

App-Entwicklung für iOS und OS X

SS 2016
Stephan Gimbel



Core Motion

- API zum Zugriff auf Motion Sensor Hardware des Devices
- Primäre Inputs: Accelerometer, Gyro, Magnetometer
Nicht alle Devices haben diese Inputs.
- Klasse welche diese Inputs verwendet ist **CMMotionManager**
Nur eine Instanz pro App verwenden (Performance).
Ist eine "globale Resource", das erhalten über eine Klassenmethode ist also ok.
- Verwendung
 1. Prüfen welche Hardware verfügbar ist.
 2. Starten des Sampling und pollen des Motion Managers bzgl. des letzten Samples.
... oder...
 1. Prüfen welche Hardware verfügbar ist.
 2. Setzen der Update Rate mit der wir Daten von der Hardware empfangen wollen.
 3. Registrierung eines Closures (und Queue auf der es ausgeführt wird) welches für jedes Sample ausgeführt wird.

Core Motion

- Verfügbarkeit von Hardwaresensoren prüfen

```
var {accelerometer, gyro, magnetometer, deviceMotion}Available: Bool  
deviceMotion ist eine Kombination aller verfügbarer Hardware (Accelerometer,  
Magnetometer, Gyro).
```

Gleich mehr dazu.

- Start der Hardwaresensoren um Daten zu sammeln

Müssen wir nur machen, wenn wir Daten pollen.

```
func start{Accelerometer, Gyro, Magnetometer, DeviceMotion}Updates()
```

- Sammelt die Hardware aktuell Daten?

```
var {Accelerometer, Gyro, Magnetometer, DeviceMotion}Active: Bool
```

- Stoppen der Hardware

Es kostet Performance Daten zu sammeln, also stoppen wenn die Daten nicht gebraucht werden.

```
func stop{Accelerometer, Gyro, Magnetometer, DeviceMotion}Updates()
```

Core Motion

- Prüfen der Daten (Polling nicht empfohlen)

```
var accelerometerData: CMAccelerometerData  
CMAccelerometerData Objekte stellen var acceleration: CMAcceleration zur  
Verfügung.
```

```
struct CMAcceleration {  
    var x: Double // in g (9.81 m/s2)  
    var y: Double // in g  
    var z: Double // in g  
}
```

Diese Rohdaten enthalten Beschleunigung durch Schwerkraft.

Wenn die Hardware "flach" liegt, ist z also 1.0, x und y sind 0.0.

- Analog dazu...

```
var gyroData: CMGyroData  
CMGyroData Objekte stellen var rotationRate: CMRotationRate zur Verfügung.
```

```
struct CMRotationRate {  
    var x: Double // in Radian/s  
    var y: Double // in Radian/s  
    var z: Double // in Radian/s  
}
```

Vorzeichen folgt Rechter-Hand Regel.

Die Daten haben eine Verzerrung.

Core Motion

- und weiterhin...

`var magnetometerData: CMMagnetometerData`
CMMagnetometerData Objekte stellen `var magneticField: CMMagneticField` zur
Verfügung.

```
struct CMMagneticField {  
    var x: Double // in Microtesla  
    var y: Double // in Microtesla  
    var z: Double // in Microtesla  
}
```

Die Daten haben eine Verzerrung.

CMDeviceMotion

- Beschleunigungsdaten in CMDeviceMotion

```
var gravity: CMAcceleration  
var userAcceleration: CMAcceleration // Schwerkrafteinfluss entfernt durch Gyro
```

- Rotationsdaten in CMDeviceMotion

```
var rotationRate: CMRotationRate // Bias entfernt mittels Accelerometer  
var attitude: CMAttitude // Device Attitude (Orientierung) im 3D Raum  
class CMAttitude: NSObject { // Roll, Pitch und Yaw in Radian  
    var roll: Double // um Längsache (longitudinal) durch Top/Bottom  
    var pitch: Double // um Querachse (lateral) durch Seiten  
    var yaw: Double // um Achse mit Ursprung in CofG und ⊥ zum Screen nach unten  
                    // gerichtet  
}
```

Es existieren weitere mathematische Repräsentationsmöglichkeiten (z.B. Quaternionen)...

CMDeviceMotion

- Magnetfelddaten in CMDeviceMotion

```
var magneticField: CMCalibratedMagneticField
struct CMCalibratedMagneticField {
    var field: CMMagneticField
    var accuracy: CMMagneticCalibrationAccuracy
}
```

accuracy kann sein...

```
CMMagneticFieldCalibrationAccuracyUncalibrated
CMMagneticFieldCalibrationAccuracyLow
CMMagneticFieldCalibrationAccuracyMedium
CMMagneticFieldCalibrationAccuracyHigh
```

Core Motion

- Registrieren eines Blocks um Accelerometer Daten zu erhalten

```
func startAccelerometerUpdatesToQueue(queue: NSOperationQueue,  
                                       withHandler: CMAccelerometerHandler)
```

typealias CMAccelerationHandler = (CMAccelerometerData?, NSError?) -> Void
queue kann eine NSOperationQueue() sein, die wir erstellen oder NSOperation.mainQueue
(oder currentQueue).

- Registrieren eines Blocks um Gyro Daten zu erhalten

```
func startGyroUpdatesToQueue(queue: NSOperationQueue,  
                           withHandler: CMGyroHandler)
```

typealias CMGyroHandler = (CMGyroData?, NSError?) -> Void

- Registrieren eines Blocks um Magnetometer Daten zu erhalten

```
func startMagnetometerUpdatesToQueue(queue: NSOperationQueue,  
                                      withHandler: CMMagnetometerHandler)
```

typealias CMMagnetometerHandler = (CMMagnetometerData?, NSError?) -> Void

Core Motion

- Registrieren eines Blocks um DeviceMotion Daten zu erhalten

```
func startDeviceMotionUpdatesToQueue(queue: NSOperationQueue,  
                                      withHandler: CMDeviceMotionHandler)  
typealias CMDeviceMotionHandler = (CMDeviceMotionData?, NSError?) -> Void  
queue kann eine NSOperationQueue() sein, die wir erstellen oder NSOperation.mainQueue  
(oder currentQueue).
```

Errors...

```
CLErrorDeviceRequiresMovement  
CLErrorTrueNorthNotAvailable  
CLErrorMotionActivityNotAvailable  
CLErrorMotionActivityNotAuthorized
```

Core Motion

- Setzen einer Updaterate bei der der Block ausgeführt wird

```
var accelerometerUpdateInterval: NSTimeInterval  
var gyroUpdateInterval: NSTimeInterval  
var magnetometerUpdateInterval: NSTimeInterval  
var deviceMotionUpdateInterval: NSTimeInterval
```

