

クソゲーを作って学ぶ

Swift勉強会

<https://github.com/MasayaHayashi724/kuso-game-hands-on/blob/master/kuso-game-hands-on.pdf>

今日の目的

- ✗ Swiftができるようになる
- ✗ ゲームが作れるようになる
- ✗ クソゲーが作れるようになる
- ○ クソみたいなゲームが作れるようになる
- ○ アプリ開発の楽しさを知る
- ○ 何か他にもアプリを作ってみたくなる

お品書き

1. Swiftの簡単な文法解説 (30分)
 - 文法
 - ゲーム開発に使えるTips
2. ハンズオンでゲーム作ってみる (3時間)
3. オリジナルのゲーム作ってみる (6~7時間?)
(ハンズオンで作ったゲームを発展させてもOK)

Swiftの簡単な文法解説

文法

変数定義

```
let height: Double = 25.5 // letは定数、再代入できない
var score: Int = 0 // varは変数、再代入できる
let nameArray: [String] = ["masaya", "hayashi"] // 配列
var array = [1, 2, 3] // 型が明確なら省略できる
var didWin: Bool = false // Bool型(true, false)
let dict: [String: Int] = ["masaya": 6, "hayashi": 7]
// 辞書型というのもある
```

オプショナル型

```
let gender: Int? = nil
```

if文

```
let score = 12
if score > 10 {
    print("You did it!")
} else {
    print("Try again!")
}
```

```
let didWin = true
if didWin {
    print("You win!")
} else {
    print("You lose...")
}
```

for文

```
for i in 0..<10 {  
    print(i) // 9まで表示される  
}
```

```
for i in 0...10 {  
    print(i) // 10まで表示される  
}
```

```
let numbers = [1, 2, 3, 4, 5]  
for number in numbers { // for i in 0..<5 {  
    print(number) // print(numbers[i])  
} // } とするより安全(動作は同じ)
```

関数

```
func finishGame() {  
    isPaused = true  
}
```

```
func add(arg1: Double, arg2: Double) -> Double {  
    return arg1 + arg2  
}
```

```
let result = add(arg1: 1.2, arg2: 3.4) // 4.6
```

```
func add(_ arg1: Double, to arg2: Double) -> Double {  
    return arg1 + arg2  
}
```

```
let result = add(1.0, to: 2.2) // 自然言語っぽくも書ける
```

guard文

```
let a: Int?  
guard let value = a else {  
    return // aがnilであればここ  
}  
print(value) // aの値をprint
```

```
let x = 25  
guard x < 30 else {  
    return // ここには来ない  
}  
print(x) // ここに来る
```

```
let x = 25  
if x < 30 {  
    print(x) // これと同じ  
}
```

Swiftの簡単な文法解説

ゲーム開発に使えるTips

Nodeの表示方法

```
class GameScene: SKScene {  
  
    var spaceship: SKSpriteNode!  
    // `!`は絶対にあとで何かを代入するから心配しないでって意味  
    // classのプロパティは初期値を持つ必要があるが!つければOK  
  
    override func didMove(to view: SKView) {  
        spaceship = SKSpriteNode(imageNamed: "spaceship")  
        spaceship.scale(to: CGSize(width: 50, height: 50))  
        spaceship.position = CGPoint(x: 150, y: 200)  
        addChild(spaceship)  
        // frame.widthとかframe.heightとかで画面の幅と高さ  
        // を参照できるので、うまく使いましょう。  
    }  
}
```

Nodeの動かし方

```
let missile = SKSpriteNode(imageNamed: "missile")
missile.position = CGPoint(x: 120, y: 100)
addChild(missile) // Nodeの追加

let move = SKAction.moveTo(y: 200, duration: 0.3)
// 移動するSKAction
let remove = SKAction.removeFromParent()
// 画面からNodeを消すSKAction
let actionArray = [move, remove]
let sequenceAction = SKAction.sequence(actionArray)
missile.run(sequenceAction)
// 2つ以上のSKActionを順番に実行してくれる
```

ランダム値

```
let max: UInt32 = 5
let randomInt = Int(arc4random_uniform(max))
// 0~4 のランダム値(整数)
```

```
let random = CGFloat(arc4random_uniform(UINT32_MAX)) /
    CGFloat(UINT32_MAX)
// 0~1 のランダム値(実数)
// CGFloatのところをDoubleとかにもできる
// これにframe.widthを掛けたりすると画面幅までのランダム位置に
```

