対戦結果を表示します。（押したあとに、もう一度、計算して出しています。）

Lesson 1-2 enum (列挙型) で整理しよう

[このレッスンで学ぶこと]

後のカスタマイズを行いやすくするために、enum (列挙型) を使って、プログラムコードを見やすく整理していきます。
Swiftのenum (列挙型) の使用方法を学んでいきます。

[できるようになること]

enumの基本の使い方がわかるようになり、わかりやすいプログラムコードを作ることができるようになります。

1:enum (イナム) とは

(1-1) enumを使う理由

じゃんけんアプリをカスタマイズする前に、enumについて理解しておきましょう。

enumは、一連の値を効率よく変数定義できることから「列挙型」と呼ばれます。

本書の「1日目Lesson 3 じゃんけんアプリを作ろう」(P96)では、answerNumberが「0」なら「グー」、「1」なら「チョキ」、「2」なら「パー」と数値で表現していました。

◎answerNumberの値とその意味

じゃんけん結果	グー	チョキ	パー
answerNumberの値	0	1	2

ですが、プログラムコードだけを見ると、「0はグーで、1はチョキ・・・」などと、どの数値がどのようなじゃんけん結果を示しているのかがわかりづらいです。このようなプログラムコードは読み手に負担がかかり、効率が悪くなります。

本書のプログラム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。

でも、実際は、このプログラムコードより、はるかに多くのプログラムコードを書いてiOSアプリを開発していきます。どんな記憶力が良いアプリ開発者でも、時間が経てば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。

Tips

マジックナンバー

プログラミングの世界では、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。

(1-2) enumの使い方

◎enumの記述例1:定義

```
// Jyankenという名前のenumを定義
enum Jyanken {

    // 値 : ゲー
    case j_gu

    // 値 : チョキ
    case j_choki

    // 値 : パー
    case j_pa

}

// 変数answerはJyanken列挙型で定義し、初期値は.guとする
var answer : Jyanken = .j_gu
```

enumを利用する際には、「enum Jyanken」のように変数の型（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2：値の設定と取得

```
// Jyankenという名前のenum定義
// 値はUInt32の値で保持する
enum Jyanken : UInt32{
    // 値 : ゲー(値は0)
    case j_gu = 0
    // 値 : チョキ(値は1)
    case j_choki = 1
    // 値 : パー(値は2)
    case j_pa = 2
}

// 変数answerはJyanken列挙型で定義し、初期値は.guとする
var answer : Jyanken = .j_gu

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

またEnumの各値の詳細を定義することも可能です。

Enumで値を格納する場合は、「Jyanken:UInt32」のように型を指定します。

上記の例の場合は、「.j_gu」なら「0」、「.j_choki」なら「1」という値を設定することも可能です。値はString型など文字列型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3：メソッドの追加

```
enum Jyanken : UInt32{
    // 値：グー(値は0)
    case j_gu = 0
    // 値：チョキ(値は1)
    case j_choki = 1
    // 値：パー(値は2)
    case j_pa = 2

    // 値を文字に変換するメソッド
    func string() -> String {
        switch self {
            case .j_gu:
                return "グー"
            case .j_choki:
                return "チョキ"
            case .j_pa:
                return "パー"
        }
        return "不明"
    }
}

// 変数answerはJyanken列挙型で定義し、初期値はグーとする
var answer : Jyanken = .j_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
// グーが取得できる
print(answer.string())
```

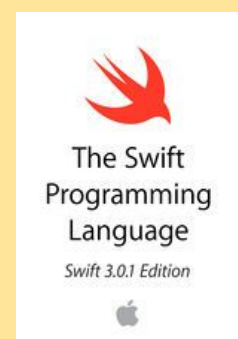
enum内部には、最後にメソッドを記述することができます。
メソッドを記述することによって、値の表現方法を変化させることが可能です。
上記の例では、じゃんけんの定義をユーザーがわかる文字列に変換しています。

SwiftのEnum（列挙型）はとても機能が豊富です。これ以上の解説は、初心者の域を超えてしまうため、本書では割愛していますが、もし、学習してみたいという方は「The Swift Programming Language」（書籍）を片手にチャレンジしてみてください！

Tips

「The Swift Programming Language」（書籍）のご紹介

Swift言語を更に学習したい方のために、
「The Swift Programming Language」をご紹介します。
この書籍は、Appleが出版しているSwift言語のマニュアル（英語）です。
iBookから無料で購入することができます。
英語で記載もされていますので、難易度が高いです。ですが、Swift言語としては、もっとも新しい書籍になり、慣れるとわかりやすいと思います。



2:enumを使ってコードを整理してみよう

本書の「1日目 Lesson 3 じゃんけんアプリを作ろう」で制作した「じゃんけんアプリ」を、さらにカスタマイズしていきます。

(2-1) enumを定義します

◎enumを定義

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

まずは、enumでJyanken列挙型を定義します。数値を設定するため、Jyanken列挙型はUInt32数値型を指定しています。

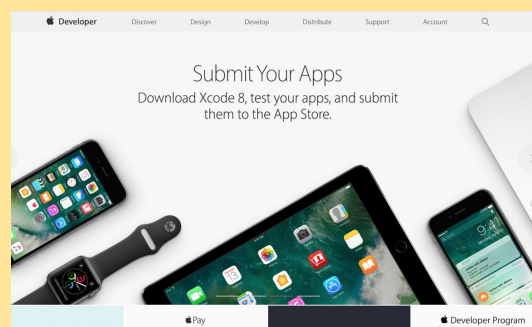
「j_gu」は「0」、「j_choki」は「1」、「j_pa」は「2」を代入します。

Tips

Apple開発者向けWebサイト「Apple Developer」のご紹介

Appleが開発者向けに情報をまとめているサイトが、「Apple Developer」 (<https://developer.apple.com/>) です。

WWDCでのセッション動画が掲載されていたり、開発版である「Xcode beta」のダウンロードが行えます。それ以外にもたくさんの開発情報が掲載されていますので、ぜひ、サイトに訪れてみてください。サイトを閲覧しているブラウザがSafariでないと動画の再生が行えないなどの制約がありますので、Safariを使ってサイトを閲覧してください。



(2-2) enumを利用します

◎enumを利用する

