

「オブジェクト」には、英語で「もの」という意味があります。ものにはそれぞれ、「特有の状態」と「ふるまい」があります。たとえば人間なら、「名前」「年齢」「身長」といったものが特有の状態で、「歩く」「ジャンプする」「ものを拾う」などがふるまいになります。オブジェクトが持つ特有の状態のことを「プロパティ」、ふるまいを「メソッド」と呼んでいます。

「クラス」はオブジェクトがどのようなプロパティやメソッドを持っているのかを記述したものをです。iPhone を例にとると、「iPhone クラス」とは iPhone の設計図に相当します。この設計図には、通話、ブラウザ閲覧、音楽再生などの「メソッド」（ふるまい）と、バッテリー、音量などの「プロパティ」（状態）について記述されています。この設計図を元に、製品となる iPhone を生産することができます。この生産された iPhone が「iPhone オブジェクト」となります。1つの設計図（クラス）から、何個もの iPhone（オブジェクト）を作れることになります。

「14 はじめての iOS アプリの作成」と同様に、「ObjectEx」という名前のプロジェクトを生成後、UIView クラスを継承した「ObjectEx.swift」を追加してください。

## 2-2-2 ソースコード

### ◆ ObjectExクラス

ObjectEx クラスは、プログラムの本体となるクラスです。

#### リスト 2-2-1 ObjectEx.swift

```
import UIKit

// オブジェクト
class ObjectEx: UIView {

    // 描画時に呼ばれる
    override func drawRect(rect: CGRect) {
        // オブジェクト型の変数の定義 ① ②
        let calendar: NSCalendar =
            NSCalendar(calendarIdentifier: NSGregorianCalendar)!

        // 取得情報フラグの準備 ③
        let flags =
            NSCalendarUnit.YearCalendarUnit | // 年
            NSCalendarUnit.MonthCalendarUnit | // 月
            NSCalendarUnit.DayCalendarUnit | // 日
            NSCalendarUnit.HourCalendarUnit | // 時
            NSCalendarUnit.MinuteCalendarUnit // 分
    }
}
```

```

        image: UIImage(named: "sample.png")!)

        // コンポーネントの配置
        self.view.addSubview(imageView)
    }

    // ラベルの生成 ①
    func makeLabel(pos: CGPoint, text: NSString, font: UIFont) -> UILabel {
        let label = UILabel()
        label.frame = CGRectMake(pos.x, pos.y, 9999, 9999) // 領域
        label.text = text // テキスト
        label.font = font // フォント
        label.textAlignment = NSTextAlignment.Left // 配置
        label.lineBreakMode = NSLineBreakMode.ByWordWrapping // 改行
        label.numberOfLines = 0 // 行数
        label.sizeToFit() // ラベルとテキストの幅と高さをあわせる ②
        return label
    }

    // イメージビューの生成 ④
    func makeImageView(frame: CGRect, image: UIImage) -> UIImageView {
        let imageView = UIImageView()
        imageView.frame = frame // 領域
        imageView.image = image // イメージ
        return imageView
    }
}

```

## 4-1-4 ソースコードの解説

### ① ラベルの生成

「ラベル」を生成するには「UILabel クラス」を使います。主なプロパティは次の通りです。

UIViewクラスのプロパティ	解説
var frame: CGRect	領域
var backgroundColor: UIColor?	背景色

表 4-1-1 UIViewクラス (UILabelクラスで継承) の主なプロパティ

UILabelクラスのプロパティ	解説
var text: String!	テキスト
var font: UIFont!	フォント
var textAlignment: NSTextAlignment	テキストの配置指定 (表4-1-3参照)

//YES/NO ダイアログの生成

```
let btnYesNo = makeButton(CGRectMake(dx+60, 70, 200, 40),
    text: "Yes/No ダイアログ表示", tag: BTN_YESNO)
self.view.addSubview(btnYesNo)
```

// アクションシート表示ボタンの生成

```
let btnSheet = makeButton(CGRectMake(dx+60, 120, 200, 40),
    text: "アクションシート表示", tag: BTN_SHEET)
self.view.addSubview(btnSheet)
```

// イメージボタンの生成 ④

```
let btnImage = makeButton(CGRectMake(dx+103, 170, 114, 114),
    image: UIImage(named: "sample.png")!, tag: BTN_IMAGE)
self.view.addSubview(btnImage)
```

```
}
```

// ボタンクリック時に呼ばれる ③

```
func onClick(sender: UIButton) {
    if sender.tag == BTN_ALERT {
        // アラートの表示
        showAlert("", text: "アラート表示ボタンを押した ")
    } else if sender.tag == BTN_YESNO {
        // Yes/No ダイアログの表示
        showYesNoDialog("", text: "Yes/No ダイアログ表示ボタンを押した ")
    } else if sender.tag == BTN_SHEET {
        // アクションシートの表示
        showActionSheet("アクションシートの表示", text :nil)
    } else if sender.tag == BTN_IMAGE {
        // アラートの表示
        showAlert(nil, text: "イメージボタンを押した ")
    }
}
```

// ボタンの生成 ①

```
func makeButton(frame: CGRect, text: NSString, tag: Int) -> UIButton {
    let button = UIButton.buttonWithType(UIButtonType.System) as UIButton
    button.frame = frame // 領域
    button.setTitle(text, forState: UIControlState.Normal) // タイトル
    button.tag = tag // タグ ②
    button.addTarget(self, action: "onClick:",
        forControlEvents: UIControlEvents.TouchUpInside) // ターゲット ③
    return button
}
```

// イメージボタンの生成 ④

```
func makeButton(frame: CGRect, image: UIImage, tag: Int) -> UIButton {
    let button = UIButton.buttonWithType(UIButtonType.Custom) as UIButton
    button.frame = frame // 領域
    button.setImage(image, forState: UIControlState.Normal) // イメージ
    button.tag = tag // タグ
}
```

「14 はじめての iOS アプリの作成」と同様に、「WebViewEx」という名前のプロジェクトを生成します。

## 4-6-2 ソースコード

### ◆ ViewControllerクラス

ViewController クラスは、プログラムの本体となるクラスです。

リスト 4-6-1 ViewController.swift

```
import UIKit

//Web ビュー
class ViewController: UIViewController, UIWebViewDelegate {
    var _webView: UIWebView? //Web ビュー

    // ロード完了時に呼ばれる
    override func viewDidLoad() {
        super.viewDidLoad()

        //Web ビューの生成
        _webView = makeWebView(CGRectMake(0, 20,
            self.view.frame.width, self.view.frame.height-20))
        self.view.addSubview(_webView!)

        // インジケータの表示 ②
        UIApplication.sharedApplication(
            ).networkActivityIndicatorVisible = true

        //HTML の読み込み ③
        var url: NSURL = NSURL(string: "http://npaka.net")!
        var urlRequest: NSURLRequest = NSURLRequest(URL: url)
        _webView?.loadRequest(urlRequest)
    }

    //Web ビューの生成
    func makeWebView(frame: CGRect) -> UIWebView {
        //Web ビューの生成 ①
        let webView = UIWebView()
        webView.frame = frame
        webView.backgroundColor = UIColor.blackColor()
        webView.scalesPageToFit = true // ページをフィットさせるかどうか
        webView.autoresizingMask = // ビューサイズの自動調整
            UIViewAutoresizing.FlexibleRightMargin |
            UIViewAutoresizing.FlexibleTopMargin |
            UIViewAutoresizing.FlexibleLeftMargin |
            UIViewAutoresizing.FlexibleBottomMargin |
```

```

        else if file.hasPrefix(" ファイル ") {
            showAlert(nil, text: file)
        }

        // テーブルのセルの選択解除 ⑩
        tableView.deselectRowAtIndexPath(
            tableView.indexPathForSelectedRow()!, animated: true)
    }

//=====
//UITableViewDataSource
//=====

// セルの数取得時に呼ばれる ⑦
func tableView(tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int {
    let files = _folders[_folderName!]
    return files!.count
}

// セルの取得時に呼ばれる ⑦
func tableView(tableView: UITableView, cellForRowAtIndexPath
    indexPath: NSIndexPath) -> UITableViewCell {
    // テーブルのセルの生成 ⑧
    var cell = tableView.dequeueReusableCellWithIdentifier(
        "table_cell") as? UITableViewCell
    if cell == nil {
        cell = UITableViewCell(style: UITableViewCellStyle.Default,
            reuseIdentifier: "table_cell")
    }

    // テーブルのセルの設定 ⑨
    var files = _folders[_folderName!]
    cell!.textLabel.text = files![indexPath.row]
    if cell!.textLabel.text!.hasPrefix(" フォルダ ") {
        cell!.accessoryType =
            UITableViewCellAccessoryType.DisclosureIndicator
    }
    return cell!
}
}