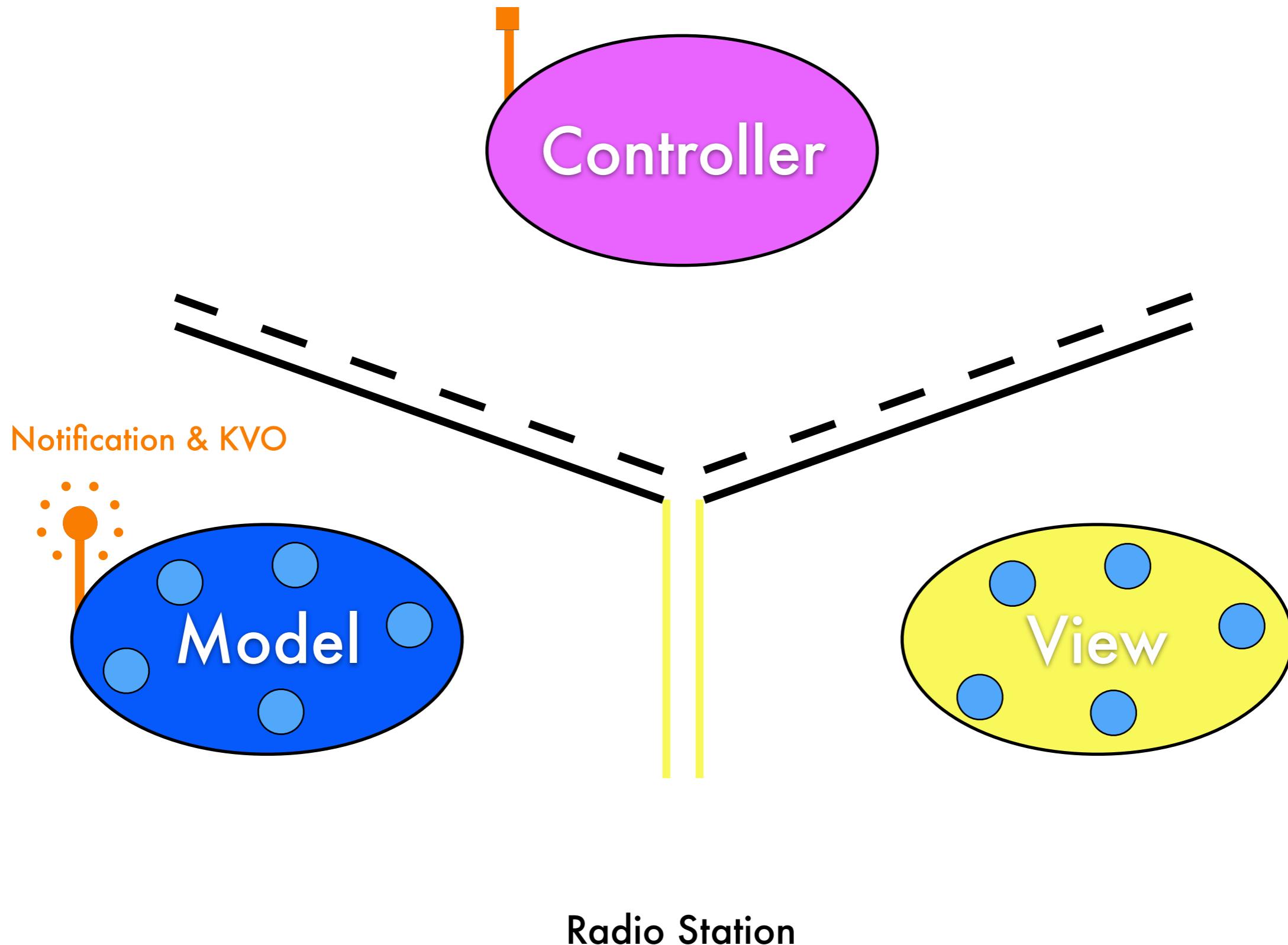
- Mehrere Handler Blocks sind auch ok

Auch wenn nur ein CMMotionManager erlaubt ist.

Jeder der Blocks empfängt die Daten zum gleichen Interval (wie oben gesetzt).

Es ist natürlich auch erlaubt mit mehreren Objekten zur gleichen Zeit zu pollen.

MVC



NSNotification

- **Notifications**

Die Radio Station aus der ersten Vorlesung (MVC).

Für Model (oder globale) zu Controller Kommunikation.

- **NSNotificationCenter**

Erhalten des Default "Notification Center" via `NSNotificationCenter.defaultCenter()`

Dann schicken der folgenden Nachricht, wenn wir einem Radiosender zuhören wollen...

```
var observer: NSObjectProtocol?          // Brauchen wir um wieder anzuhalten
observer = addObserverForName(String, // (genauer) Name des Radiosenders
                             object: AnyObject?,    // der Broadcaster/Object (nil für "alle")
                             queue: NSOperationQueue?) // Queue in der das Closure ausgeführt wird
{ (notification: NSNotification) -> Void in
    let info: [NSObject: AnyObject]? = notification.userInfo
    // info ist Dictionary von Notification-spezifischen Informationen
}
```

NSNotification

- Beispiel

Beobachten von Änderungen der Schriftgröße von bevorzugten Schriften (User kann dies in den Settings modifizieren)...

```
let center = NSNotificationCenter.defaultCenter()
var observer = addObserverForName(UIContentSizeCategoryDidChangeNotification
                                  object: UIApplication.sharedApplication(),
                                  queue: NSOperationQueue.mainQueue())
{ notification in
    // reset von Fonts von Objekten, die preferred Fonts verwenden
    // oder prüfen von Size Category und damit etwas machen
    let c = notification.userInfo?[UIContentSizeCategoryNewValueKey]
}
center.removeObserver(observer) // Wenn wir fertig sind
```



NSNotification

- **Posting einer NSNotification**

Erstellen eines NSNotification...

```
let notification = NSNotification(  
    name: String          // Name des "Radiosenders"  
    object: AnyObject?,   // wer schickt diese Notification (meist self)  
    userInfo: Dictionary // Info die an Zuhörer weitergegeben soll  
)
```

... dann die NSNotification posten ...

```
NSNotificationCenter.defaultCenter().postNotification(notification)
```

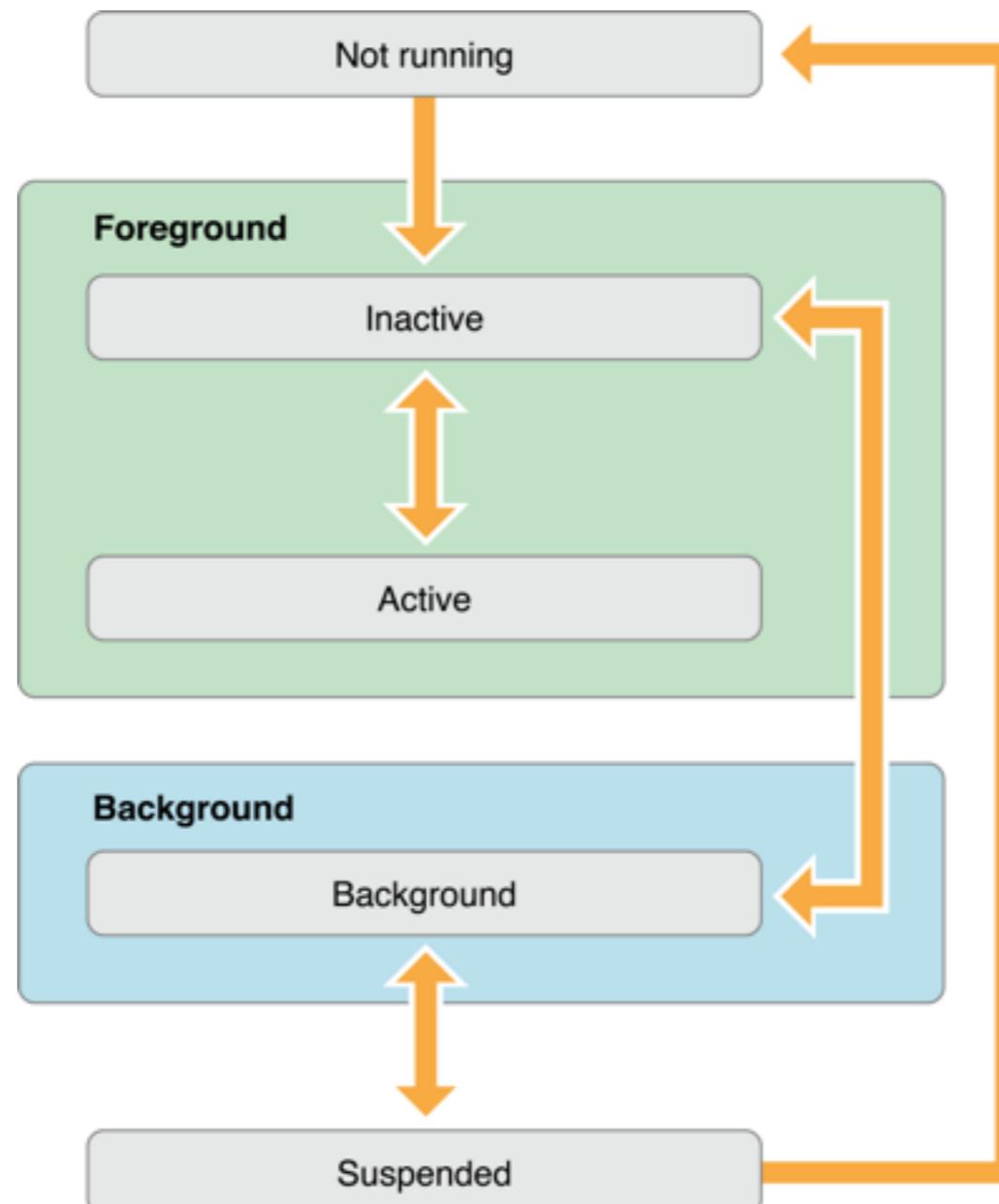
Jeder Block der mit addObserverForName hinzugefügt wurde, wird ausgeführt.

Entweder sofort, auf der selben Queue wie postNotification (wenn queue nil war)...

Oder asynchron durch einstellen des Blocks auf die queue spezifiziert mit addObserverForName.