```
37 // じゃんけん (Enum)
38 var answerNumber: Jyanken = .j_gu ①
39
40 @IBAction func shuffleAction(_ sender: Any) {
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 == Jyanken.j_gu.rawValue {
59         answerNumber = .j_gu
60     } else if newAnswerNumber == Jyanken.j_choki.rawValue {
61         answerNumber = .j_choki
62     } else if newAnswerNumber == Jyanken.j_pa.rawValue {
63         answerNumber = .j_pa
64     }
65
66     if answerNumber == .j_gu { ④
67         // グー
68         answerLabel.text = "グー"
69         answerImageView.image = UIImage(named: "gu")
70
71     } else if answerNumber == .j_choki {
72         // チョキ
73         answerLabel.text = "チョキ"
74         answerImageView.image = UIImage(named: "choki")
75
76     } else if answerNumber == .j_pa {
77         // パー
78         answerLabel.text = "パー"
79         answerImageView.image = UIImage(named: "pa")
80
81     }
82 }
```

- 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Jyanken型に置き換えます。初期値で「グー」を指定しています。
- 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、「.j_gu」「.j_choki」「.j_pa」とenumに置き換えます。

じゃんけんの結果を「0」や「1」などの数値で記載するよりも、「.j_gu」「.j_choki」のように記載することで、遥かに読みやすく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整理します

◎テキストの表示と画像の表示

コードを確認すると、「answerLabel.text = "グー"」のように画面に表示する箇所と、「answerImageView.image = UIImage(named: "gu")」のように画像に変換している箇所があります。

このコードは、enumを活用することで、もっとスッキリと記述できます。また、後の変更に強く、不具合が発生したときもメンテナンスが行いやすくなります。

```
65
66     if answerNumber == .j_gu {
67         // グー
68         answerLabel.text = "グー"
69         answerImageView.image = UIImage(named: "gu")
70     } else if answerNumber == .j_choki {
71         // チョキ
72         answerLabel.text = "チョキ"
73         answerImageView.image = UIImage(named: "choki")
74     } else if answerNumber == .j_pa {
75         // パー
76         answerLabel.text = "パー"
77         answerImageView.image = UIImage(named: "pa")
78     }
79
80
81 }
```

◎enumにメソッドを定義

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。

enumの値からじゃんけん結果の文字列に変換するメソッドでは、変換後のじゃんけんの文字列を配列に格納しています。「self.rawValue」で、enumの値を取得して、配列の添字として指定することで、じゃんけんの文字列を返却しています。

同様に、enumの値から画像の名前を取得するメソッドも作成します。