```

```
//=====
// ファイルの読み書き
//=====
// 文字列をバイト配列に変換 ①
func str2data(str: NSString) -> NSData? {
    return str.dataUsingEncoding(NSUTF8StringEncoding)
}

// バイト配列を文字列に変換 ②
func data2str(data: NSData) -> NSString {
    return NSString(data: data, encoding: NSUTF8StringEncoding)!
}

// バイト配列の書き込み ③
func data2file(data: NSData, fileName: NSString) -> Bool {
    var path = NSHomeDirectory().stringByAppendingPathComponent("Documents")
    path = path.stringByAppendingPathComponent(fileName)
    return data.writeToFile(path, atomically: true)
}

// バイト配列の読み込み ④
func file2data(fileName: NSString) -> NSData? {
    var path = NSHomeDirectory().stringByAppendingPathComponent("Documents")
    path = path.stringByAppendingPathComponent(fileName)
    return NSData(contentsOfFile: path,
        options: NSDataReadingOptions.DataReadingMappedIfSafe, error: nil)
}

//=====
// UITextFieldDelegate
//=====
// 改行ボタン押下時に呼ばれる
func textFieldShouldReturn(sender: UITextField) -> Bool {
    // キーボードを隠す
    self.view.endEditing(true)
    return true
}
}
```

エンコーディングに指定する定数は、表5-1-1と同じです。

今回はバイト配列を文字列に変換する `data2str()` メソッドを定義しています。

```
func data2str(data: NSData) -> NSString {  
    return NSString(data: data, encoding: NSUTF8StringEncoding)!  
}
```

### 3 バイト配列の書き込み

バイト配列をファイルに書き込むには、「NSData クラス」の `writeToFile()` メソッドを使います。

#### NSDataクラス

```
func writeToFile(path: String, atomically atomically: Bool) -> Bool
```

機能：ファイルへのバイト配列の書き込み

引数：path                      書き込み先のパス

         atomically          安全に書き込むか

戻り値：成功したかどうか

書き込み先のパスは、「NSHomeDirectory 関数」で「ホーム」のパスを取得し、`stringByAppendingPathComponent()` メソッドで「Documents」と「ファイル名」を追加しています。

`stringByAppendingPathComponent()` メソッドは、文字列追加時にパスのセパレータ "/" を自動的に付加します。tmp フォルダに書き込む時は、「Documents」のかわりに「tmp」を指定します。

```
func NSHomeDirectory() -> String!
```

機能：ホームのパスの取得

戻り値：ホームのパス

#### NSStringクラス

```
func stringByAppendingPathComponent(aString: String) -> String
```

機能：セパレータ付きでファイル名・フォルダ名を追加

引数：aString          追加するファイル名・フォルダ名

戻り値：パス

今回はバイト配列をファイルに書き込む `data2file()` メソッドを定義しています。

```

        http2data(URL_TEST)
    }
}

//=====
//HTTP 通信
//=====
// バイト配列を文字列に変換
func data2str(data: NSData) -> NSString {
    return NSString(data: data, encoding: NSUTF8StringEncoding)!
}

//HTTP 通信
func http2data(url: String) {
    // インジケーターのアニメーションの開始 ③
    _indicator?.startAnimating()

    //HTTP 通信 ①
    let URL = NSURL(string: url)!
    let request = NSURLRequest(URL: URL)
    NSURLConnection.sendAsynchronousRequest(request,
        queue: NSOperationQueue.mainQueue(),
        completionHandler: self.fetchResponse)
}

// レスポンス受信時に呼ばれる
func fetchResponse(res: NSURLResponse!, data: NSData!, error: NSError!) {
    //UI の更新
    if error == nil {
        _textField!.text = data2str(data)
    } else {
        _textField!.text = "通信エラー"
    }

    // インジケーターのアニメーションの停止 ③
    _indicator?.stopAnimating()
}
}

```



```

        self.presentViewController(alert, animated: true, completion: nil)
    }

//=====
//MultipeerConnectivity
//=====
    // ボタンの指定
    func updateButton() {
        // セッション未接続
        if _state == MCSessionState.NotConnected {
            _btnAdvertise!.alpha = 1.0
            _btnBrowse!.alpha = 1.0
            _btnSend!.alpha = 0.0
            _btnDisconnect!.alpha = 0.0
        }
        // セッション接続中
        else if _state == MCSessionState.Connecting {
            _btnAdvertise!.alpha = 0.0
            _btnBrowse!.alpha = 0.0
            _btnSend!.alpha = 0.0
            _btnDisconnect!.alpha = 0.0
        }
        // セッション接続
        else if _state == MCSessionState.Connected {
            _btnAdvertise!.alpha = 0.0
            _btnBrowse!.alpha = 0.0
            _btnSend!.alpha = 1.0
            _btnDisconnect!.alpha = 1.0
        }
    }

    // テキストフィールドの更新
    func updateTextField(text: String) {
        _textField!.text = text
    }

    // 文字列をバイト配列に変換
    func str2data(str: NSString) -> NSData? {
        return str.dataUsingEncoding(NSUTF8StringEncoding)
    }

    // バイト配列を文字列に変換
    func data2str(data: NSData) -> NSString {
        return NSString(data: data, encoding: NSUTF8StringEncoding)!
    }

//=====
//UITextFieldDelegate
//=====
    // 改行ボタン押下時に呼ばれる

```

```

        self.text = NSString(data: contents as NSData,
                               encoding: NSUTF8StringEncoding)!
        return true
    }

    // ドキュメント書き込み時に呼ばれる ⑦
    override func contentsForType(typeName: String,
                                     error outError: NSErrorPointer) -> AnyObject? {
        return self.text.dataUsingEncoding(NSUTF8StringEncoding)
    }
}

```

## 5-6-5 ソースコードの解説

### ① iCloudへのキーバリューの書き込み

iCloud へのキーバリュー型のデータの書き込みを行うには、「NSUbiquitousKeyValueStore クラス」を使います。NSUbiquitousKeyValueStore クラスの使い方は、NSUserDefaults クラスと同じです。NSUbiquitousKeyValueStore クラスの `defaultStore()` メソッドで NSUbiquitousKeyValueStore オブジェクトを取得します。

NSUbiquitousKeyValueStoreクラス

```
class func defaultStore() -> NSUbiquitousKeyValueStore
```

機能：NSUbiquitousKeyValueStore オブジェクトの取得

戻り値：NSUbiquitousKeyValueStore オブジェクト

NSUbiquitousKeyValueStore オブジェクトにキーと値の組み合わせで保存する情報を指定します。値の型ごとに利用するメソッドは変わります。

NSUbiquitousKeyValueStoreクラスのメソッド	解説
func setInteger(value: Int, forKey akey: String)	Int型の値をプリファレンスに指定
func setFloat(value: Float, forKey akey: String)	Float型の値をプリファレンスに指定
func setDouble(value: Double, forKey akey: String)	Double型の値をプリファレンスに指定
func setBool(value: Bool, forKey akey: String)	Bool型の値をプリファレンスに指定
func setObject(value: AnyObject!, forKey akey: String)	AnyObject型の値をプリファレンスに指定
func setURL(url: NSURL!, forKey akey: String)	NSURL型の値をプリファレンスに指定

表 5-6-1 値の型をプリファレンスに指定するメソッド

最後に `synchronize()` メソッドを呼ぶことで、指定した値が保存されます。

```

if device.orientation == UIDeviceOrientation.LandscapeLeft {
    _orientation = "横 (左90度回転)"
} else if device.orientation == UIDeviceOrientation.LandscapeRight {
    _orientation = "横 (右90度回転)"
} else if device.orientation == UIDeviceOrientation.PortraitUpsideDown {
    _orientation = "縦 (上下逆)"
} else if device.orientation == UIDeviceOrientation.Portrait {
    _orientation = "縦"
} else if device.orientation == UIDeviceOrientation.FaceUp {
    _orientation = "画面が上向き"
} else if device.orientation == UIDeviceOrientation.FaceDown {
    _orientation = "画面が下向き"
}
}

```

// モーション通知時の処理

```

func updateMotion(motion: CMDeviceMotion) {
    var str = NSMutableString()

```

// 端末の加速度の取得 ②

```

var gravity = motion.gravity
str.appendString("AccelerometerEx\n")

```

// 加速度にローパスフィルタをあてる ③

```

_aX = (Float(gravity.x)*FILTERING_FACTOR) + (_aX*(1.0-FILTERING_FACTOR))
_aY = (Float(gravity.y)*FILTERING_FACTOR) + (_aY*(1.0-FILTERING_FACTOR))
_aZ = (Float(gravity.z)*FILTERING_FACTOR) + (_aZ*(1.0-FILTERING_FACTOR))