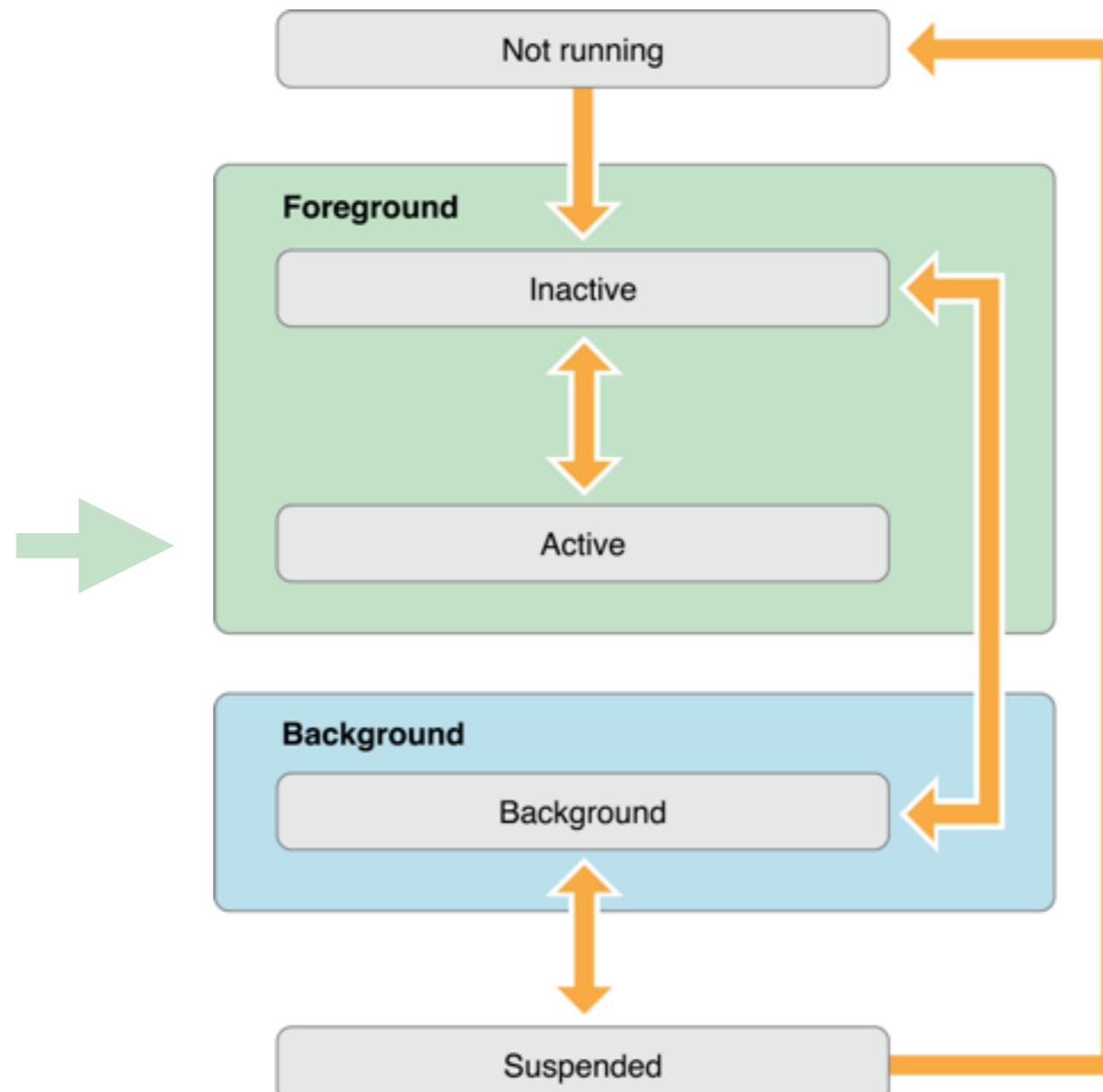
Application Lifecycle

Laufender Code,
aber keine UI Events.



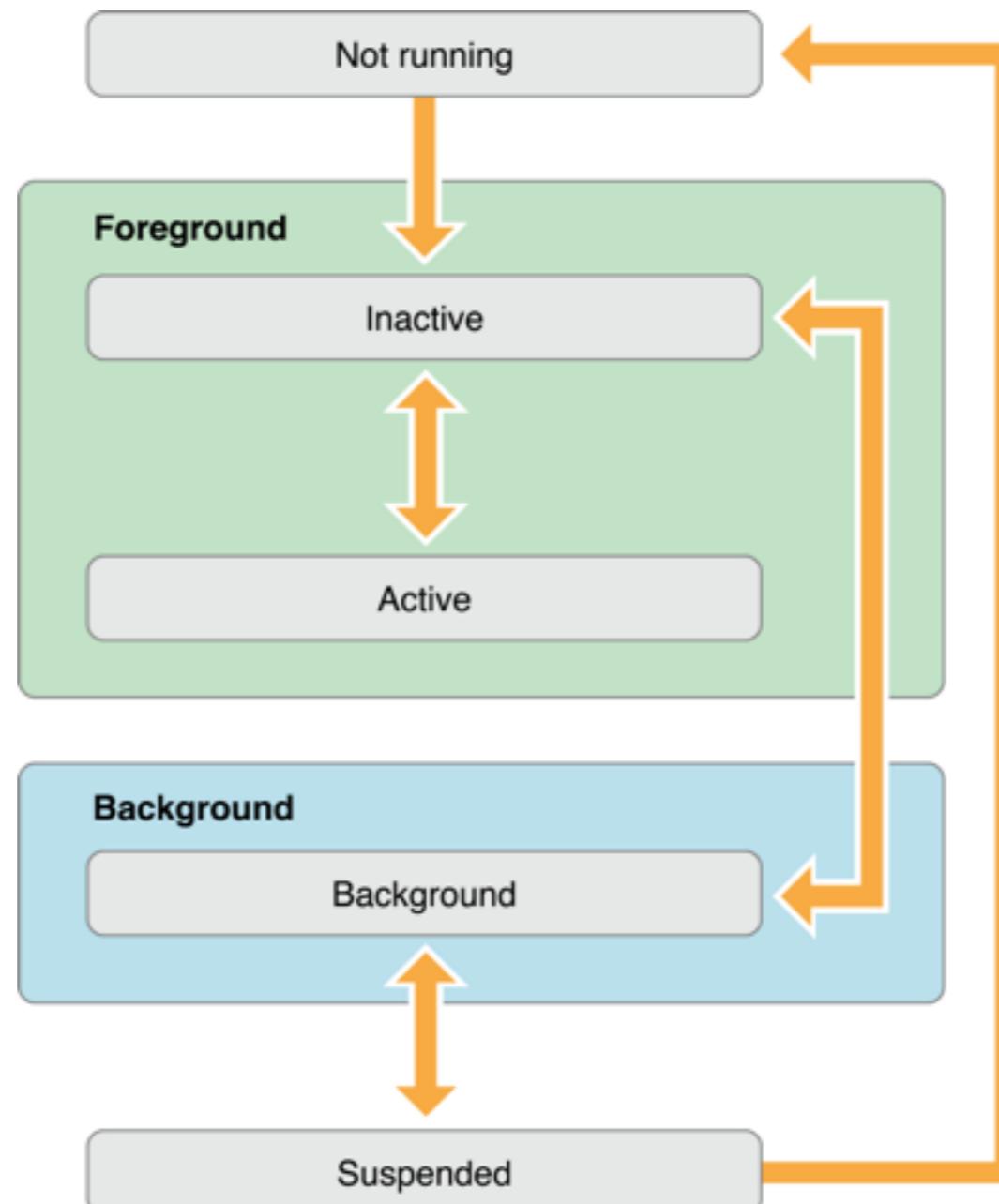
Application Lifecycle

Laufender Code,
Empfang und
Verarbeitung von
UI Events.