```
27 // じゃんけんのenum
28 enum Jyanken : UInt32 {
29     // グーの定義
30     case j_gu = 0
31     // チョキの定義
32     case j_choki = 1
33     // パーの定義
34     case j_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から画像名に変換
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メソッドを利用

```
61 // じゃんけん (Enum)
62 var answerNumber:Jyanken = .j_gu
63
64 @IBAction func shuffleAction(_ sender: Any) {
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 == Jyanken.j_gu.rawValue {
83         answerNumber = .j_gu
84     } else if newAnswerNumber == Jyanken.j_choki.rawValue {
85         answerNumber = .j_choki
86     } else if newAnswerNumber == Jyanken.j_pa.rawValue {
87         answerNumber = .j_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、Jyanken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。

以前のif文で分岐するコードと比べると、enumのメソッドを活用することでスッキリしたプログラムになったことがわかります。




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]
対戦型じゃんけんアプリへとカスタマイズするために画面のボタンの削除方法と、複数のボタンを均等に配置していく方法を学びます。 新しいAutoLayoutの設定方法を学びます。	複数のUIパーツの配色の設定ができるようになります。複数のUIパーツを均等なサイズで配置するためのAutoLayoutの活用もできるようになります。

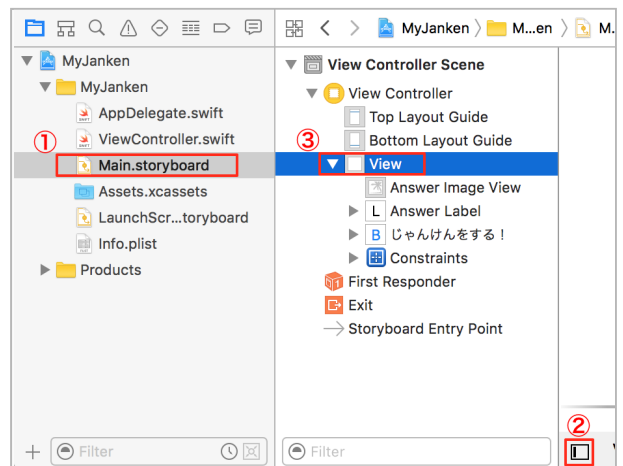
1: 「じゃんけんをする！」ボタンを削除しよう

対戦型じゃんけんアプリの画面を作っていきます。既に配置されている「じゃんけんをする！」ボタンを削除して、「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。

(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



(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。

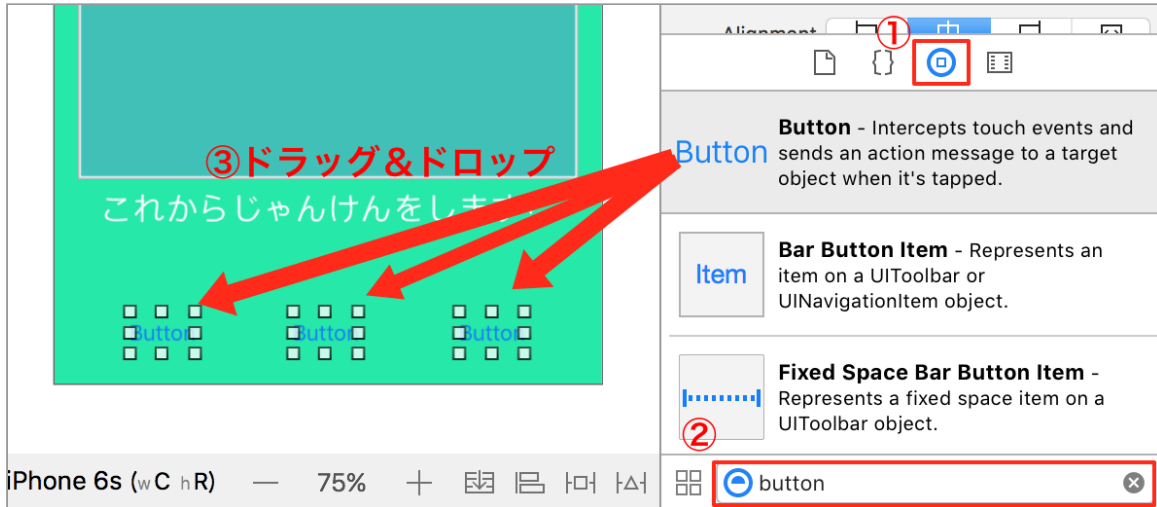
◎「じゃんけんをする！」を削除



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