```

// 加速度の更新

```

str.appendFormat("X 軸加速度 :%+.2f\n", _aX)
str.appendFormat("Y 軸加速度 :%+.2f\n", _aY)
str.appendFormat("Z 軸加速度 :%+.2f\n", _aZ)
str.appendString("\n")

```

```

if _motionManager!.gyroAvailable {

```

// 端末の傾きの取得 ④

```

let attitude = motion.attitude

```

// 端末の傾きの更新

```

str.appendFormat("X 軸回転角度 :%+.2f\n", attitude.pitch*180/M_PI)
str.appendFormat("Y 軸回転角度 :%+.2f\n", attitude.yaw*180/M_PI)
str.appendFormat("Z 軸回転角度 :%+.2f\n", attitude.roll*180/M_PI)
str.appendString("\n")
}

```

// 端末の向きを更新

```

str.appendFormat(" 端末の向き : %@\n", _orientation)
str.appendString("\n")

```

```

        self.presentViewController(alert, animated: true, completion: nil)
    }

//=====
// イメージピッカーのオープン
//=====
// イメージピッカーのオープン
func openPicker(sourceType: UIImagePickerControllerSourceType) {
    // カメラとフォトアルバムの利用可能チェック ①
    if !UIImagePickerController.isSourceTypeAvailable(sourceType) {
        showAlert(nil, text: " 利用できません ")
        return
    }

    // イメージピッカーの生成 ②
    let picker = UIImagePickerController()
    picker.sourceType = sourceType
    picker.delegate = self

    // ビューコントローラのビューを開く
    let model = UIDevice.currentDevice().model as NSString // モデル名
    if !model.containsString("iPad") {
        presentViewController(picker, animated: true, completion: nil)
    } else {
        let popoverCtl = UIPopoverController(contentViewController: picker)
        popoverCtl.presentPopoverFromRect(_btnCamera!.bounds, inView: _btnCamera!,
            permittedArrowDirections: UIPopoverArrowDirection.Any,
            animated: true)
    }
}

// フォト書き込み完了時に呼ばれる ⑥
func finishExport(image: UIImage,
    didFinishSavingWithError error: NSError!,
    contextInfo: AnyObject) {
    NSOperationQueue.mainQueue().addOperationWithBlock {
        if error == nil {
            self.showAlert(nil, text: " フォト書き込み完了 ")
        } else {
            self.showAlert(nil, text: " フォト書き込み失敗 ")
        }
    }
}

//=====
// UIImagePickerControllerDelegate
//=====
// イメージピッカーのイメージ取得時に呼ばれる ④
func imagePickerController(picker: UIImagePickerController,
    didFinishPickingMediaWithInfo info: [NSObject : AnyObject])

```

```

tableView.delegate = self
tableView.dataSource = self
return tableView;
}

// アラートの表示 (
func showAlert(title: NSString?, text: NSString?) {
    let alert = UIAlertController(title: title, message: text,
        preferredStyle: UIAlertControllerStyle.Alert)
    alert.addAction(UIAlertAction(title: "OK",
        style: UIAlertActionStyle.Default, handler: nil))
    self.presentViewController(alert, animated: true, completion: nil)
}

//URL スキームのオープン ①
func openURL(urlPath: String) -> Bool {
    var url = NSURL(string: urlPath)!
    return UIApplication.sharedApplication().openURL(url)
}

//=====
// アドレスの読み込み
//=====
// アドレスの読み込み
func readAddress() {
    _names.removeAll()
    _tels.removeAll()

    //ABAddressBook オブジェクトの取得 ②
    var book: ABAddressBook =
        ABAddressBookCreateWithOptions(nil, nil).takeUnretainedValue()

    // アクセス許可状態の確認 ③
    let status = ABAddressBookGetAuthorizationStatus()
    // アクセス許可を求めたことがない
    if status == ABAuthorizationStatus.NotDetermined {
        ABAddressBookRequestAccessWithCompletion(book, {
            (granted: Bool, error: NSError!) in
            if granted {
                NSOperationQueue.mainQueue().addOperationWithBlock {
                    self.readAddress()
                }
            } else {
                self.showAlert(nil, text:
                    "「設定→プライバシー→連絡先→AddressEx」をオンにしてください")
            }
        })
    }
    return
}
// アクセス許可されていない

```

```

else if status != ABAuthorizationStatus.Authorized {
    showAlert(nil, text:
        "「設定→プライバシー→連絡帳→ AddressEx」をオンにしてください")
    return
}

```

// アドレスのレコードの取得 ④

```

let records: NSArray =
    ABAddressBookCopyArrayOfAllPeople(book).takeUnretainedValue()
for record in records {

```

// レコードからの名前の取得 ⑤

```

var firstName =
    ABRecordCopyValue(record, kABPersonFirstNameProperty) == nil ? "" :
    ABRecordCopyValue(record, kABPersonFirstNameProperty)
        .takeUnretainedValue() as String
var lastName =
    ABRecordCopyValue(record, kABPersonLastNameProperty) == nil ? "" :
    ABRecordCopyValue(record, kABPersonLastNameProperty)
        .takeUnretainedValue() as String
_names.append("\(lastName) \(firstName)")

```

// レコードからの電話番号の取得 ⑥

```

var tels: ABMultiValue =
    ABRecordCopyValue(record, kABPersonPhoneProperty)
        .takeUnretainedValue()
var tel = ""
if ABMultiValueGetCount(tels) > 0 {
    tel = ABMultiValueCopyValueAtIndex(tels, 0)
        .takeUnretainedValue() as String
}
_tels.append(tel)
}

```

// テーブルビューの更新 ⑦

```

_tableView!.reloadData()
}

```

```

//=====
//UITableViewDelegate
//=====

```

// セルの高さの取得

```

func tableView(tableView: UITableView,
    heightForRowAtIndexPath indexPath: NSIndexPath) -> CGFloat {
    return 50
}

```

// セルのクリック時に呼ばれる

```

func tableView(tableView: UITableView,
    didSelectRowAtIndexPath indexPath: NSIndexPath) {

```

```

// 選択解除
tableView.deselectRowAtIndexPath(
    tableView.indexPathForSelectedRow()!, animated: true)

// 通話
let tel = _tels[indexPath.row]
if countElements(tel) > 0 {
    if !openURL("tel:\(tel)") {
        showAlert(nil, text: "通話機能がありません ")
    }
}
}

// セルの取得時に呼ばれる
func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    // テーブルのセルの生成
    var cell = tableView.dequeueReusableCellWithIdentifier("table_cell")
    as? UITableViewCell
    if cell == nil {
        cell = UITableViewCell(style: UITableViewCellStyle.Subtitle,
            reuseIdentifier: "table_cell")
    }
    cell!.textLabel.text = _names[indexPath.row]
    cell!.detailTextLabel?.text = _tels[indexPath.row]
    return cell!
}

//=====
//UITableViewDataSource
//=====
// セルの数取得時に呼ばれる
func tableView(tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int {
    return _names.count
}
}