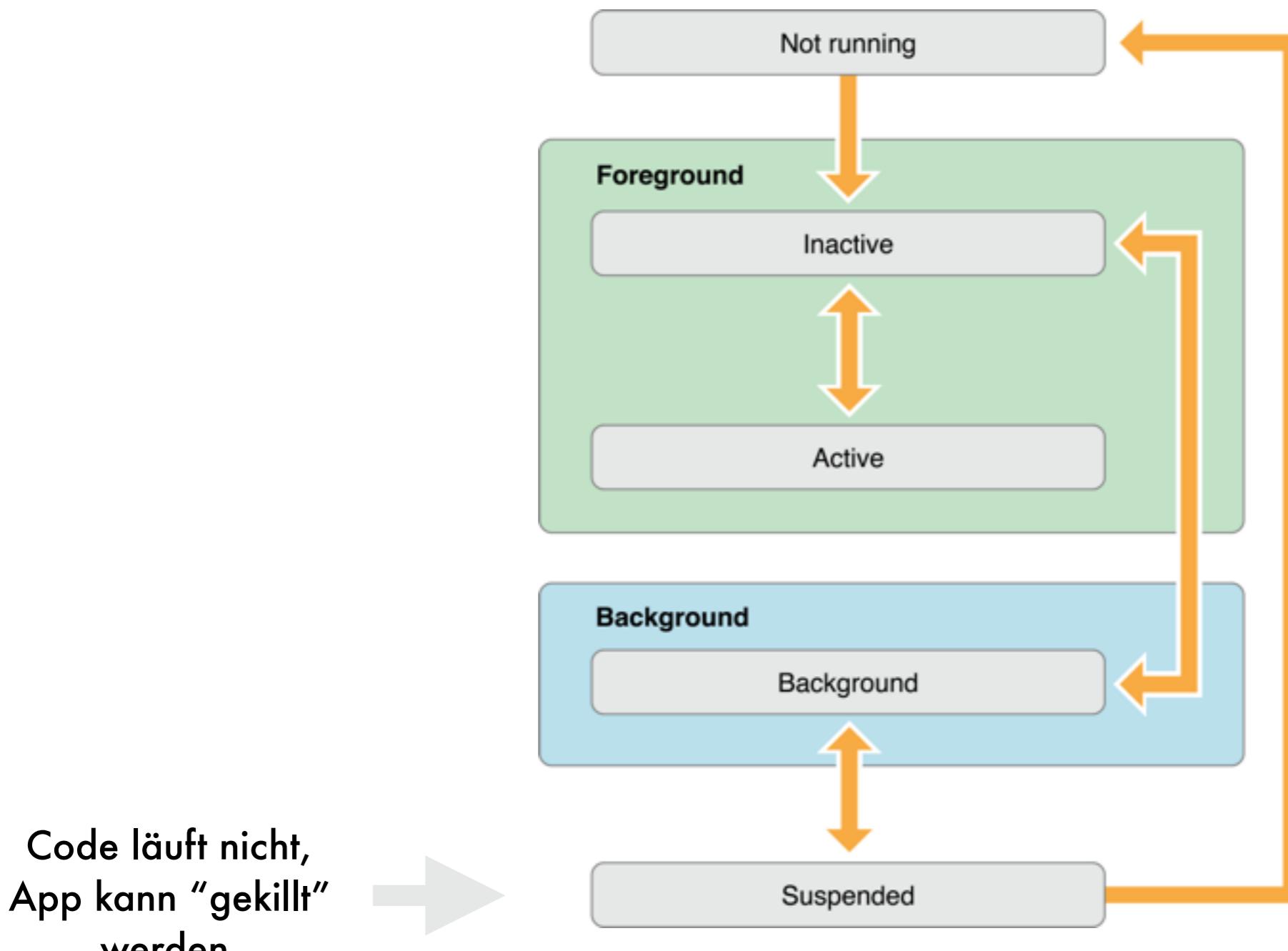


Application Lifecycle

Laufender Code,
für limitierte Zeit,
keine UI Events.

