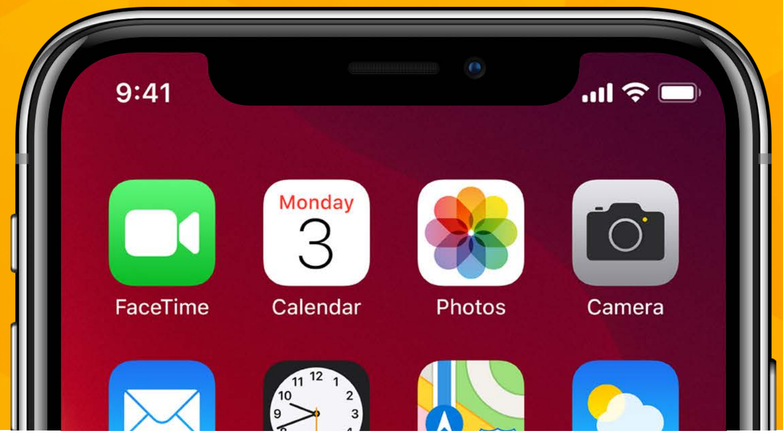


iOS Application Development

Lecture 8: Clousures, Improving Complex Input Screens, and Animations

Simon Völker & Philipp Wacker
Media Computing Group
RWTH Aachen University

hci.rwth-aachen.de/ios

Seminar Review Meetings

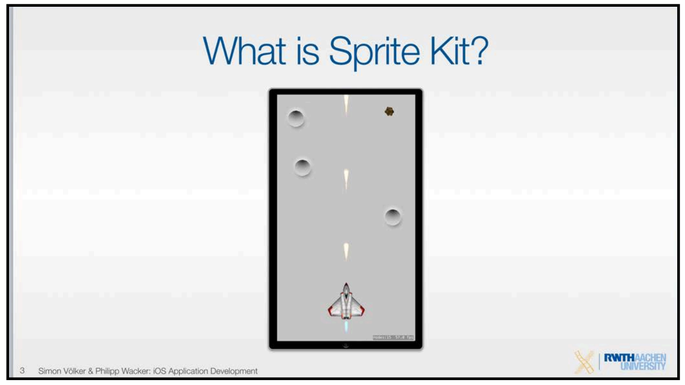
- First Seminar meetings are next week:

Topic	Group	Presentation	Discussions
MapKit, CoreLocation	**Group 8**	**18.11.19, first group**	**11.11.19**
Debugging in Xcode and Instruments	**Group 4**	**18.11.19, second group**	**11.11.19**
SiriKit	**Group 2**	**19.11.19, second group**	**12.11.19**
Extensions & Inter-App communication	**Group 5**	**19.11.19, first group**	**12.11.19**
WatchOS	Group 14	25.11.19, second group	18.11.19
Swift UI	Group 11	25.11.19, first group	18.11.19
Scene Kit	Group 7	26.11.19, first group	19.11.19
GameplayKit	Group 9	26.11.19, second group	19.11.19
RealityKit & Reality Composer	Group 1	09.12.19, first group	02.12.19
Networking on iOS	Group 6	2.12.19, first group	25.11.19
ClassKit	Group 3	2.12.19, second group	25.11.19
Bringing People into AR	Group 13	09.12.19, second group	02.12.19
Core Data	Group 10	3.12.19, first group	26.11.19
Core ML + Create ML	Group 12	3.12.19, second group	26.11.19

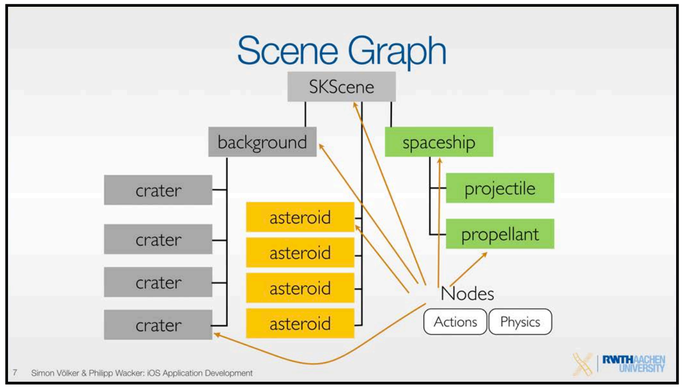
Seminar

- 20 minute presentations + 10 min Q&A
- Deliverables: Slides + well documented demo code