```

### 6-4-3 ソースコードの解説

#### ① URLスキームのオープン

iOS の他のアプリを起動するには、「UIApplication クラス」の `openURL()` メソッドに「URL スキーム」と呼ばれる特別な書式の文字列を渡します。

```
import MobileCoreServices
```

```
// アクション
```

```
class ViewController: UIViewController {
```

```
    // 定数
```

```
    let BTN_ACTION = 0 // アクション
```

```
    // 変数
```

```
    var _imageView: UIImageView? // イメージビュー
```

```
// ロード完了時に呼ばれる
```

```
    override func viewDidLoad() {
```

```
        super.viewDidLoad()
```

```
        let dx: CGFloat = (UIScreen.mainScreen().bounds.size.width-320)/2
```

```
        // イメージビューの生成
```

```
        _imageView = UIImageView(CGRectMake(dx+120, 20, 80, 80),
```

```
            image: UIImage(named: "sample.png")!)
```

```
        self.view.addSubview(_imageView!)
```

```
        // アラート表示ボタンの生成
```

```
        let btnAlert = makeButton(CGRectMake(dx+60, 120, 200, 40),
```

```
            text: "Action", tag: BTN_ACTION)
```

```
        self.view.addSubview(btnAlert)
```

```
    }
```

```
// イメージビューの生成
```

```
func makeImageView(frame: CGRect, image: UIImage) -> UIImageView {
```

```
    let imageView = UIImageView()
```

```
    imageView.frame = frame
```

```
    imageView.image = image
```

```
    return imageView
```

```
}
```

```
// ボタンクリック時に呼ばれる
```

```
func onClick(sender: UIButton) {
```

```
    if sender.tag == BTN_ACTION {
```

```
        dispatch_async(dispatch_get_global_queue(
```

```
            DISPATCH_QUEUE_PRIORITY_DEFAULT,0), {
```

```
            // アクティビティビューを開く ①
```

```
            let activityViewController = UIActivityViewController(
```

```
                activityItems: [self._imageView!.image!],
```

```
                applicationActivities: nil)
```

```
            activityViewController.completionWithItemsHandler = {
```

```
                (activityType: String!, completed: Bool,
```

```
                returnedItems: Array!, error: NSError!) in
```

```
                if activityType == "net.npaka.ActionEx.ActionExt" &&
```

```
                    completed {
```

```
                        // NSItemProvider オブジェクトの取得 ②
```

```
                        let extensionItem = returnedItems[0] as NSExtensionItem
```



```
//=====
// ロード完了時に呼ばれる
override func viewDidLoad() {
    super.viewDidLoad()
    let dx: CGFloat = (UIScreen.mainScreen().bounds.size.width-320)/2

    // テキストフィールドの生成
    _textField = makeTextField(CGRectMake(dx+10, 20, 300, 30), text: "")
    _textField!.text = "これはテストです。"
    self.view.addSubview(_textField!)

    // 書き込みボタンの生成
    let btnWrite = makeButton(CGRectMake(dx+60, 60, 200, 40),
        text: "書き込み", tag: BTN_WRITE)
    self.view.addSubview(btnWrite)
}

// テキストフィールドの生成
func makeTextField(frame: CGRect, text: String) -> UITextField {
    let textField = UITextField()
    textField.frame = frame
    textField.text = text
    textField.borderStyle = UITextBorderStyle.RoundedRect
    textField.keyboardType = UIKeyboardType.Default
    textField.returnKeyType = UIReturnKeyType.Done
    textField.delegate = self
    return textField
}

// テキストボタンの生成
func makeButton(frame: CGRect, text: NSString, tag: Int) -> UIButton {
    let button = UIButton.buttonWithType(UIButtonType.System) as UIButton
    button.frame = frame
    button.setTitle(text, forState: UIControlState.Normal)
    button.tag = tag
    button.addTarget(self, action: "onClick:",
        forControlEvents: UIControlEvents.TouchUpInside)
    return button
}

// ボタンクリック時に呼ばれる
func onClick(sender: UIButton) {
    if sender.tag == BTN_WRITE {
        //App Group の共有領域に書き込み ①
        let userDefaults = NSUserDefaults(suiteName: "group.TodayEx")!
        userDefaults.setObject(_textField!.text, forKey: "text")
        userDefaults.synchronize()
    }
}
}
```

```
//App Group の共有領域からの読み込み ⑥
let defaults:NSUserDefaults = UserDefaults(suiteName: "group.TodayEx")!
_label!.text = defaults.stringForKey("text")
}

// ウィジェットの画面更新時に呼ばれる ⑦
func widgetPerformUpdateWithCompletionHandler(
    completionHandler: ((NCUpdateResult) -> Void)!) {
    completionHandler(NCUpdateResult.NewData)
}
}
```

## 6-7-5 TodayExのソースコードの解説

### ① App Groupの共有領域に書き込み

App Group の共有領域にデータを書き込むには、プリファレンスと同じく「NSUserDefaults クラス」を使います。イニシャライザでグループの ID を指定します。

NSUserDefaultsクラス

```
init(suiteName suiteName: String)
機能 : NSUserDefaults オブジェクトの生成
引数 : suiteName グループの ID
戻り値 : NSUserDefaults オブジェクト
```

今回は、setObject () メソッドで文字列を追加して、synchronized () メソッドで保存しています。

```
let userDefaults = UserDefaults(suiteName: "group.TodayEx")!
userDefaults.setObject(_textField!.text, forKey: "text")
userDefaults.synchronize()
```

## 6-7-6 TodayExtのソースコードの解説

### ② ラベルの定義

「@IBOutlet」付きでラベルの定義を行います。

```

let BTN_MOVE = 0 // 移動

// 変数
var _imageView: UIImageView? // イメージビュー
var _animeIdx = 1           // アニメインデックス

// ロード完了時に呼ばれる
override func viewDidLoad() {
    super.viewDidLoad()
    let dx: CGFloat = (UIScreen.mainScreen().bounds.size.width-320)/2

    // イメージビューの生成
    _imageView = makeImageView(CGRectMake(dx+40, 60, 240, 240),
        image: UIImage(named: "sample.png")!)
    self.view.addSubview(_imageView!)

    // ボタンの生成
    let button = makeButton(CGRectMake(dx+60, 360, 200, 40),
        text: "UIView アニメーション ", tag: BTN_MOVE)
    self.view.addSubview(button)
}

// ボタンクリック時に呼ばれる
func onClick(sender: UIButton) {
    let dx: CGFloat = (UIScreen.mainScreen().bounds.size.width-320)/2
    if sender.tag == BTN_MOVE {
        // アニメーション前の平行緯度・回転角度・透明度の指定 ❶
        if _animeIdx == 1 {
            _imageView!.transform = CGAffineTransformMakeScale(0.5, 0.5)
            _imageView!.alpha = 0.8
        } else if _animeIdx == 2 {
            var trans = CGAffineTransformMakeRotation(
                CGFloat(180.0*(M_PI/180.0)))
            _imageView!.transform = CGAffineTransformScale(trans, 0.5, 0.5)
            _imageView!.alpha = 0.8
        } else if _animeIdx == 3 {
            _imageView!.transform = CGAffineTransformMakeTranslation(0, 400)
            _imageView!.alpha = 0.8
        } else if _animeIdx == 4 {
            _imageView!.frame = CGRectMake(dx+60, 340, 200, 40) ❷
        }

        //UIView アニメーションの設定 ❷
        UIView.beginAnimations("anime0", context:nil)
        UIView.setAnimationDuration(0.5)
        UIView.setAnimationCurve(UITableViewAnimationCurve.EaseInOut)
        UIView.setAnimationRepeatCount(0)
        UIView.setAnimationRepeatAutoreverses(false)

        //UIView アニメーションのデリゲートの指定 ❸
    }
}

```

```

layer.frame = CGRectMake(0, 0, 80, 80)
layer.position = CGPointMake(w/2, h/2)
layer.contents = image!.CGImage

// ビューへのレイヤーの追加 ②
self.view.layer.addSublayer(layer)

// レイヤーアニメーションの生成 ③
let anime = CABasicAnimation(keyPath: "transform")
anime.duration = 0.5
anime.repeatCount = 999
anime.autoreverses = true
anime.fromValue = NSValue(CATransform3D: CATransform3DIdentity)
anime.toValue = NSValue(CATransform3D:
    CATransform3DMakeRotation(CGFloat(M_PI), 0, 1, 0)) ④

// レイヤーへのレイヤーアニメーションの追加 ⑤
layer.addAnimation(anime, forKey: "rotation")
}
}

```

## 7-4-4 ソースコードの解説

### ① レイヤーの生成

レイヤーを生成するには「CALayer クラス」を使います。  
 主なプロパティは次の通りです。

CALayerクラスのプロパティ	解説
var frame: CGRect	領域
var position: CGPoint	位置
var transform: CATransform3D	3次元のアフィン変換
var contents: AnyObject!	コンテンツ

表 7-4-1 CALayerクラスの主なプロパティ

コンテンツには、CGImage オブジェクトを指定できます。CGImage オブジェクトは、UIImage クラスの CGImage プロパティで取得できます。

UIImageクラスのプロパティ	解説
var CGImage: CGImage!	CGImageオブジェクト

表 7-4-2 CGImageオブジェクトの取得

## リスト 8-1-2 GameViewController.swift

```
import UIKit
import SpriteKit

// ゲームビューコントローラ
class GameViewController: UIViewController {

    // ロード完了時に呼ばれる
    override func viewDidLoad() {
        super.viewDidLoad()

        //SpriteKit のビューの設定 ❶
        let skView = self.view as SKView
        skView.showsFPS = true //FPS の表示
        skView.showsNodeCount = true // ノード数の表示

        // シーンを追加 ❷
        let scene = GameScene(size: skView.frame.size)
        skView.presentScene(scene)
    }
}
```

## ◆ GameSceneクラス

GameScene クラスは、シーンとなるクラスです。

## リスト 8-1-3 GameScene.swift

```
import UIKit
import SpriteKit

// ゲームシーン
class GameScene: SKScene { ❸

    // 初期化時に呼ばれる
    override func didMoveToView(view: SKView) {

        // シーンの設定 ❹
        self.backgroundColor = SKColor.whiteColor() // 背景色
        self.scaleMode = SKSceneScaleMode.AspectFit // スケールモード

        // ラベルを追加 ❺
        let label = SKLabelNode(fontNamed: "ArialMT")
        label.text = "Hello SpriteKit!"
        label.fontSize = 20
        label.fontColor = SKColor.blackColor()
        label.position = CGPointMake(self.frame.size.width/2,
                                     self.frame.size.height/2)
    }
}
```