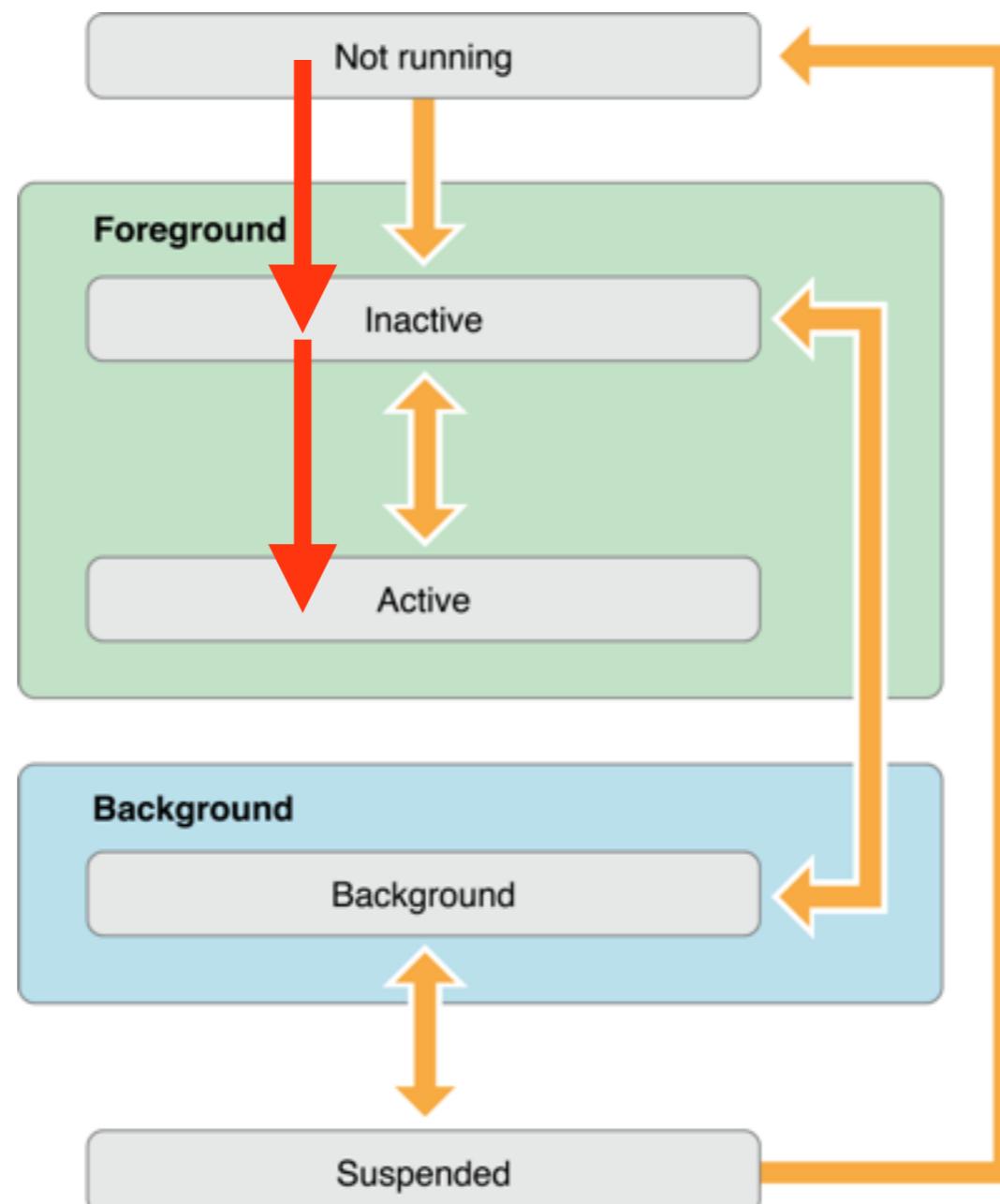


Application Lifecycle



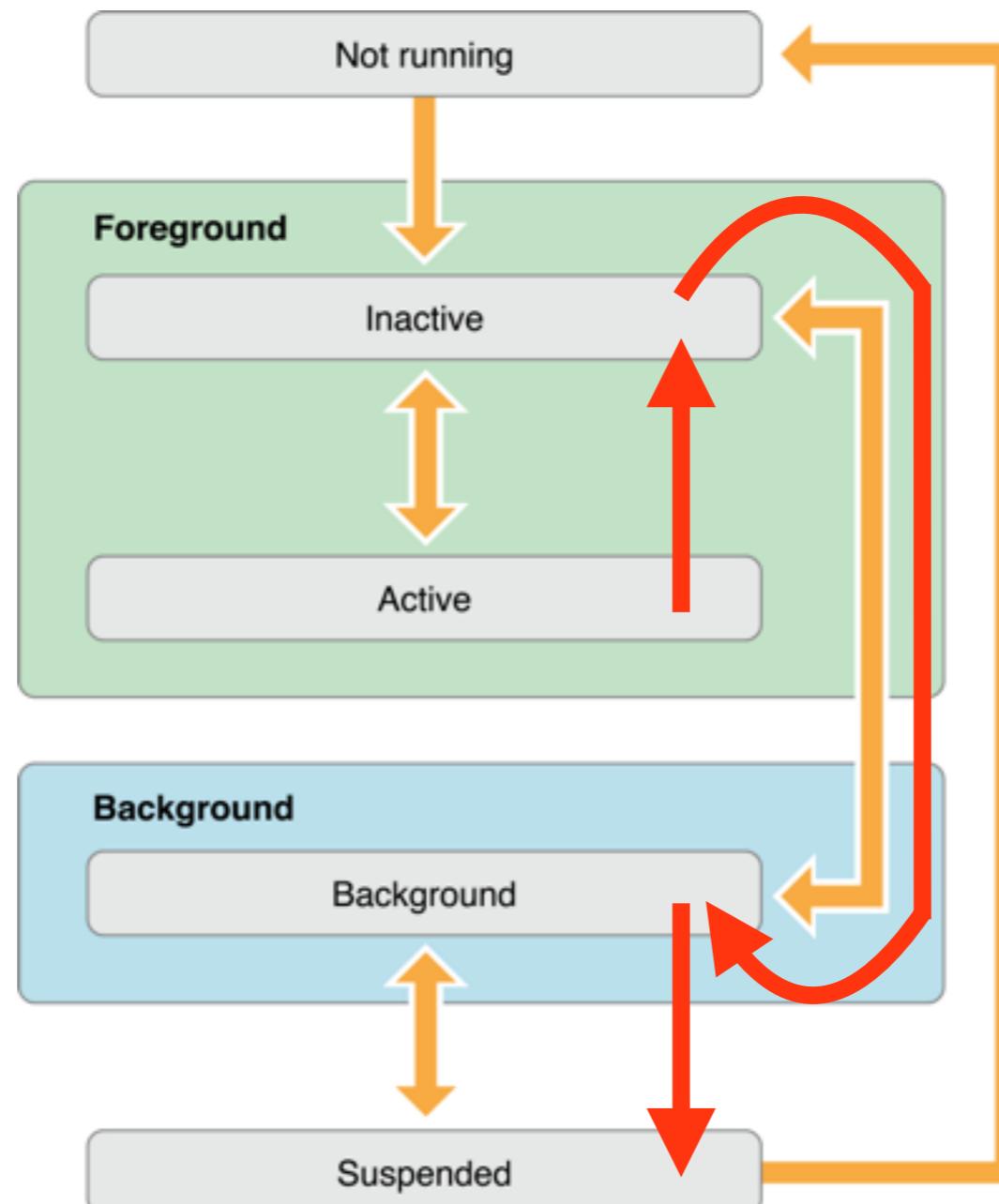
Application Lifecycle

Launch

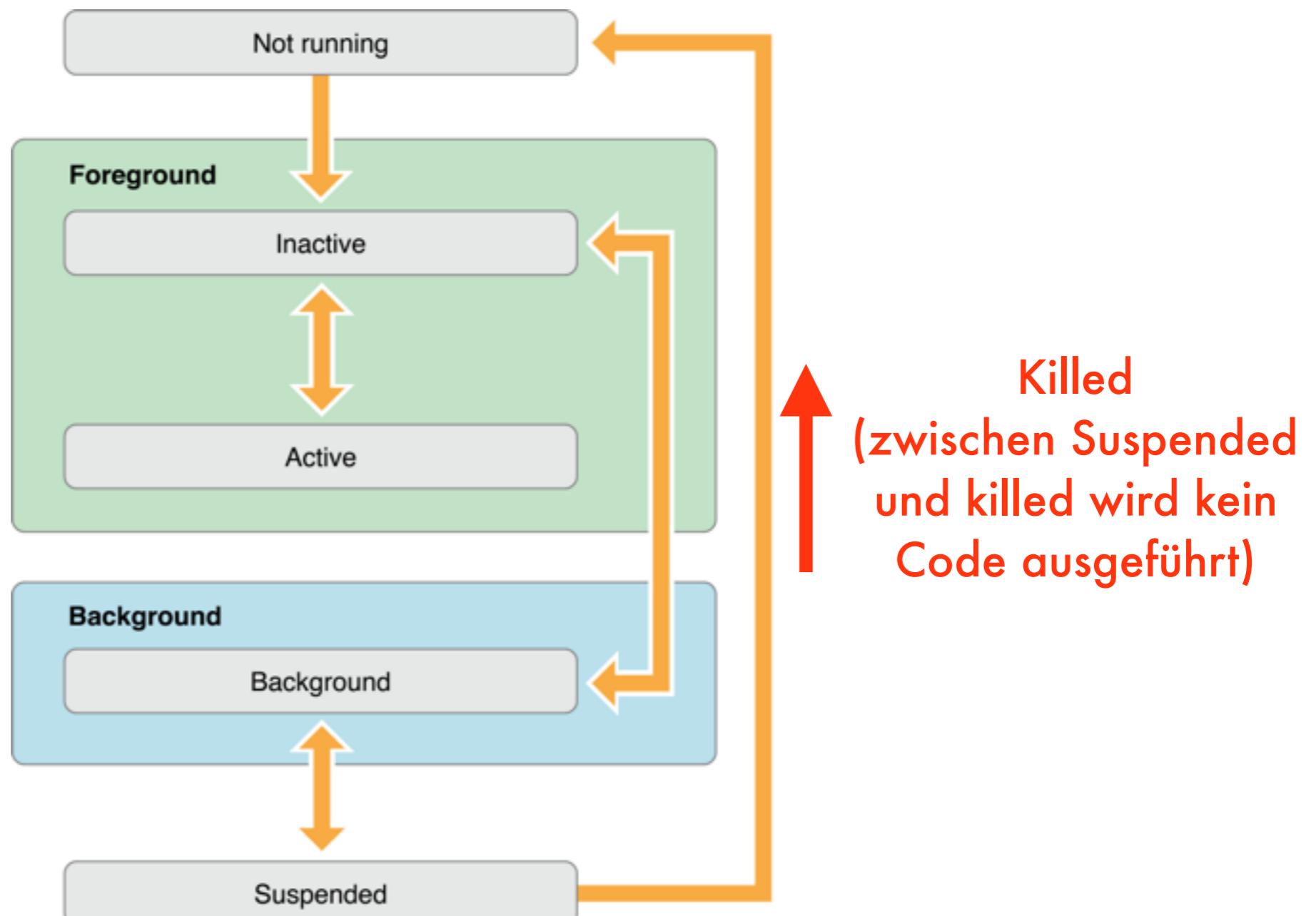


Application Lifecycle

Switch zu
anderer App



Application Lifecycle



Application Lifecycle

- Das AppDelegate empfängt...

```
func application(application,  
didFinishLaunchingWithOptions: [NSObject: AnyObject])
```

... und wir können beobachten ...

```
UIApplicationDidFinishLaunchingNotification
```

Das übergebene Dictionary (auch in
notification.userInfo) gibt an, warum die App
gestartet wurde.

Beispiele...

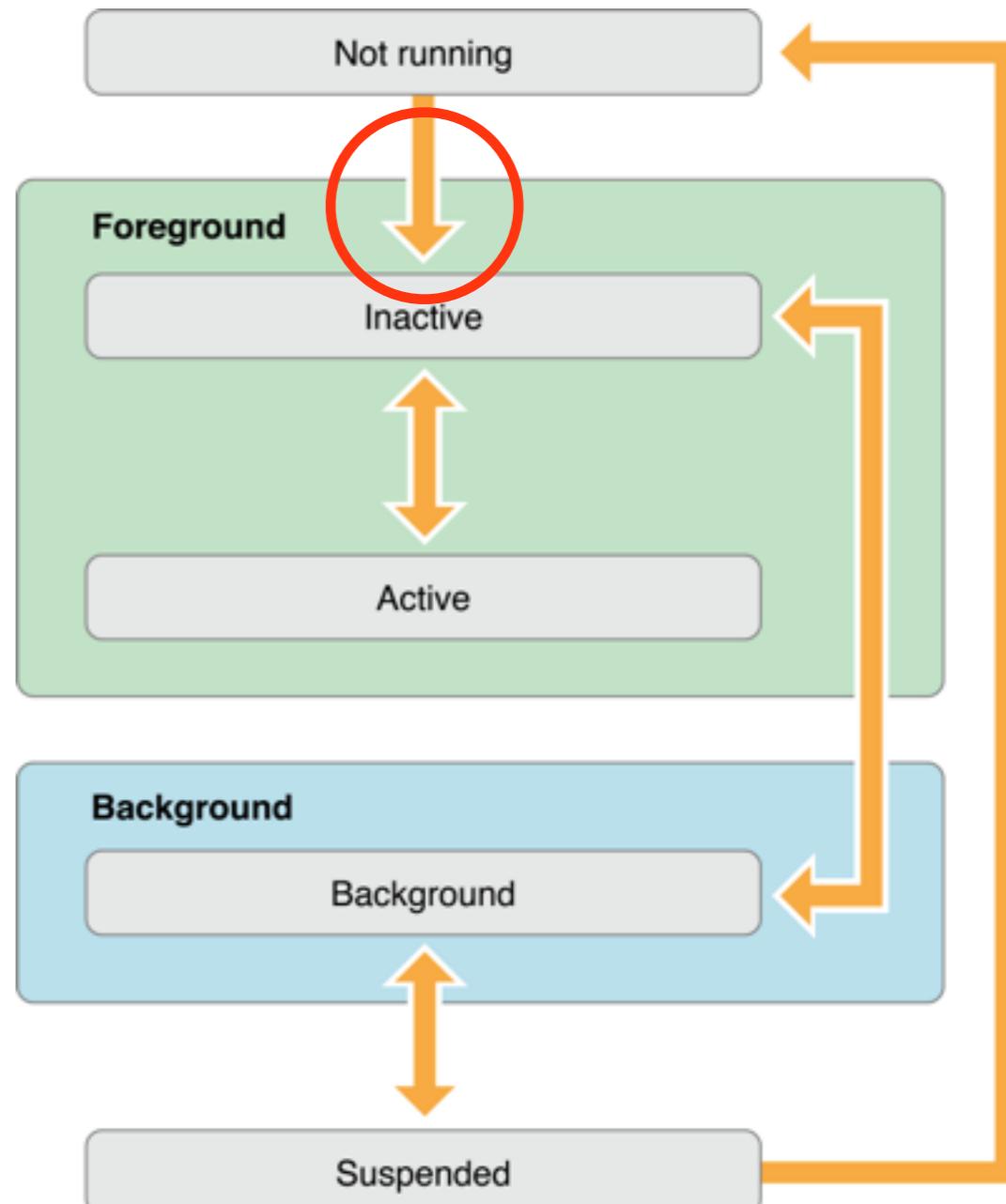
Öffnen einer URL.