- 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 表示されたボタンを、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、ボタンを横一列に3個配置します。

(2-2) タイトルを設定します

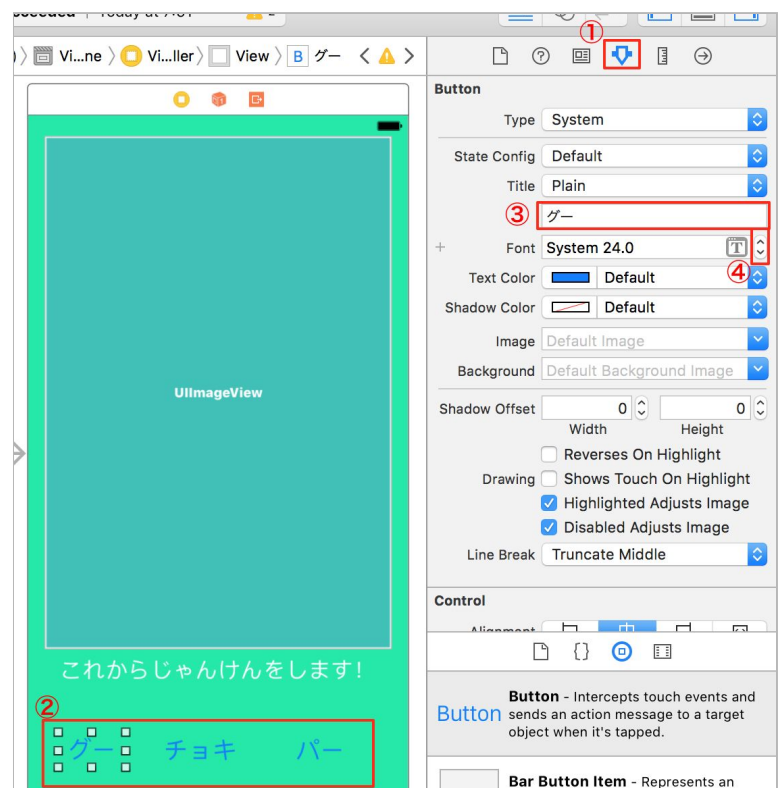
ボタンのタイトルをデフォルトの「Button」から「グー」、「チョキ」、「パー」に変更します。

- ① [Attributes inspector] を選択してください。
- ② 左端のボタンを選択します。
- ③ [Title] に「グー」と入力します。
- ④ ボタンのフォントサイズを24に変更します。

同様に、真ん中のボタンの [Title] は「チョキ」、右端のボタンの [Title] は「パー」と入力します。それぞれ、フォントサイズは24に変更します。

フォントサイズを大きくしたことで「グー」「チョキ」「パー」の文字が「...」と表示された場合は、ボタン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。

◎Title設定

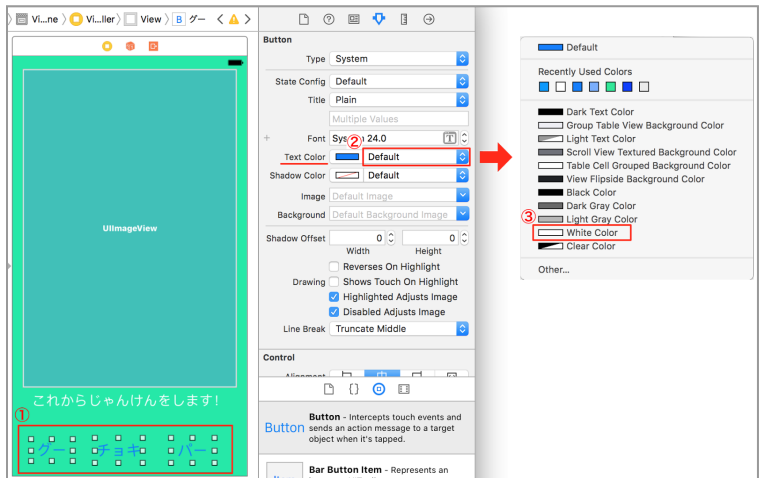


(2-3) 配色を変更します

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、「グー」、「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。

◎テキストの配色変更

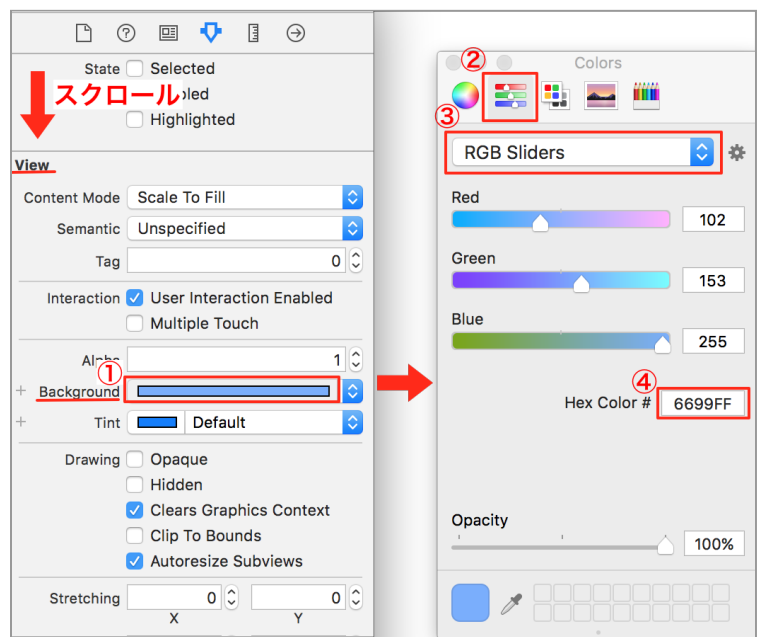


次に、ボタンの背景色を変更します。

- 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 #] に「6699FF」と入力します。

3つのボタンのテキストがホワイトになり、背景がブルーになっていることを確認します。

◎ボタンの配色変更



3:AutoLayoutで、レイアウトを整えよう

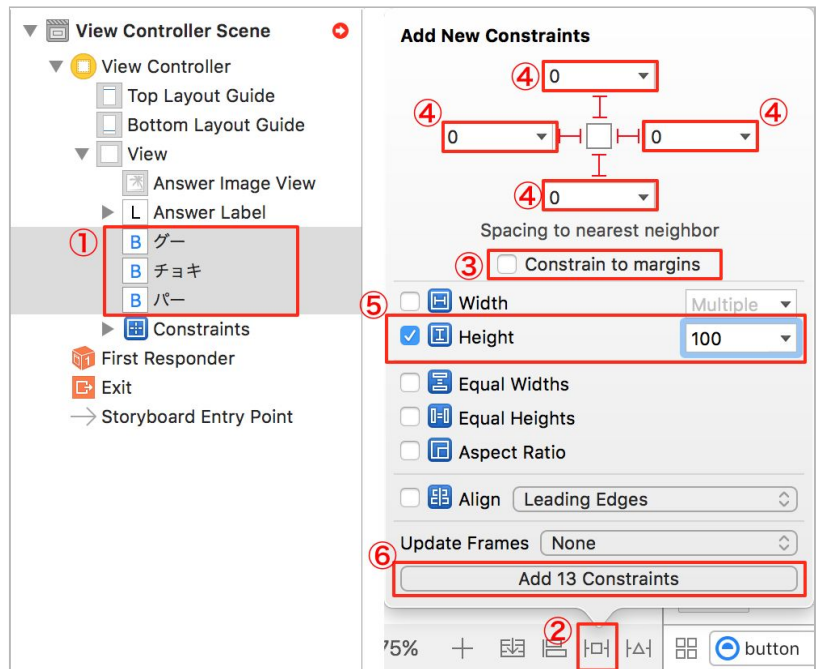
AutoLayoutとは、画面の大きさに合わせて、配置したUIパーツを自動的にレイアウトする機能です。

(3-1) ボタンの余白と高さを設定します

「グー」、「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「グー」、「チョキ」、「パー」を「command ⌘」キーを押しながら、同時に選択します。
- ② [Add New Constraints] をクリックします。
[Add New Constraints] 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] という画面が表示されます。
- 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。
入力後は「Tab」キーで移動します。
- ⑤ [Height] をチェック入れて、「100」と入力します。
- ⑥ [Add 13 Constraints] をクリックして、制約を追加します。

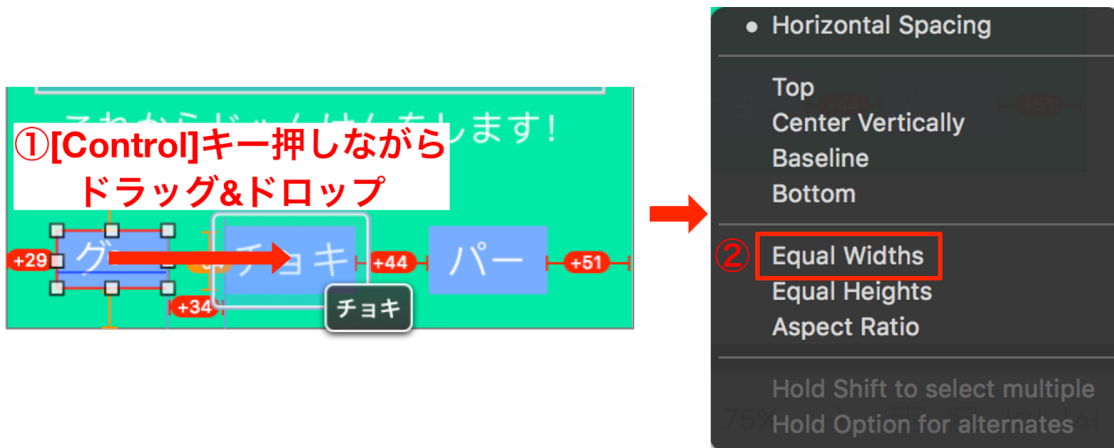