「8-1 SpriteKit」と同様に、「SceneTransitionEx」という名前のプロジェクトを生成してください。

## 8-2-2 画像ファイルの準備

今回使用する画像ファイルは、次の2種類です。



図 8-2-3 sample0.png - 80x80 ポイント



図 8-2-4 sample1.png - 80x80 ポイント

## 8-2-3 ソースコード

### ◆ GameViewControllerクラス

GameViewController クラスは、ビューコントローラとなるクラスです。

#### リスト 8-2-1 GameViewController.swift

```
import UIKit
import SpriteKit

// ゲームビューコントローラ
class GameViewController: UIViewController {

    // ロード完了時に呼ばれる
    override func viewDidLoad() {
        super.viewDidLoad()

        // ビューの設定
        let skView = self.view as SKView
        skView.showsFPS = true //FPS の表示
        skView.showsNodeCount = true // ノード数の表示

        // シーンの追加
        let scene = GameScene(size: skView.frame.size)
        scene.setTransitionType(0) ①
        skView.presentScene(scene)
```

```
}  
}
```

## ◆ GameSceneクラス

GameScene クラスは、シーンとなるクラスです。

### リスト 8-2-2 GameScene.swift

```
import UIKit  
import SpriteKit  
  
// ゲームシーン  
class GameScene: SKScene {  
    var _type = 0// トランジション種別  
  
    // 初期化時に呼ばれる  
    override func didMoveToView(view: SKView) {  
        // シーンの設定  
        self.backgroundColor = SKColor.whiteColor()  
        self.scaleMode = SKSceneScaleMode.AspectFit  
    }  
  
    // トランジション種別の指定 ①  
    func setTransitionType(type: Int) {  
        _type = type  
  
        // スプライトの追加  
        let texture = SKTexture(imageNamed: "sample\\(_type%2).png")  
        let sprite = SKSpriteNode(texture: texture)  
        sprite.position = CGPointMake(  
            self.frame.size.width/2, self.frame.size.height/2)  
        sprite.size = CGSizeMake(320, 320)  
        self.addChild(sprite)  
    }  
  
    // タッチ時に呼ばれる  
    override func touchesBegan(touches: NSSet, withEvent event: UIEvent) {  
        // シーンの生成  
        let scene = GameScene(size: self.frame.size)  
        scene.setTransitionType((_type+1)%12) ①  
  
        // トランジションオブジェクトの生成 ②  
        var trans: SKTransition? = nil  
        switch _type {  
        case 1:
```

```

let skView = self.view as SKView;
skView.showsFPS = true;           //FPS の表示
skView.showsNodeCount = true;    // ノード数の表示

// シーンを追加
let scene = GameScene(size: skView.frame.size)
skView.presentScene(scene);
    }
}

```

## ◆ GameSceneクラス

GameScene クラスは、シーンとなるクラスです。

### リスト 8-3-2 GameScene.swift

```

import UIKit
import SpriteKit

// ゲームシーン
class GameScene: SKScene {

    // 初期化時に呼ばれる
    override func didMoveToView(view: SKView) {
        // シーンの設定
        self.backgroundColor = SKColor.blackColor() // 背景色
        self.scaleMode = SKSceneScaleMode.AspectFit // スケールモード

        // バーの生成
        makeBar()

        // ボール追加アクションの追加 ③
        NSTimer.scheduledTimerWithTimeInterval(0.1, target: self,
            selector: "addBall", userInfo: nil, repeats: true)
    }

    // 物理シミュレーションの処理後に呼ばれる
    override func didSimulatePhysics() {
        // ボールが画面下に落ちたら削除 ④
        self.enumerateChildNodesWithName("ball", usingBlock: {
            (node: SKNode!, stop: UnsafeMutablePointer<ObjCBool>) in
            if node.position.y < 0 {
                node.removeFromParent()
            }
        })
    }
}

```



```

ambientLightNode.light!.type = SCNLightTypeAmbient
ambientLightNode.light!.color = UIColor.darkGrayColor()
scene.rootNode.addChildNode(ambientLightNode)

// 3D モデルの読み込みとシーンへの追加 ❷
let modelScene = SCNScene(named: "art.scnassets/model.dae")!
let modelNode = modelScene.rootNode.childNodeWithName("model",
    recursively: true)!
scene.rootNode.addChildNode(modelNode)
modelNode.position = SCNVector3(x: 0, y: 0, z: -80)
modelNode.scale = SCNVector3(x: 0.2, y: 0.2, z: 0.2)

// マテリアルの生成とジオメトリへの追加 ❸
let material = SCNMaterial()
material.diffuse.contents = UIImage(named: "texture")
material.specular.contents = UIColor.grayColor()
material.locksAmbientWithDiffuse = true
modelNode.geometry!.firstMaterial = material

// タップジェスチャーの追加 ❹
let tapGesture = UITapGestureRecognizer(target: self, action: "onTap:")
let gestureRecognizers = NSMutableArray()
gestureRecognizers.addObject(tapGesture)
gestureRecognizers.addObjectsFromArray(scnView.gestureRecognizers!)
scnView.gestureRecognizers = gestureRecognizers
}

// タップ時に呼ばれる
func onTap(gestureRecognizer: UIGestureRecognizer) {
    // タップしたノードの取得 ❺
    let scnView = self.view as SCNView
    let pos = gestureRecognizer.locationInView(scnView)
    let hitResults = scnView.hitTest(pos, options: nil)
    if hitResults!.count > 0 {
        let node = hitResults![0].node!

        // マテリアルへのアニメーションの追加 ❻
        let material = node.geometry!.firstMaterial
        SCNTransaction.begin()
        SCNTransaction.setAnimationDuration(0.5)
        SCNTransaction.setCompletionBlock {
            SCNTransaction.begin()
            SCNTransaction.setAnimationDuration(0.5)
            material!.emission.contents = UIColor.blackColor()
            SCNTransaction.commit()
        }
        material!.emission.contents = UIColor.yellowColor()
        SCNTransaction.commit()
    }
}
}

```

## 9-1-3 ソースコード

### ◆ ViewControllerクラス

ViewController クラスは、プログラムの本体となるクラスです。

#### リスト 9-1-1 ViewController.swift

```
import UIKit

// パズルゲーム
class ViewController: UIViewController {
    // 定数
    let BTN_START = 0 // スタート
    let SCREEN = UIScreen.mainScreen().bounds.size // 画面サイズ

    // 変数
    var _gameView: UIView? // ゲームビュー
    var _titleLabel: UILabel? // タイトルラベル
    var _piece = [UIImageView]() // ピース画像 ②
    var _data = [Int]() // ピース配置情報 ②
    var _shuffle: Int = 0 // シャッフル
    var _startButton: UIButton? // スタートボタン

    //=====
    //UI
    //=====
    // ロード完了時に呼ばれる
    override func viewDidLoad() {
        super.viewDidLoad()

        // ゲーム画面の XY 座標とスケールの指定 ①
        let vx: CGFloat = (SCREEN.width-360)/2
        let vy: CGFloat = (SCREEN.height-640)/2
        let scale = SCREEN.width/360
        _gameView = UIView()
        _gameView!.frame = CGRectMake(vx, vy, 360, 640)
        _gameView!.transform = CGAffineTransformMakeScale(scale, scale)
        self.view.addSubview(_gameView!)

        // 背景の生成
        let bg = makeImageView(CGRectMake(0, 0, 360, 640),
            image: UIImage(named: "bg.png")!)
        _gameView?.addSubview(bg)

        // 絵の背景の生成
        let picturebg = makeImageView(CGRectMake(29, 179, 302, 302),
```

```

        image: UIImage(named: "picturebg.png")!)
    _gameView?.addSubview(picturebg)

    // タイトルラベルの生成
    _titleLabel = makeLabel(CGRectMake(0, 90, 360, 70),
        text: "Moon Puzzle", font: UIFont.systemFont(ofSize: 48))
    _gameView?.addSubview(_titleLabel!)

    // 絵のビットマップの取得
    let picture = UIImage(named: "picture.png")!
    let piece = UIImage(named: "piece.png")!
    for var i = 0; i < 16; i++ {
        _piece.append(makePieceImageView(CGRectMake(
            CGFloat(30+(i%4)*75),
            CGFloat(180+Int(i/4)*75),
            75, 75),
            index: i, picture: picture, piece: piece))
        _data.append(i)
        _gameView?.addSubview(_piece[i])
    }

    // スタートボタンの生成
    _startButton = makeButton(CGRectMake(124, 455, 114, 114),
        image: UIImage(named: "start.png")!, tag: BTN_START)
    _gameView?.addSubview(_startButton!)
}

// ラベルの生成
func makeLabel(frame: CGRect, text: NSString, font: UIFont) -> UILabel {
    var label = UILabel()
    label.frame = frame
    label.text = text
    label.font = font
    label.textColor = UIColor.whiteColor()
    label.textAlignment = NSTextAlignment.Center
    label.lineBreakMode = NSLineBreakMode.ByWordWrapping
    label.numberOfLines = 0
    return label
}

// イメージビューの生成
func makeImageView(frame: CGRect, image: UIImage) -> UIImageView {
    var imageView = UIImageView()
    imageView.frame = frame
    imageView.image = image
    return imageView
}

// ピースイメージビューの生成
func makePieceImageView(frame: CGRect, index: Int,