Bestimmte Region in der Welt betreten.

Fortsetzen einer Aktivität die auf einem anderen Device
gestartet wurde.

Notification empfangen (push oder lokal).

Angebundenes Bluetooth Device will interagieren.



Application Lifecycle

- Das AppDelegate empfängt...

```
func application(application,  
didFinishLaunchingWithOptions: [NSObject: AnyObject])
```

... und wir können beobachten ...

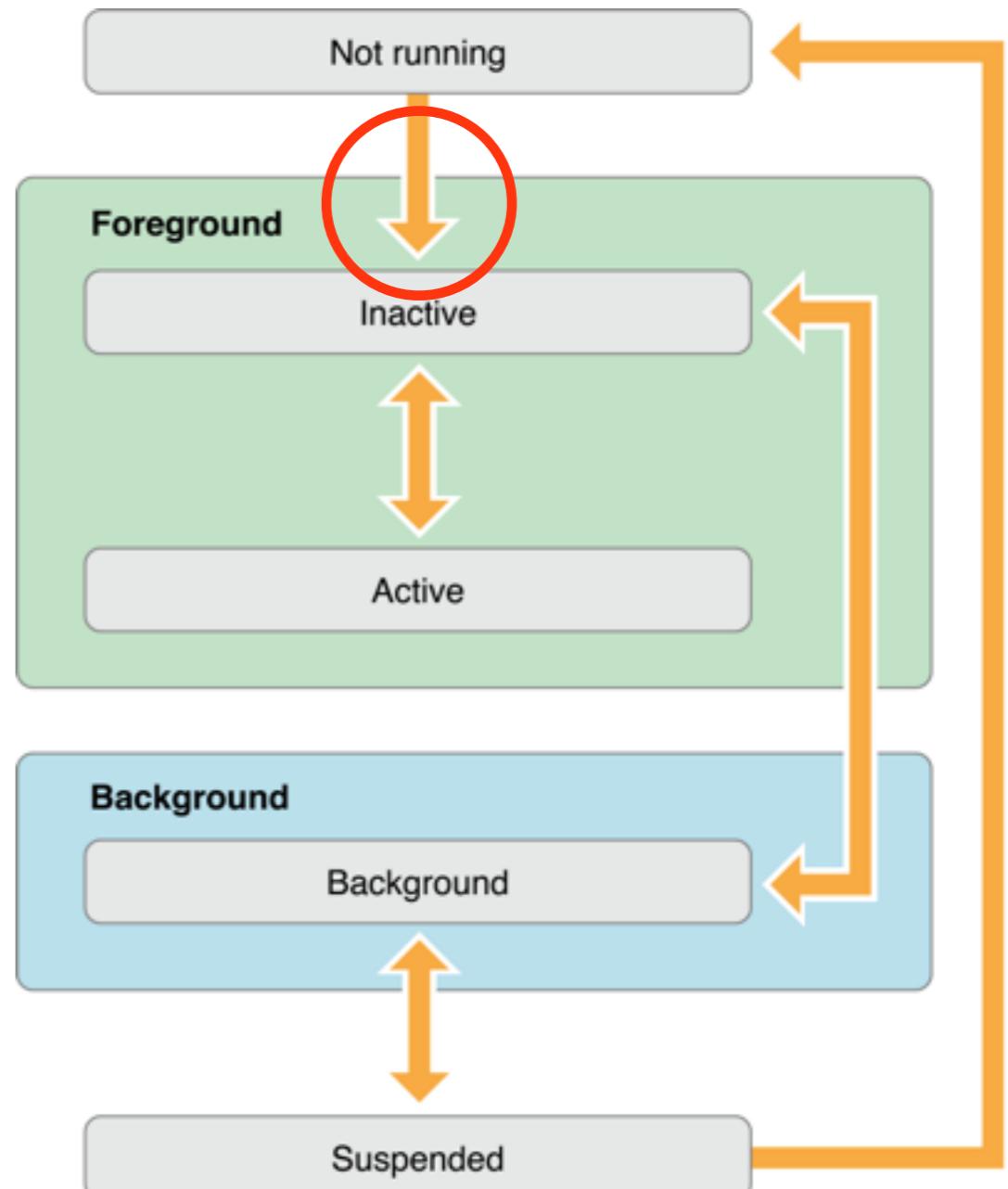
```
UIApplicationDidFinishLaunchingNotification
```

In ferner Vergangenheit wurde hier das UI erstellt.

Zum Beispiel die Instanzierung eines Split View Controllers und das hinzufügen eines Navigation Controllers, dann pushen eines Content View Controllers, usw.

Heute existieren dafür Storyboards, die all dies erledigen.

Daher wird diese Methode meist nicht implementiert.



Application Lifecycle

- Das AppDelegate empfängt...

```
func applicationWillResignActive(UIApplication)
```

... und wir können beobachten ...

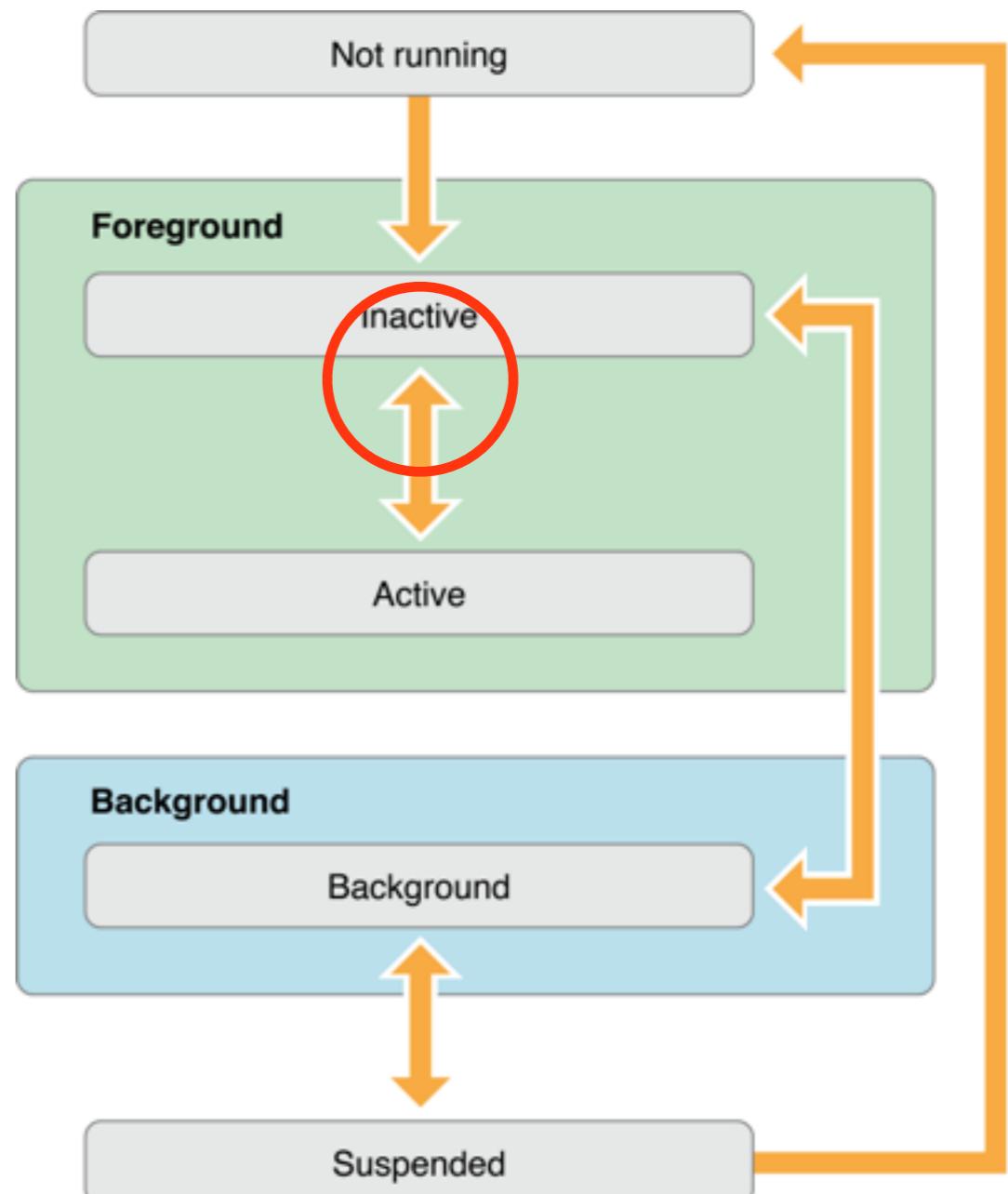
```
UIApplicationWillResignActiveNotification
```