◎ 3つのボタンのAutoLayout設定



もし、この章でのAutoLayoutの設定が難しいと感じた方は、書籍の「1日目Lesson 3 じゃんけんアプリを作ろう」の「3 各パーツの表示位置、幅や高さを設定しよう」(P69)を参考にしてみてください。

(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つのボタン幅が均等になるように AutoLayoutを設定



- 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。操作が完了すると黒いポップアップ画面が表示されます。
- ② ポップアップした画面の [Equal Widths] をクリックします。これが、幅を均等にする設定になります。
- ③ 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) AutoLayoutの警告を解消します（※この操作は、最新のXcodeでは不要です。）

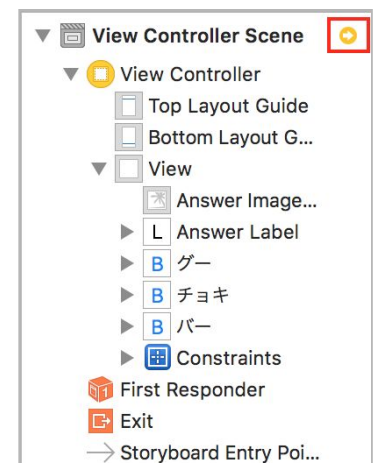
◎ 警告の表示

[Document Outline] の [View Controller Scene] に  が表示されます。

これは、パーツにAutoLayoutの制約として設定した位置と、実際の位置が一致していないため警告マークが表示されています。


これを、「ビューの誤配置」と言います。

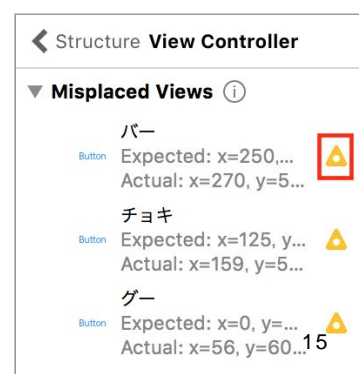
 マークをクリックして、「ビューの誤配置」を解消します。



◎ 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。



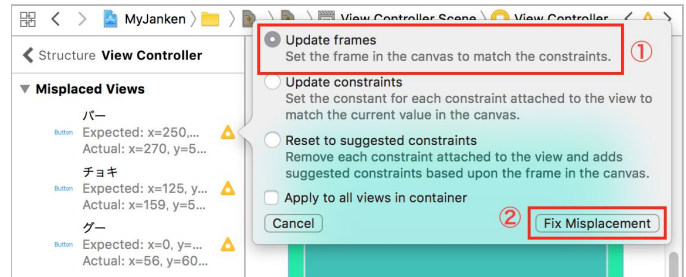
◎配置のズレを解消

ラジオボタン「Update frames」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わせます」という意味です。

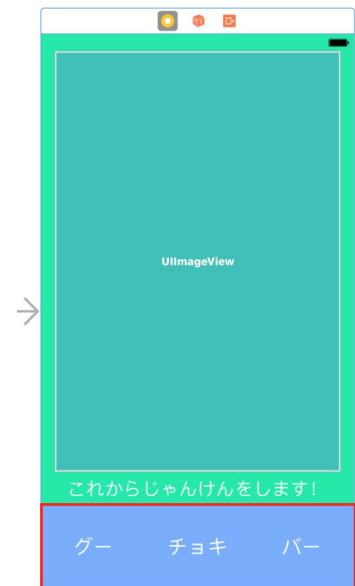
「Fix Misplacement」をクリックして、位置のズレを解消します。

ひとつめの警告が解消できたら、警告がなくなるまで、同じ手順で解消していきます。



◎位置のズレ解消後

警告が解消できたら、「ゲー」「チョコキ」「パー」のボタンが高さ「100」ポイント、横幅いっぱい設定されるようになります。

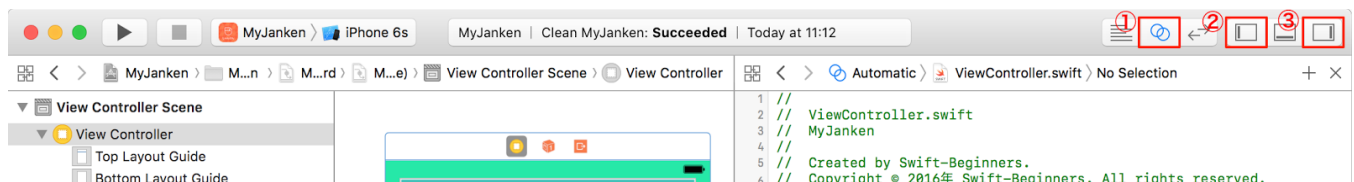


4:関連付けをしよう

新しく追加したボタンとプログラムとの関連付けをしていきます。

(4-1) Assistant Editorを表示します

◎Assistant Editor切り替え

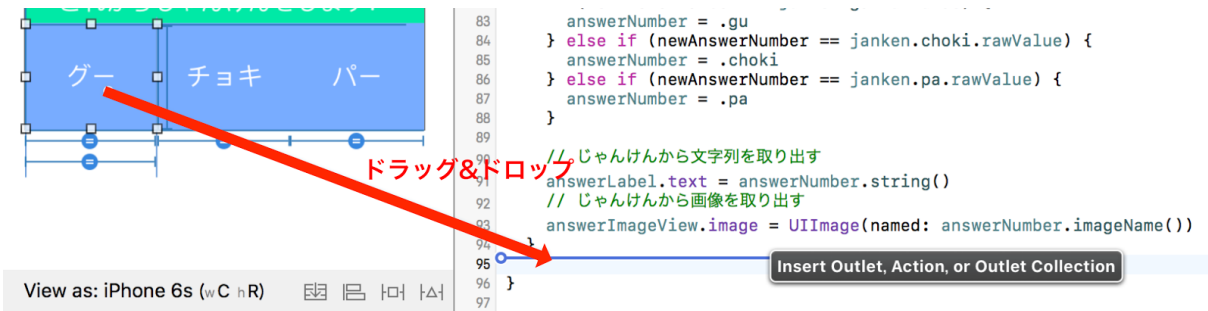


- ① [Assistant Editor] ボタンをクリックして、エディタを表示します。
- ② [Navigator] ボタンをクリックして [Navigator] を閉じます。
- ③ [Utilities] ボタンをクリックして [Utilities] を閉じます。

ここから、関連付けを行います。

(4-2) 3つのボタンの関連付け

◎ 「ゲー」 ボタン関連付け