```

## ◆ ActionViewクラス

ActionView クラスは、プログラム本体となるクラスです。

### リスト 9-2-2 ActionView.swift

```
import UIKit

// アクションゲーム
class ActionView: UIView {
    // シーン定数 ①
    let S_TITLE = 0    // タイトル
    let S_PLAY = 1     // プレイ
    let S_GAMEOVER = 2 // ゲームオーバー

    // 画面サイズ定数
    let SCREEN = UIScreen.mainScreen().bounds.size // 画面サイズ
    let SCALE: CGFloat = 1 // 画面スケール

    // システム
    var _init = 0          // 初期化 ①
    var _scene = 0         // シーン ①
    var _touchDown = false // タッチダウン
    var _g = Graphics()    // グラフィックス
    var _images = [UIImage]() // イメージ郡
    var _message: String?  // メッセージ
    var _score = 0         // スコア

    // プレイヤー ③
    var _playerY: CGFloat = 0 // Y座標
    var _jumpPow: CGFloat = 0 // ジャンプ力
    var _jumpAble = true      // ジャンプ可

    // ブロック ②
    var _blockNum: Int = 16 // 数
    var _blockDX: CGFloat = 0 // X 相対位置
    var _blockH: [CGFloat] = [CGFloat]() // 高さ

    //=====
    //UI
    //=====
    // 初期化
    required init(coder: NSCoder) {
        super.init(coder: coder)

        // 画面サイズの調整
        if SCREEN.width >= 768 {
            SCALE = 2
            SCREEN.width /= 2
        }
    }
}
```

```

        SCREEN.height /= 2
    }

    // イメージの取得
    for var i = 0; i < 4; i++ {
        _images.append(UIImage(named: "act\(i).png")!)
    }

    // 高さ
    for var i = 0; i < _blockNum; i++ {
        _blockH.append(0)
    }

    // タイマーの開始
    NSTimer.scheduledTimerWithTimeInterval(0.04,
        target: self, selector: "onTick:", userInfo: nil, repeats: true)
}

// 定期処理
func onTick(timer: NSTimer) {
    self.setNeedsDisplay()
}

//=====
// 描画
//=====
// 描画
override func drawRect(rect: CGRect) {
    // コンテキストの指定
    _g.setContext(UIGraphicsGetCurrentContext())
    _g.setScale(SCALE, SCALE)

    // バッファのクリア
    _g.setColor(0, 0, 0)
    _g.fillRect(0, 0, SCREEN.width, SCREEN.height)

    // シーンの初期化
    if _init >= 0 {
        _scene = _init

        // タイトル
        if _scene == S_TITLE {
            _message = "Touch Start"
            _score = 0
            _playerY = SCREEN.height-50
            _jumpPow = 0
            _jumpAble = true
            _blockDX = 0
            for var i = 0; i < _blockNum; i++ {_blockH[i] = 1}
        }
    }
}

```

```

}
// プレイ
else if _scene == S_PLAY {
    _message = nil
    _touchDown = false
}
// ゲームオーバー
else if _scene == S_GAMEOVER {
    _message = "Game Over"
}
_init = -1
}

// プレイ時の処理
if _scene == S_PLAY {
    // スコア加算
    _score++

    // 衝突判定
    if _playerY > SCREEN.height-_blockH[2]*50 {
        _init = S_GAMEOVER
        _jumpAble = false
    }
    // 上移動
    else if _jumpPow >= 0 {
        _playerY -= _jumpPow*2
        _jumpPow--
    }
    // 下移動 ③
    else {
        _playerY += 12
        _jumpAble = false
        if _blockH[2] != 0 &&
            _playerY > SCREEN.height-_blockH[2]*50 {
            _playerY = SCREEN.height-_blockH[2]*50
            _jumpAble = true
        }
    }
}
// ブロックのスクロール ②
_blockDX -= 10
if _blockDX == -50 {
    _blockDX = 0

    // 横方向の位置を戻し、高さを右隣のブロックと同じに
    for var i = 0; i < _blockNum-1; i++ {
        _blockH[i] = _blockH[i+1]
    }

    // 新規ブロックの高さの設定 ③
    var idx = rand(6)

```

```

//1つ前が穴のときは2つ前の高さ
if _blockH[_blockNum-2] == 0 {
    _blockH[_blockNum-1] = _blockH[_blockNum-3]
}
//1/6の確率で穴
else if idx == 0 {
    _blockH[_blockNum-1] = 0
}
//1/6の確率で1段上
else if idx == 1 {
    _blockH[_blockNum-1] = _blockH[_blockNum-2]+1
    if _blockH[_blockNum-1] > 4 {_blockH[_blockNum-1] = 4}
}
//1/6の確率で1段下
else if idx == 2 {
    _blockH[_blockNum-1] = _blockH[_blockNum-2]-1
    if _blockH[_blockNum-1] < 1 {_blockH[_blockNum-1] = 1}
}
}

// ジャンプのコントロール ④
if _jumpAble {
    if _touchDown {
        _jumpAble = false
        _jumpPow = 14
    }
} else {
    if !_touchDown {
        _jumpPow = -10
    }
}
}

// ゲームオーバー時の処理
else if _scene == S_GAMEOVER {
    if _playerY < 700 {
        _playerY+=16
    }
}
}

// 背景の描画
_g.drawImage(_images[0], 0, 0, SCREEN.width, SCREEN.height)

// プレイヤーの描画
var idx = 3
if _jumpAble {idx = 2+_score%2}
_g.drawImage(_images[idx], 75, _playerY-50)

// ブロックの描画
for var i = 0; i < _blockNum; i++ {
    _g.drawImage(_images[1],

```

```

        _blockDX+CGFloat(i*50), SCREEN.height-_blockH[i]*50)
    }

    // スコアの描画
    _g.setColor(1, 1, 1)
    _g.setFont(UISystemFont(ofSize:24))
    _g.drawString("SCORE \(_score)", 10, 10)

    // メッセージの描画
    if _message != nil {
        _g.drawString(_message!,
            (SCREEN.width-_g.getStringSize(_message!).width)/2, 50)
    }

}

//=====
// イベント
//=====

// タッチ開始時に呼ばれる
override func touchesBegan(touches: NSSet, withEvent event: UIEvent) {
    if _scene == S_TITLE {
        _init = S_PLAY
    } else if _scene == S_PLAY {
        _touchDown = true
    } else if _scene == S_GAMEOVER {
        if _playerY >= SCREEN.height+50 {
            _init = S_TITLE
        }
    }
}

// タッチ終了時に呼ばれる
override func touchesEnded(touches: NSSet, withEvent event: UIEvent) {
    _touchDown = false
}

// タッチキャンセル時に呼ばれる
override func touchesCancelled(touches: NSSet, withEvent event: UIEvent) {
    _touchDown = false
}

//=====
// ユーティリティ
//=====

// 乱数の取得
func rand(num: UInt32) -> Int {
    return Int(arc4random()%num)
}
}