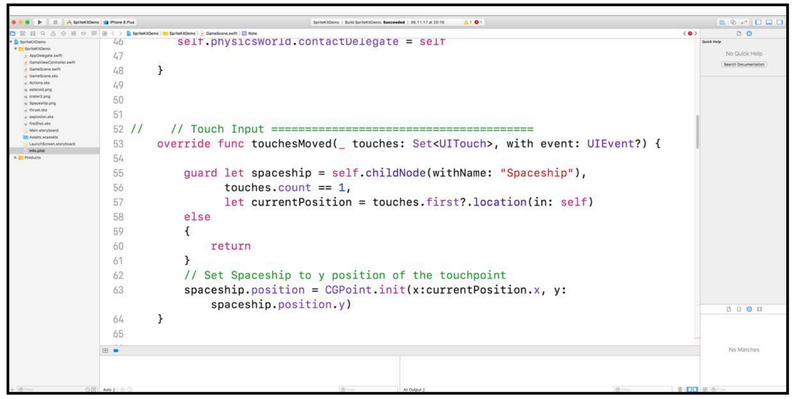
Introduction



Structure



Demo



Closures

Closures

- Similar to Objective-C Blocks and Java Lambdas
- Self-contained blocks of functionality
  - „Anonymous functions“

```
{ (parameters) -> return type in
statements
}
```

Closure Syntax

- Function:

```
func sumFunction(numbers: [Int]) -> Int {
    var total = 0
    //Code
    return total
}

let sum = sumFunction(numbers: [5,34,12,42])
```

- Closure:

```
let sumClosure = { (numbers: [Int]) -> Int in
    var total = 0
    //Code
    return total
}

let sum = sumClosure([8,34,16,42])
```

Closure Types

- No parameters, no return value:

```
let printClosure1 = { () -> Void in
    print("This closure does not take any parameters and does not return a value.")
}
```

- With parameters, no return value:

```
let printClosure2 = { (string: String) -> Void in
    print(string)
}
```

- No parameters, with return value:

```
let randomNumberClosure1 = { () -> Int in
    return 0
}
```

- With parameters, with return value:

```
let randomNumberClosure2 = { (minValue: Int, maxValue: Int) -> Int in
    return 0
}
```

Trailing Closure

Closure as the only argument:

```
let sortedTracks = tracks.sorted(by: { (firstTrack, secondTrack) -> Bool in
    return firstTrack.starRating < secondTrack.starRating
})
```

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) -> Bool in
    return firstTrack.starRating < secondTrack.starRating
}
```

Closure as the last argument:

```
func performRequest(url: String, response: (Int) -> Void)
{ }

performRequest(url: "https://www.apple.com") { (data) in
    print(data)
}
```

Simplifying Closures

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) -> Bool in
    return firstTrack.starRating < secondTrack.starRating
}
```

Infer the return type:

```
let sortedTracks = tracks.sorted { (firstTrack, secondTrack) in
    return firstTrack.starRating < secondTrack.starRating
}
```

Use placeholder arguments:

```
let sortedTracks = tracks.sorted {
    return $0.starRating < $1.starRating
}
```

Automatic return

```
let sortedTracks = tracks.sorted {$0.starRating < $1.starRating}
```

Mapping

```
let names = ["Johnny", "Nellie", "Aaron", "Rachel"]
var fullNames: [String] = []

for name in names {
    let fullName = name + " Smith"
    fullNames.append(fullName)
}
```

Use the map() function:

```
let fullNames = names.map { (name) -> String in
    return name + " Smith"
}
```

Short version:

```
let fullNames = names.map { $0 + " Smith" }
```

Filter

```
let numbers = [4, 8, 15, 16, 23, 42]
var numbersLessThan20: [Int] = []

for number in numbers {
    if number < 20 {
        numbersLessThan20.append(number)
    }
}
```

Use the filter() function:

```
let numbersLessThan20 = numbers.filter { (number) -> Bool in
    return number < 20
}
```

Short version:

```
let numbersLessThan20 = numbers.filter { $0 < 20 }
```

Reduce

```
let numbers = [8, 6, 7, 5, 3, 0, 9]
var total = 0

for number in numbers {
    total = total + number
}
```

Use the reduce() function:

```
let total = numbers.reduce(0) { (currentTotal, newValue) -> Int in
    return currentTotal + newValue
}
```

Short version:

```
let total = numbers.reduce(0, { $0 + $1})
```

Capturing Values

```
func makeIncrementer(forIncrement amount: Int) -> () -> Int {
    var runningTotal = 0
    func incrementer() -> Int {
        runningTotal += amount
        return runningTotal
    }
    return incrementer
}
```