タイマー

```
class GameScene: SKScene {
    var timer: Timer? = nil

    override func didMove(to view: SKView) {
        timer = Timer.scheduledTimer(
            timeInterval: 1.0,
            target: self,
            selector: #selector(addAsteroid),
            userInfo: nil,
            repeats: true
        )
    }

    func addAsteroid() {
        // do something
    }
}
```

端末の傾きの取得 ([ここを参考に](#))

```
class GameScene: SKScene {
    let motionManger = CMMotionManager()
    var acceleration: CGFloat = 0.0
    override func didMove(to view: SKView) {
        motionManger.accelerometerUpdateInterval = 0.2
        motionManger.startAccelerometerUpdates(to: OperationQueue.main) { [unowned self] (data, error) in
            guard let accelerometerData = data else { return }
            let acceleration = accelerometerData.acceleration
            self.acceleration = CGFloat(acceleration.x) * 10
        }
    }
    override func didSimulatePhysics() {
        let nextPositionX = spaceship.position.x + acceleration.x
        guard nextPositionX > 30 else { return }
        guard nextPositionX < frame.width - 30 else { return }
        spaceship.position.x = nextPositionX
    }
}
```

衝突処理

- `physicsBody` : 衝突を判定する範囲
- `isDynamic` : `true`にしておく
- `categoryBitMask` : このノードのIDみたいなもの
- `contactTestBitMask` : 衝突対象の `categoryBitMask` との ANDが1以上になれば、衝突する
- `collisionBitMask` : とりあえず0にしておこう
- `func didBegin(_ contact: SKPhysicsContact)` : 衝突する度に呼ばれる関数

データの保存

```
// デフォルト値の保存
UserDefaults.standard.register(defaults: ["best": 0])
UserDefaults.standard
    .register(defaults: ["tutorial": false])
UserDefaults.standard
    .register(defaults: ["name": "masaya"])

// 保存している値の更新
UserDefaults.standard.set(120, forKey: "best")
UserDefaults.standard.set(true, forKey: "tutorial")
UserDefaults.standard.set("hayashi", forKey: "name")

// 保存している値の読み出し
UserDefaults.standard.integer(forKey: "best")
UserDefaults.standard.bool(forKey: "tutorial")
UserDefaults.standard.string(forKey: "name")

// 値の削除
UserDefaults.standard.removeObject(forKey: "name")
```

ゲームの終了処理

```
class GameScene: SKScene {
    var vc: GameViewController!

    func finishGame() {
        self_vc.dismiss(animated: true, completion: nil)
    }
}
```

```
class GameViewController: UIViewController {

    ...

    scene.vc = self

    ...

}

Masa
```

Swiftの簡単な文法解説

おわり

2Dゲームを作ってみよう

Swift vs Unity

Swift (Spritekit(今日これ), Scenekit, Metal)

- 無料で簡単
- Appleが作っている

Unity

- Androidも作れる
- Asset Store

素材の探し方

良いフリー素材を探すのは結構時間がかかります。
あらかじめ探しておくと当日の開発がスムーズに。

おすすめサイト

- ぴぽや <http://blog.pipoya.net/>
- illust AC <https://www.ac-illust.com/>
- Rド <http://www.geocities.co.jp/Milano-Cat/3319/muz/002.html>
- Folce-zero <http://folce.zatunen.com>

では、作ってみましょう！

↓完成品↓

<https://github.com/MasayaHayashi724/save-the-earth>

<https://itunes.apple.com/jp/app/id1265444161>

使用する素材

1. `$ mkdir kuso-game` (お好きなところで)
2. `$ cd kuso-game`

```
$ git clone  
https://github.com/MasayaHayashi724/kuso-game-hands-on.git
```
3. `$ cd kuso-game-hands-on`
4. `$ open .`
5. `$ images`ディレクトリ内に使う素材があります

最初にすること

1. GameScene.sks のHello, World!を消す
2. GameScene.swift の20行目のSKScene を GameScene に
3. GameScene.swift の class GameScene の下にある
private ついてるやつ2つ消す
4. GameScene.swift の func didMove(to ...) と
func touchesEnded(...) の {} の中身を全部消す
5. func didMove(to ...) と func touchesEnded(...) 以外の
func を全部消す
6. さっそく clone した kuso-game-hands-on の images の中の
画像を Assets.xcassets にドラッグ&ドロップ