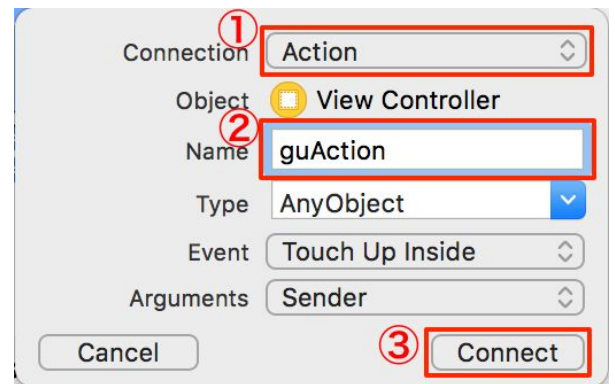
「ゲー」 ボタンを選択して、「control」キーを押しながら、右側のコードにドラッグ&ドロップします。

一番下の「}」の上でドラッグを離して配置してください。

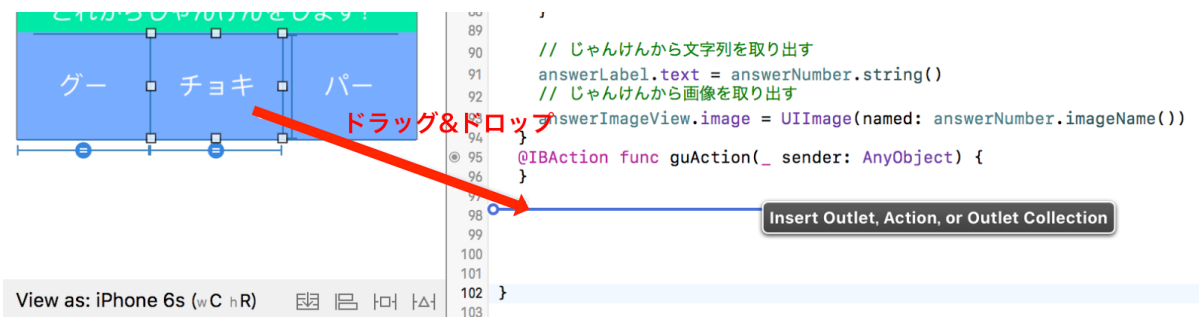
◎ 「ゲー」 ボタン関連付け

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

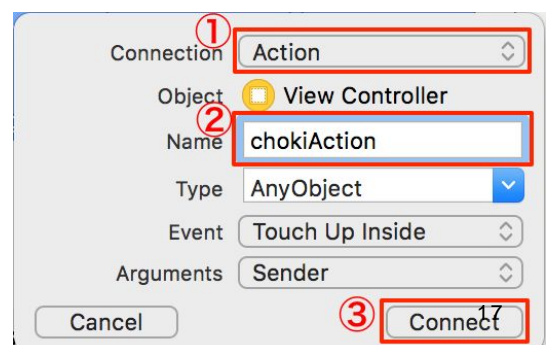


同じく「チョキ」 ボタンを選択して、「control」キーを押しながら、右側のコードにドラッグ&ドロップします。

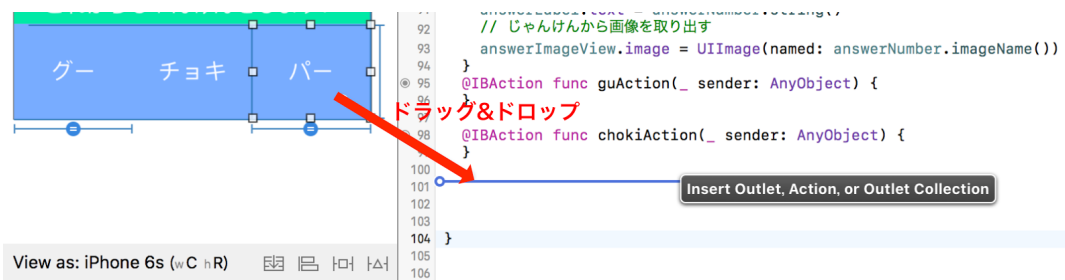
◎ 「チョキ」 ボタン関連付け

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

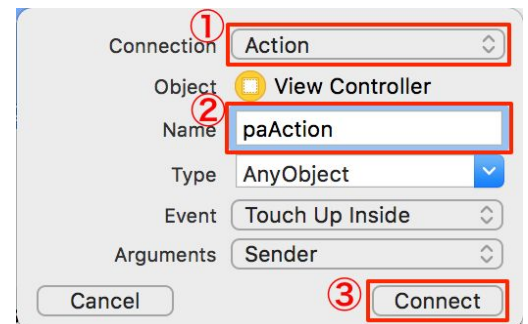


「パー」 ボタンを選択して、「control」 キー を押しながら、右側のコードにドラッグ&ドロップします。

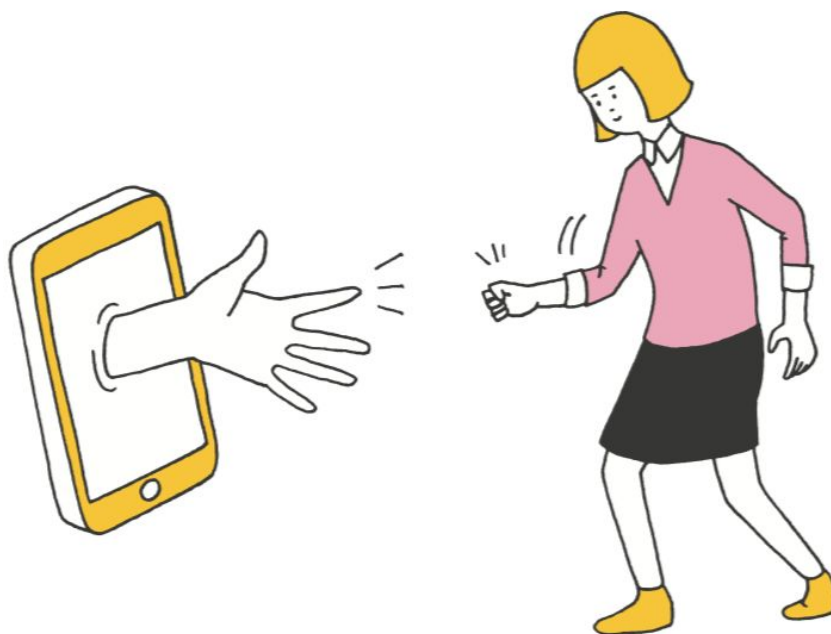
◎ 「パー」 ボタン関連付け

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 対戦型へカスタマイズしよう

[このレッスンで学ぶこと]	[できるようになること]
<p>ランダムにじゃんけん結果を表示していく方法や、タイマーを用いたアニメーションを学びます。</p> <p>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。</p>	<p>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。</p> <p>簡単な状態遷移について理解できるようになります。</p>

1:アニメーションを試みよう

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

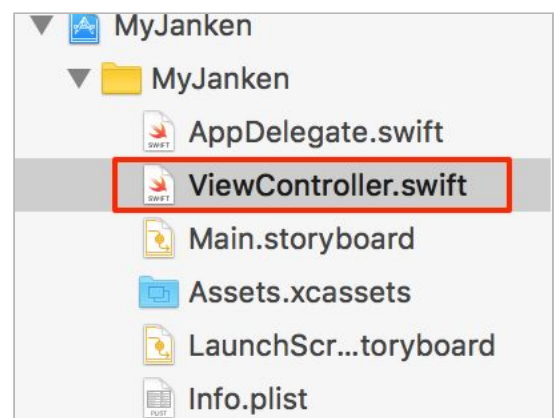


- ① [Navigator] ボタンをクリックして [Navigator] を開きます。
- ② [Standard Editor] ボタンをクリックして、エディタを表示します。

(1-2) ViewController.swiftを選択します

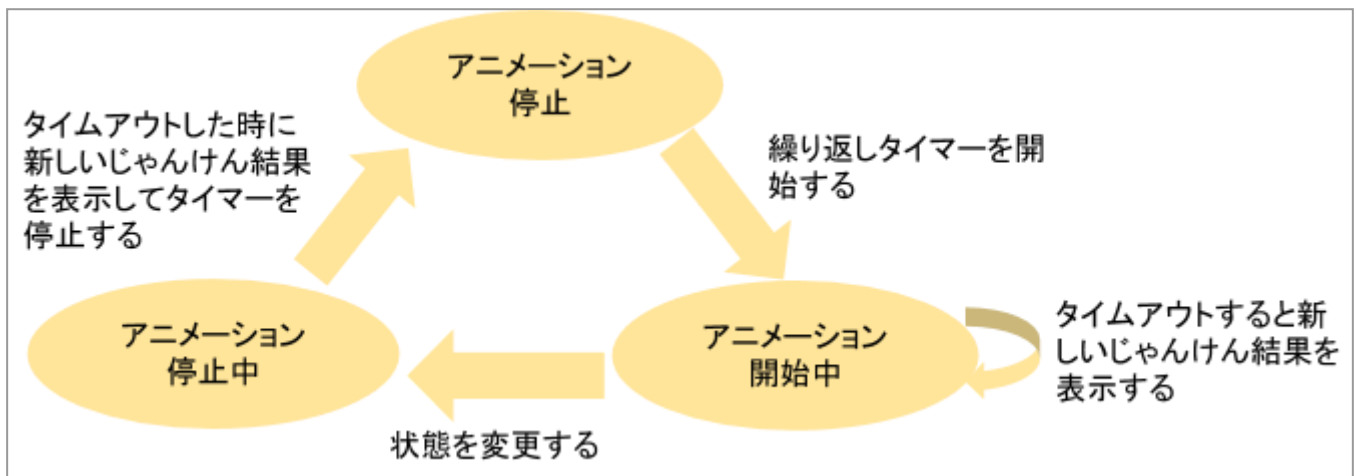
[Project navigator] から [ViewController.swift] を選択します

◎ViewController.swift選択



(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の定義

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。Int型を指定することで、メンバの「start」「stopping」「stop」に「0」「1」「2」と順番に値がセットされます。

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。