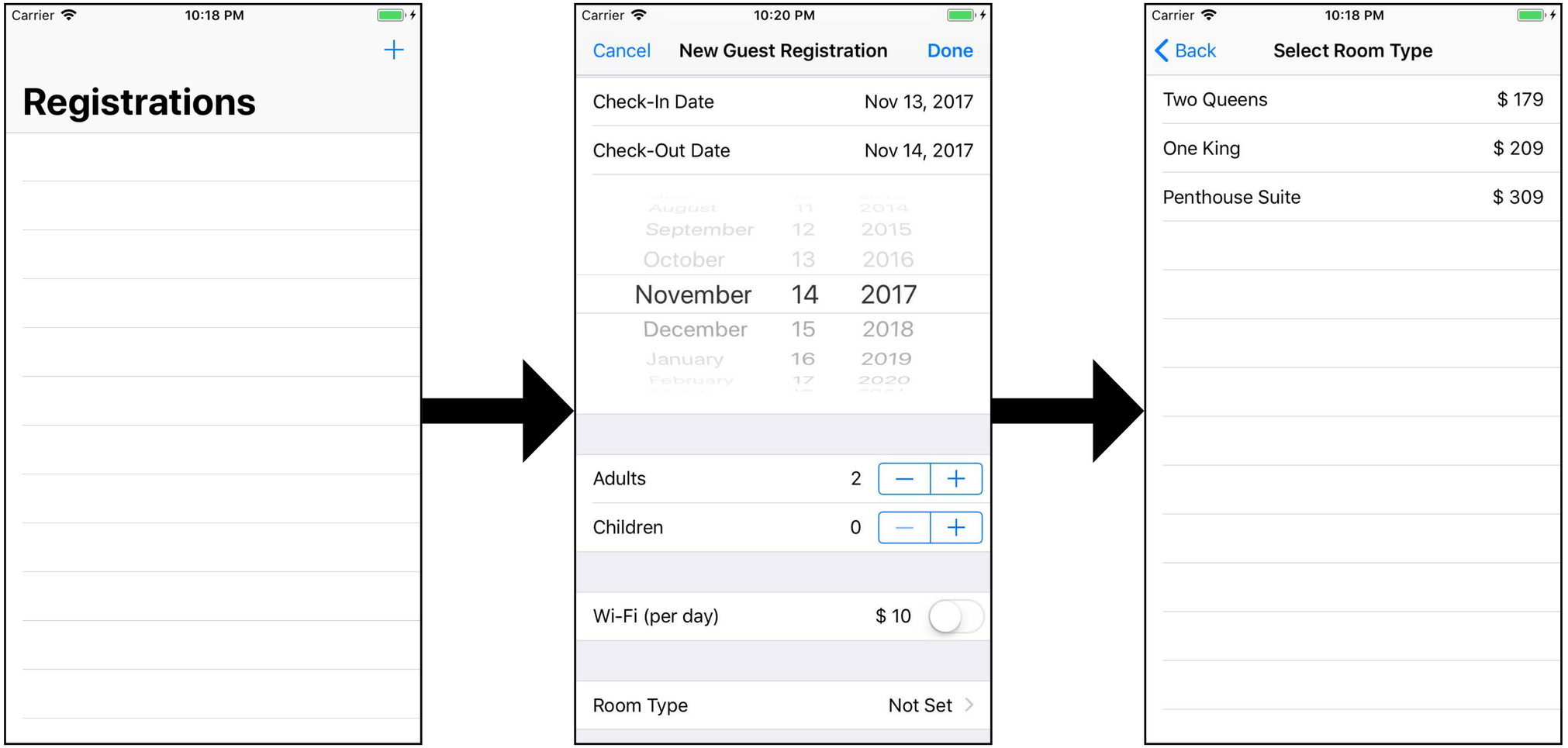
```
let incrementByTen = makeIncrementer(forIncrement: 10)

incrementByTen() //10
incrementByTen() //20
incrementByTen() //30
```

```
let incrementBySeven = makeIncrementer(forIncrement: 7)
incrementBySeven()  //7
incrementByTen()    //40
let alsoIncrementBy10 = incrementByTen
alsoIncrementBy10() //50
```

Improving Complex Input Screens

Improving Complex Input Screens



Animation

Why Animations?

- Direct the User's Attention
- Keep the User Oriented
- Connect User Behaviors

What Can Be Animated?

- UIView:
  - frame
  - bounds
  - center
  - transform
  - alpha

UIView Animations

Animation Closures:

```
animate(withDuration:animations:)
animate(withDuration:animations:completion:)
animate(withDuration:delay:options:animations:completion:)
```

Animate with Duration:

```
UIView.animate(withDuration: 2.0) {
    aView.alpha = 0.3
}
```

Animation Playground