```



## ◆ Graphicsクラス

Graphics クラスは、よく使うグラフィックス描画の命令を簡単に利用できるようにまとめたクラスです。

### リスト 9-2-3 Graphics.swift

```
import UIKit

// グラフィックス 5
class Graphics {
    var _context: CGContextRef? // コンテキスト
    var _attrs: [NSString: NSObject] = [ // 属性
        NSFontAttributeName: UIFont.systemFontOfSize(12),
        NSForegroundColorAttributeName: UIColor.blackColor()]

    // 初期化
    init() {
    }

    //=====
    // アクセス
    //=====
    // コンテキストの指定
    func setContext(context: CGContextRef) {
        _context = context;
    }

    // 色の指定
    func setColor(r: CGFloat, _ g: CGFloat, _ b: CGFloat) {
        CGContextSetRGBFillColor(_context, r, g, b, 1.0)
        CGContextSetRGBStrokeColor(_context, r, g, b, 1.0)
        _attrs[NSForegroundColorAttributeName] =
            UIColor(red: r, green: g, blue: b, alpha: 1.0)
    }

    // フォントの指定
    func setFont(font: UIFont) {
        _attrs[NSFontAttributeName] = font
    }

    // ライン幅の指定
    func setLineWidth(lineWidth: CGFloat) {
        CGContextSetLineWidth(_context, lineWidth)
    }

    // スケールの指定
    func setScale(scaleX: CGFloat, _ scaleY: CGFloat) {
        CGContextScaleCTM(_context, scaleX, scaleY)
    }
}
```

```

// 文字列サイズの取得
func getStringSize(str: String) -> CGSize {
    let nsstr = str as NSString
    return nsstr.sizeWithAttributes(_attrs)
}

//=====
// 描画
//=====
// ラインの描画
func drawLine(x0: CGFloat, _ y0: CGFloat, _ x1: CGFloat, _ y1: CGFloat) {
    CGContextSetLineCap(_context, kCGLineCapRound)
    CGContextMoveToPoint(_context, x0, y0)
    CGContextAddLineToPoint(_context, x1, y1)
    CGContextStrokePath(_context)
}

// 矩形の描画
func drawRect(x: CGFloat, _ y: CGFloat, _ w: CGFloat, _ h:CGFloat) {
    CGContextMoveToPoint(_context, x, y)
    CGContextAddLineToPoint(_context, x+w, y)
    CGContextAddLineToPoint(_context, x+w, y+h)
    CGContextAddLineToPoint(_context, x, y+h)
    CGContextAddLineToPoint(_context, x, y)
    CGContextAddLineToPoint(_context, x+w, y)
    CGContextStrokePath(_context)
}

// 矩形の塗り潰し
func fillRect(x: CGFloat, _ y: CGFloat, _ w: CGFloat, _ h: CGFloat) {
    CGContextFillRect(_context, CGRectMake(x, y, w, h))
}

// イメージの描画
func drawImage(image: UIImage, _ x: CGFloat, _ y: CGFloat) {
    image.drawAtPoint(CGPointMake(x, y))
}

// イメージの描画
func drawImage(image: UIImage, _ x: CGFloat, _ y: CGFloat,
    _ w: CGFloat, _ h: CGFloat) {
    image.drawInRect(CGRectMake(x, y, w, h))
}

// 文字列の描画
func drawString(str: String, _ x: CGFloat, _ y: CGFloat) {
    str.drawAtPoint(CGPointMake(x, y), withAttributes:_attrs)
}
}

```

## 9-2-4 ソースコードの解説

### ① シーンの遷移

このゲームには次の3つのシーンがあります。現在のシーンは `_scene` 変数で保持しています。次に遷移するシーンは `_init` 変数で保持しています。遷移しない時は `_init` は -1 を保持しています。

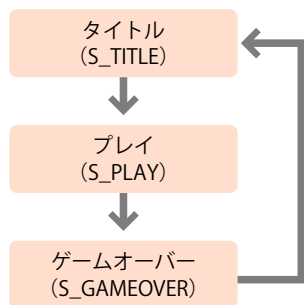


図 9-2-7 シーンの遷移

- **タイトル(S\_TITLE)**: "Touch Start"という文字列を表示するシーンです。画面をタッチすると、シーンは「プレイ」に遷移します。
- **プレイ(S\_PLAY)**: 実際にゲームをプレイするシーンです。キャラクターが落下したら、シーンは「ゲームオーバー」に遷移します。
- **ゲームオーバー(S\_GAMEOVER)**: "Game Over"という文字列を表示するシーンです。画面をタッチすると、シーンは「タイトル」に遷移します。

### ② ブロックのスクロール

スクロールする地面は、ブロックを横に並べて、画面の右から左へ少しずつ動かすことによって実現しています。ブロックの表示位置が画面より下になると、ブロックは見えなくなり落とし穴に見えます。ブロックは画面更新のたびに10ポイントずつ左に移動していき、一番左のブロックが画面から消えたとき、すべてのブロックの横方向の位置を戻し、高さを右隣のブロックと同じにします。このトリックで、無限に続く地面に見せかけています。

`_blockDX` 変数には一番左のブロックの X 座標、`_blockH` 配列には全ブロックの高さを保持しています。

```
// ブロックのスクロール ②
_blockDX -= 10
if _blockDX == -50 {
    _blockDX = 0
```

```
// 横方向の位置を戻し、高さを右隣のブロックと同じに
```

```

        for var i = 0; i < _blockNum-1; i++ {
            _blockH[i] = _blockH[i+1]
        }

        ~省略~
    }

```

### ③ 新規ブロックの高さの設定

すべてのブロックの横方向の位置を戻すとき、一番右に新規ブロックの高さを設定します。地面の高さは、1つ前が穴のときは2つ前の高さにします。そうでないときは1/6の確率で穴、1/6の確率で1段上、1/6の確率で1段下、3/6の確率で同じ高さとしします。

```

// 新規ブロックの高さの設定 ③
var idx = rand(6)
// 1つ前が穴のときは2つ前の高さ
if _blockH[_blockNum-2] == 0 {
    _blockH[_blockNum-1] = _blockH[_blockNum-3]
}
// 1/6の確率で穴
else if idx == 0 {
    _blockH[_blockNum-1] = 0
}
// 1/6の確率で1段上
else if idx == 1 {
    _blockH[_blockNum-1] = _blockH[_blockNum-2]+1
    if _blockH[_blockNum-1] > 4 {_blockH[_blockNum-1] = 4}
}
// 1/6の確率で1段下
else if idx == 2 {
    _blockH[_blockNum-1] = _blockH[_blockNum-2]-1
    if _blockH[_blockNum-1] < 1 {_blockH[_blockNum-1] = 1}
}

```

### ④ ジャンプのコントロール

キャラクターのジャンプは、\_jumpAble 変数と \_jumpPow 変数でコントロールします。\_jumpAble は、ジャンプができる状態かどうかを表します。キャラクターのすぐ下に地面があるときは true、ないときは false になります。\_jumpPow はジャンプ力を表します。タッチされたときに \_jumpAble が true なら、\_jumpPow 変数に14をチャージします。\_jumpPow の値は画面更新のたびに1ずつ減りますが、\_jumpPow が0以上のあいだはキャラクターが上昇し、0以下になると下降し始めます。キャラクターの足の下が地面のとき0が代入されます。ユーザーがタッチを離れたときには、\_jumpPow に0が代入され、ただちに下降し始めます。

```

    if !_jumpAble {
        if _touchDown {
            _jumpAble = false
            _jumpPow = 14
        }
    } else {
        if !_touchDown {
            _jumpPow = -10
        }
    }
}

```

## 5 Graphicsクラスの利用

よく使うグラフィックス描画の命令を簡単に利用できるようにまとめた「Graphics クラス」を用意しました。

「Graphics クラス」の持つアクセス関連のメソッドは次の通りです。

Graphicsクラスのメソッド	解説
func setContext(context: CGContextRef)	コンテキストの指定
func setColor(r: CGFloat, g: CGFloat, b: CGFloat)	色の指定
func setFont(font: UIFont)	フォントの指定
func setLineWidth(lineWidth: CGFloat)	ライン幅の指定
func setScale(scaleX: CGFloat, scaleY: CGFloat)	スケールの指定
func getStringSize(str: String) -> CGSize	文字サイズの取得

表 9-2-1 Graphicsクラスの持つアクセス関連のメソッド

「Graphics クラス」の持つ描画関連のメソッドは次の通りです。

Graphicsクラスのメソッド	解説
func drawLine(x0: CGFloat, y0: CGFloat, x1: CGFloat, y1: CGFloat)	ラインの描画
func drawRect(x: CGFloat, y: CGFloat, w: CGFloat, h: CGFloat)	矩形の描画
func fillRect(x: CGFloat, y: CGFloat, w: CGFloat, h: CGFloat)	矩形の塗り潰し
func drawImage(image: UIImage, x: CGFloat, y: CGFloat)	イメージの描画
func drawImage(image: UIImage, x: CGFloat, y: CGFloat, w: CGFloat, h: CGFloat)	イメージの描画
func drawString(str: String, x: CGFloat, y: CGFloat)	文字列の描画

表 9-2-2 Graphicsクラスの持つ描画関連のメソッド

## 9-2-5 画面の向きとアプリ名とアイコンの設定

### ◆ 画面の向き

サポートする画面の向きは、プロジェクトファイルの「General」の「Deployment Info」の「Device Orientation」で設定します。今回は横画面のゲームなので、「Landscape Left」と「Landscape Right」をチェックします。

### ◆ ステータスバーの非表示

ステータスバーを非表示にするには、プロジェクトファイルの「General」の「Deployment Info」の「Status Bar Style」で「Hide status bar」にチェックを入れます。