```
102  @IBAction func paAction(_ sender: Any) {
103  }
104
105  // アニメーションの状態を表すenum
106  enum Jotai : Int {
107  // アニメーション中
108  case start
109  // アニメーション停止中
110  case stopping
111  // アニメーション停止
112  case stop
113  }
114
115  // アニメーションの状態を管理する変数
116  var animationState : Jotai = .stop
117
118  }
119
```

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

```
115 // アニメーションの状態を管理する変数
116 var animationState : Jotai = .stop
117
118 // タイマーの変数を作成
119 var jTimer : Timer?
120
121 // じゃんけんを次々変わるアニメーションを開始する
122 func startAnimation() {
123     // timerをアンラップしてnowTimerに代入
124     if let nowTimer = jTimer {
125         // もしタイマーが、実行中だったらスタートしない
126         if nowTimer.isValid == true {
127             // 何も処理しない
128             return
129         }
130     }
131
132     // タイマーをスタート
133     jTimer = Timer.scheduledTimer(timeInterval: 0.1,
134                                     target: self,
135                                     selector: #selector(self.timerInterrupt(_:)),
136                                     userInfo: nil,
137                                     repeats: true)
138
139     // 状態をアニメーション中に変更する
140     animationState = .start
141 }
142
143 // じゃんけんを次々変わるアニメーションを停止する
144 func stopAnimation() {
145     if animationState == .start {
146         // 状態を停止中に変更する
147         animationState = .stopping
148     }
149 }
150
151
```

追加