シューティングゲームを作る

1. プロジェクトの作成
2. メニュー画面とゲーム画面の作成
3. 宇宙船を追加 (傾けると移動、タップでミサイル)
4. 小惑星を追加 (ランダム位置に出現して迫ってくる)
5. 衝突処理 (ミサイル-小惑星、小惑星-宇宙船or地球)
6. スコアとライフを導入
7. ゲーム終了処理
8. ベストスコアの保存
9. 小惑星のスピードを上げる

1. プロジェクトの作成

- Xcode 開く -> Create a new Xcode project -> Game を選択して Next
- Product Name は適当に決める
- Organization Name は自分の名前とかに
- Organization Identifier は com.[Organization Name] とかに
- Swift、SpriteKit、Universal、チェックは3つとも外す
- 好きな場所に Create

2. メニュー画面とゲーム画面の作成

- `Main.storyboard` に `UIViewController` をドロップ
- 矢印をそっちに移動
- `UIButton` をドロップする
- `Ctrl` 押しながらボタンから Game 画面へドラッグ & ドロップして `Present Modally` 選択
- `Cmd + R` でビルドして Run してみる(左上の再生ボタンでもOK)
- ボタン押して画面遷移できるか確認

3. 宇宙船を追加 (傾けると移動、タップでミサイル)

- spaceship を画面に追加 (earthには気をつけて)
 - ゲームが始まると `didMove(to: ...)` が呼ばれる
 - Xcodeは賢いのでちょっと打ったら候補を出してくれます
 - spaceship の前には `self` をつけると補完がうまく働きます
- タップでミサイルを発射する
 - `touchedEnded` というメソッドがタップした指を離すと自動的に呼ばれます
- 傾きによって spaceship を動かす

実機でビルドする方法

- <http://qiita.com/DKN915/items/7a2ce97f3758e2daf486>
- 上の記事を参考にしてね。
- 初めてやるときはちょっと時間かかるので、PCにスマホを接続しながら他の作業をするとgood

4. 小惑星を追加 (ランダム位置に出現して迫ってくる)

- 1秒ごとに小惑星を追加 (Timerを利用)
- 3種類からランダムで選択
- ランダムな位置に追加
- 衝突判定を追加
- 画面下方向に動かす
- <https://github.com/MasayaHayashi724/save-the-earth/pull/3/commits/cbfe4dded7516d488681083809b64f8355f5f23e>

5. 衝突処理(ミサイル-小惑星、小惑星-宇宙船or地球)

- `File` -> `New` -> `File` -> `SpriteKit Particle File` -> `Fire`を選択 -> `Explosion.sks`とかの名前で保存
 - 小惑星の爆発に使うエフェクトです
- 衝突判定を実装
 - 衝突する度に `didSimulatePhysics` が自動的に呼ばれる
 - `contact` は `bodyA` と `bodyB` を持っていますが、どっちが小惑星でどっちがミサイルか、などは僕達が判断してあげる必要があります
 - `Explosion.sks` のファイル名を間違えないように

6. スコアとライフを導入

- ライフが減っていくシステムを導入
 - `didSimulatePhysics`内で処理すればOK
- スコアを表示するラベルを追加
 - `SKSpriteNode`の仲間みたいな `SKLabelNode` を使います
 - 追加方法は `SKSpriteNode` と同じ
- 小惑星を破壊するとスコアを増加させる
 - これも `didSimulatePhysics` 内で処理する

7. ゲーム終了処理

- ライフがなくなったらゲームを一時停止する
 - `isPaused` というプロパティに `true` を代入するだけでゲームを一時停止することができます
- 一時停止してから一定時間でメニュー画面に戻る
 - 遷移先のビューに値を渡す処理
 - 少しトリッキーな処理ですが、できるようになっておくと便利です
 - リザルト画面を表示するときとかにも使えます

8. ベストスコアの保存

- ベストスコアをメニュー画面に表示
 - Storyboard 内のメニュー画面の Custom Class を MenuViewController.swift (Super Class を UIViewController として新規作成する) に設定
 - SKLabelNode に似た UILabel をメニュー画面に設置
 - Assistant Editor (右上の○2つのボタン) を表示して UILabel を Outlet 接続 (Ctrl 押しながら Assistant Editor にドラッグ&ドロップ)
 - 詳しくは林に聞いてください
- ベストスコアが更新されたらそれを保存する

9. 難易度を上げる

- 小惑星のスピードを時間とともに上げていく
- これ以外にもいろいろ難易度を上げていく手段があるのでいろいろやってみましょう

お疲れさまでした！

とりあえずシューティングゲームができました！

好きなゲームを作つてみよう！

or

さっきのゲームを進化させよう！