An dieser Stelle das UI "pausieren".

Zum Beispiel beendet Breakout einen abprallenden Ball, Tetris einen fallenden Block, usw.

Dies passiert z.B. weil ein Anruf ankommt.

Oder die App auf dem Weg in den Background ist.



Application Lifecycle

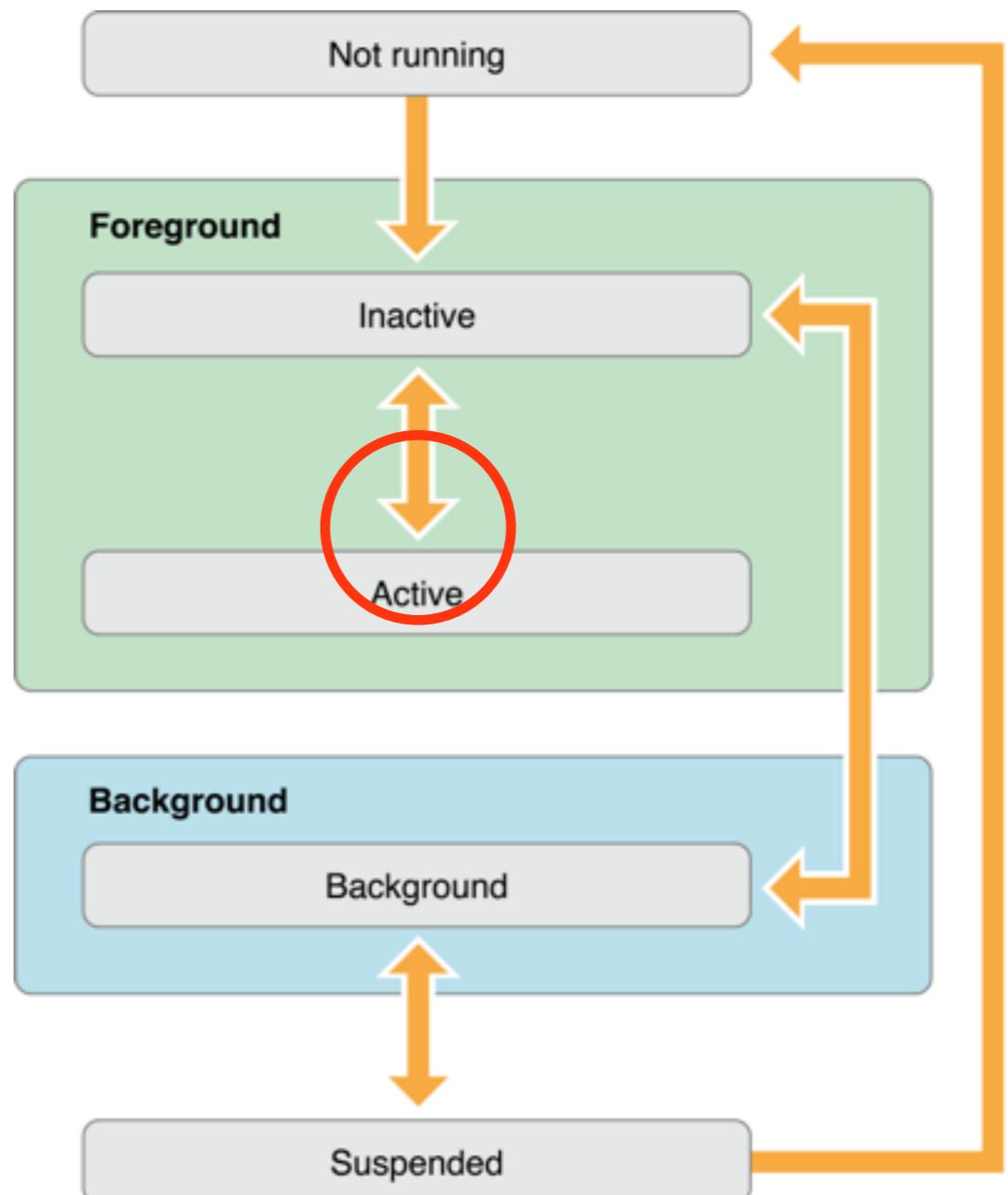
- Das AppDelegate empfängt...

```
func applicationDidBecomeActive(UIApplication)
```

... und wir können beobachten ...

```
UIApplicationDidBecomeActiveNotification
```

Wenn das UI vorher “pausiert” wurde, können Dinge hier wieder aktiviert werden.



Application Lifecycle

- Das AppDelegate empfängt...

```
func applicationDidEnterBackground(UIApplication)
```

... und wir können beobachten ...

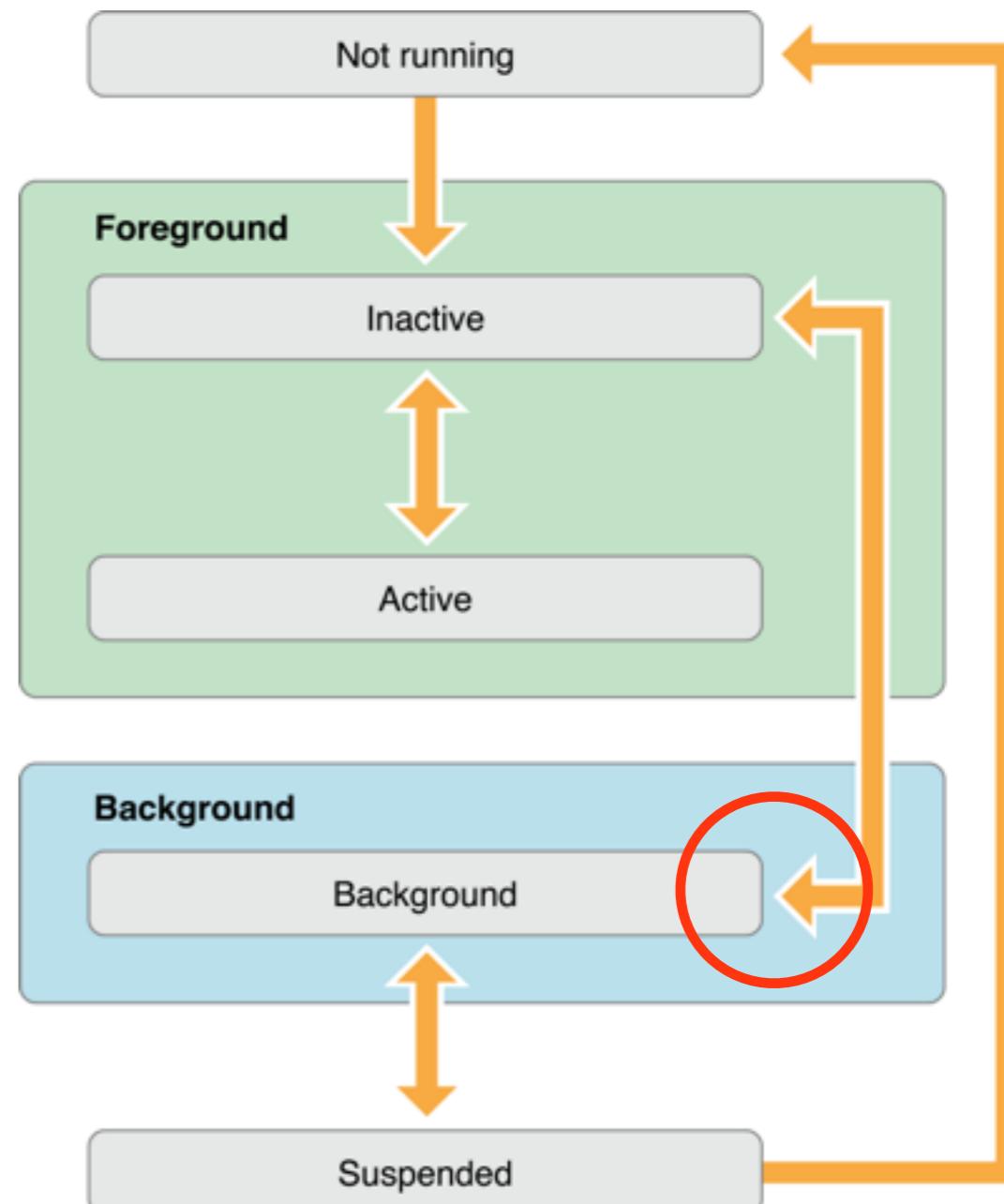
```
UIApplicationDidEnterBackgroundNotification
```