```
// タイマーの変数を作成
var jTimer : Timer?
```

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

```
143 // じゃんけんを次々変わるアニメーションを停止する
144 func stopAnimation() {
145     if animationState == .start {
146         // 状態を停止中に変更する
147         animationState = .stopping
148     }
149 }
150
151 func timerInterrupt(_ timer:Timer) {
152     // 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
153     var newAnswerNumber:UInt32 = 0
154
155     // 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
156     // repeat は繰り返しを意味する
157     repeat {
158         // じゃんけん結果をランダムに算出 (乱数)
159         newAnswerNumber = arc4random_uniform(3)
160
161         // 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
162         // 異なる結果のときは、repeat を抜ける
163     } while answerNumber.rawValue == newAnswerNumber
164
165     // 新しいじゃんけんの結果を格納
166     if (newAnswerNumber == Jyanken.j_gu.rawValue) {
167         answerNumber = .j_gu
168     } else if (newAnswerNumber == Jyanken.j_choki.rawValue) {
169         answerNumber = .j_choki
170     } else if (newAnswerNumber == Jyanken.j_pa.rawValue) {
171         answerNumber = .j_pa
172     }
173
174     // じゃんけんから文字列を取り出す
175     answerLabel.text = answerNumber.string()
176     // じゃんけんから画像を取り出す
177     answerImageView.image = UIImage(named: answerNumber.imageName())
178
179     if animationState == .stopping {
180         // タイマー停止
181         timer.invalidate()
182         // 状態を停止に変更する
183         animationState = .stop
184     }
185 }
186 }
```

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「Jyanken」を使います。

```
142 // タイマーの変数を作成
143 var jTimer : Timer?
144
145 // タップしたじゃんけん
146 var tappedJanken:Jyanken = .j_gu 追加
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
```

◎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ます。

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

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

```
96 @IBAction func guAction(_ sender: Any) {
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 = .j_gu 追加
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 }
109
110 @IBAction func chokiAction(_ sender: Any) {
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 = .j_choki 追加
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 }
123
124 @IBAction func paAction(_ sender: Any) {
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 = .j_pa 追加
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 }
```


(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 == jyanken.j_gu.rawValue) {
203         answerNumber = .j_gu
204     } else if (newAnswerNumber == jyanken.j_choki.rawValue) {
205         answerNumber = .j_choki
206     } else if (newAnswerNumber == jyanken.j_pa.rawValue) {
207         answerNumber = .j_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 == .j_gu) {
228             if (tappedJanken == .j_pa) {
229                 kekkaText = "勝ち"
230             }
231         } else if (answerNumber == .j_choki) {
232             if (tappedJanken == .j_gu) {
233                 kekkaText = "勝ち"
234             }
235         } else if (answerNumber == .j_pa) {
236             if (tappedJanken == .j_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。