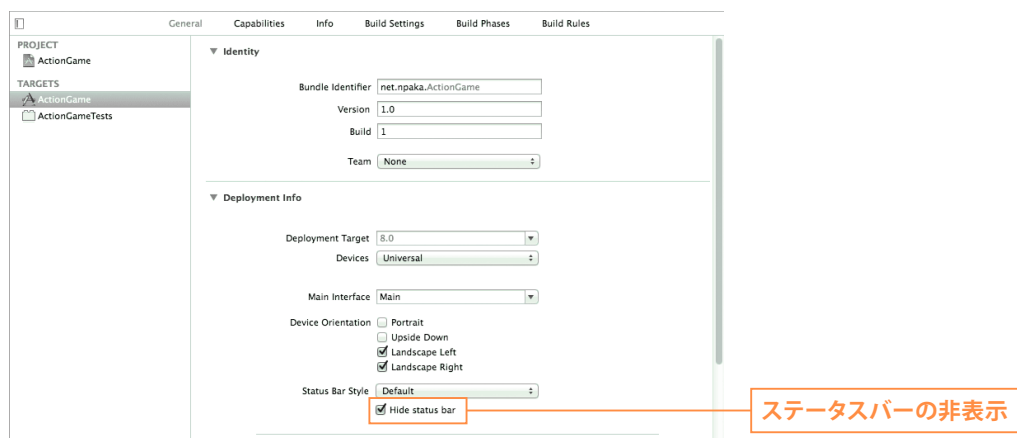


図 9-2-8 ステータスバーの非表示

### ◆ アプリ名

アプリ名は、プロジェクトファイルの「Info」の「Custom iOS Target Properties」の「Bundle name」で設定します。今回は「ジャンプランナー」と設定しました。

### ◆ アイコン

「アプリアイコン」を設定するには、「プロジェクトナビゲータ」の「Images.xcassets」の「AppIcon」を選択します。今回は、次のアプリアイコンを設定しました。



図 9-2-9 アプリアイコンの設定

### ◆ 起動画面

起動画面を設定するには、「プロジェクトナビゲータ」の「LaunchScreen.xib」を選択して編集します。  
今回はラベルに「ジャンプランナー」を指定しました。

### ◆ GameViewControllerクラス

GameViewController クラスは、SpriteKit のビューの生成とシーンの追加を行うクラスです。

リスト 9-3-1 GameViewController.swift

```
import UIKit
import SpriteKit

// ゲームビューコントローラ
class GameViewController: UIViewController {

    // ロード完了時に呼ばれる
    override func viewDidLoad() {
        super.viewDidLoad()

        // スプライトキットのビューの設定
        let skView = self.view as SKView
        skView.showsFPS = false //FPS の表示
        skView.showsNodeCount = false // ノード数の表示

        // シーンの追加 ①
        let scene = GameScene(size: CGSizeMake(360, 640))
        scene.scaleMode = SKSceneScaleMode.AspectFill
        skView.presentScene(scene)
    }
}
```

### ◆ GameSceneクラス

GameScene クラスは、プログラムの本体となるクラスです。

リスト 9-3-2 GameScene.swift

```
import SpriteKit
import UIKit

// ゲームシーン
class GameScene: SKScene {

    // シーン定数 ②
    let SCENE_TITLE = 0 // タイトル
    let SCENE_PLAY = 1 // プレイ
    let SCENE_GAMEOVER = 2 // ゲームオーバー
```



```

var _yuY = 0    //Y 座標
var _yuLV = 0   // レベル
var _yuHP = 0   // 体力
var _yuEXP = 0  // 経験値

// 敵パラメータ ④
var _enType = 0 // 種類
var _enHP = 0   // 体力

//=====
//UI
//=====
// 初期化
required init(coder: NSCoder) {
    super.init(coder: coder)

    // 画面サイズの調整
    if SCREEN.width >= 768 {
        SCALE = 2
        SCREEN.width /= 2
        SCREEN.height /= 2
    }

    // イメージの取得
    for var i = 0; i < 7; i++ {
        _images.append(UIImage(named: "rpg\(i).png")!)
    }

    // タイマーの開始
    NSTimer.scheduledTimerWithTimeInterval(0.1,
        target: self, selector: "onTick:", userInfo: nil, repeats: true)
}

// 定期処理
func onTick(timer: NSTimer) {
    self.setNeedsDisplay()
}

//=====
// 描画
//=====
// ステータスの描画
func drawStatus() {
    _g.setColor(1, 1, 1)
    _g.fillRect((SCREEN.width-504)/2, 8, 504, 54)
    if _yuHP != 0 {
        _g.setColor(0, 0, 0)
    } else {
        _g.setColor(1, 0, 0)
    }
}

```

```

// アイコンの指定
let imgIcon = cell!.contentView.viewWithTag(1) as UIImageView
imgIcon.image = status.icon

// 名前の指定
let lblName = cell!.contentView.viewWithTag(2) as UILabel
lblName.text = status.name

// テキストの追加
let lblText = cell!.contentView.viewWithTag(3) as UILabel
lblText.text = status.text
lblText.frame = CGRectMake(60, 30, SCREEN.width-70, 1024)
lblText.sizeToFit()
return cell!
}

//=====
//Twitter
//=====
//Twitter のアカウント情報の取得 ①
func initTwitterAccount() {
    _account = nil
    _accountStore = ACAccountStore()
    let twitterType = _accountStore?
        .accountTypeWithAccountTypeIdentifier(ACAccountTypeIdentifierTwitter)
    _accountStore?.requestAccessToAccountsWithType(
        twitterType, options:nil) {(granted, error) in
        if granted {
            let accounts = self._accountStore?
                .accountsWithAccountType(twitterType)
            if accounts!.count > 0 {
                self._account = accounts![0] as? ACAccount
                self.timeline()
                return
            }
        }
        dispatch_async(dispatch_get_main_queue(), {
            self.showAlert(nil, text:"Twitter アカウントが登録されていません ")
        })
    })
}

// アイコンの読み込み
func loadIcon(status: Status) {
    // 通信によるアイコン読み込み
    dispatch_async(dispatch_get_global_queue(
        DISPATCH_QUEUE_PRIORITY_DEFAULT,0)) {
        let request = NSURLRequest(URL: NSURL(string: status.iconURL)!,
            cachePolicy: NSURLRequestCachePolicy.UseProtocolCachePolicy,

```

```

        timeoutInterval: 30.0)
let data = NSURLConnection.sendSynchronousRequest(request,
    returningResponse:nil, error:nil)

// テーブル更新
dispatch_async(dispatch_get_main_queue()) {
    status.icon = UIImage(data: data!)
    self._tableView?.reloadData()
}
}

// タイムラインの取得
func timeline() {
    // インジケータ表示
    self.setIndicator(true)

    // タイムラインの読み込み ②
    let url = NSURL(string:
        "https://api.twitter.com/1.1/statuses/home_timeline.json")!
    let params: Dictionary<NSObject, AnyObject!> = ["count": "20"]
    let timeline = SLRequest(forServiceType: SLServiceTypeTwitter,
        requestMethod: SLRequestMethod.GET, URL: url, parameters: params)
    timeline.account = self._account
    timeline.performRequestWithHandler(){(responseData: NSData!,
        urlResponse: NSHTTPURLResponse!, error: NSError!) in
        //JSON のパース ③
        var jsonError: NSError? = nil
        let obj : AnyObject? = NSJSONSerialization.JSONObjectWithData(
            responseData, options: nil, error: &jsonError)

        // エラー表示
        if error != nil {
            self.showAlert(nil, text: "通信失敗しました ")
        } else if obj == nil || jsonError != nil {
            self.showAlert(nil, text: "パースに失敗しました ")
        } else if !obj!.isKindOfClass(NSArray) {
            self.showAlert(nil, text: "パースに失敗しました ")
        } else {
            //JSON をパースしたデータを Status クラスの配列に変換 ④
            self._items.removeAll()
            let statuses = obj as NSArray
            for var i = 0; i < statuses.count; i++ {
                let item = Status()
                let status = statuses[i] as NSDictionary
                let user = status["user"] as NSDictionary
                item.text = status["text"] as String
                item.name = user["screen_name"] as String
                item.iconURL = user["profile_image_url"] as String
                self.loadIcon(item)
            }
        }
    }
}

```

```

        self._items.append(item)
    }
}

// テーブル更新
dispatch_async(dispatch_get_main_queue(), {
    let tableView = self._tableView
    tableView?.reloadData()
})

// インジケータ非表示
self.setIndicator(false)
}
}

// つぶやきの送信 5
func twit() {
    let twitterCtl = SLComposeViewController(
        forServiceType: SLServiceTypeTwitter)
    self.presentViewController(twitterCtl, animated: true, completion: nil)
}

//=====
// ユーティリティ
//=====
// バイト配列を文字列に変換
func data2str(data: NSData) -> NSString {
    return NSString(data: data, encoding: NSUTF8StringEncoding)!
}
}

```

## ◆ Statusクラス

Status クラスは、Twitter の1ツイート分の状態情報を保持するクラスです。

### リスト 9-5-2 Status.swift

```

import UIKit

// ステータス
class Status {
    var name = ""
    var text = ""
    var iconURL = ""
    var icon: UIImage?
}

```