Hier (schnell) reagieren und aufräumen.

Wir haben nur kurz Zeit dafür.

Es gibt die Möglichkeit mehr Zeit anzufordern, dies sollte aber nicht missbraucht werden (da das System uns dann automatisch killt).

Darauf vorbereiten an dieser Stelle gekillt zu werden (passiert wahrscheinlich nicht, aber sollten wir in Betracht ziehen).



Application Lifecycle

- Das AppDelegate empfängt...

```
func applicationWillEnterForeground(UIApplication)
```

... und wir können beobachten ...

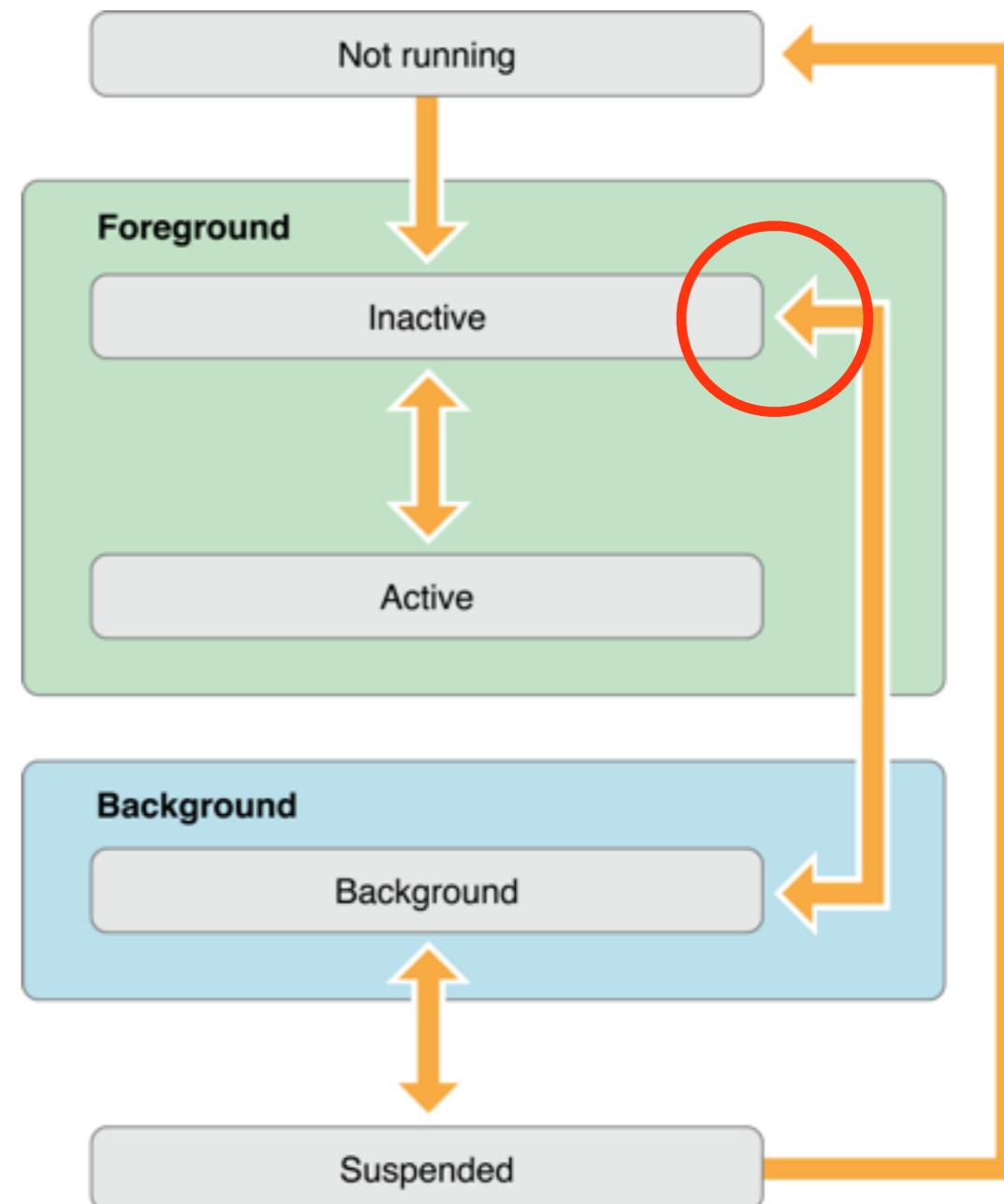
```
UIApplicationWillEnterForegroundNotification
```

Glück gehabt, wir sind nicht aus dem Background State gekillt worden!

Zeit hier wieder "aufzuräumen".

Dinge rückgängig machen, die wir in DidEnterBackground gemacht haben.

Mit hoher Wahrscheinlichkeit werden wir bald Active gesetzt.



UIApplicationDelegate

- Andere AppDelegate Items von Interesse

Lokale Notifications (setzen von Timern für eine bestimmte Zeit ... weckt die App auf).

Remote (Push) Notification (ankommende Informationen von Daten Servern).

State Restoration (Speichern des Zustandes des UI, so dass dies wieder hergestellt werden kann, wenn wir gekillt werden).

Data Protection (Files können geschützt werden, wenn der Device Screen gelockt ist).

Öffnen einer URL (im Xcode Info Tab der Project Settings kann eine URL registriert werden).

Background Fetching (Abfragen und Empfangen von Ergebnissen im Background).

UIApplication

- **Shared Instance**

Es existiert eine einzige `UIApplication` Instanz in unserer App.

```
let myApp = UIApplication.sharedApplication()
```

Managed sämtliches globales Verhalten.

Kein Bedarf dies je zu subclassen.

Delegiert alles in das wir involviert sein müssen zu seinem `UIApplicationDelegate`.

Hat einige nützliche Eigenschaften...

- **Öffnen einer URL in einer anderen App**

```
func openURL(NSURL)
```

```
func canOpenURL(NSURL) -> Bool
```

- **Registrierung oder scheduling von Notifications (Push oder Lokal)**

```
func (un)registerForRemoteNotifications()
```

```
func scheduleLocalNotification(UILocalNotification)
```

```
func registerUserNotificationSettings(UIUserNotificationSettings)
```

UIApplication

- **Setzen des Fetch Interval für Background Fetching**

Müssen wir setzen, wenn Background Fetching funktionieren soll...

```
func setMinimumBackgroundFetchInterval(NSTimeInterval)
```

Setzen wir gewöhnlich auf UIApplicationBackgroundFetchIntervalMinimum

- **Fragen nach mehr Zeit, wenn im Background**

```
func backgroundTaskWithExpirationHandler(handler: () -> Void) ->  
    UIBackgroundTaskIdentifier
```

Niemals vergessen endBackgroundTask(UIBackgroundTaskIdentifier) aufzurufen, wenn wir fertig sind!

- **Aktivieren des “Network in Use” Spinner (im Statusbar oben)**

```
var networkActivityIndicatorVisible: Bool
```

- **Dinge abfragen/erfahren**

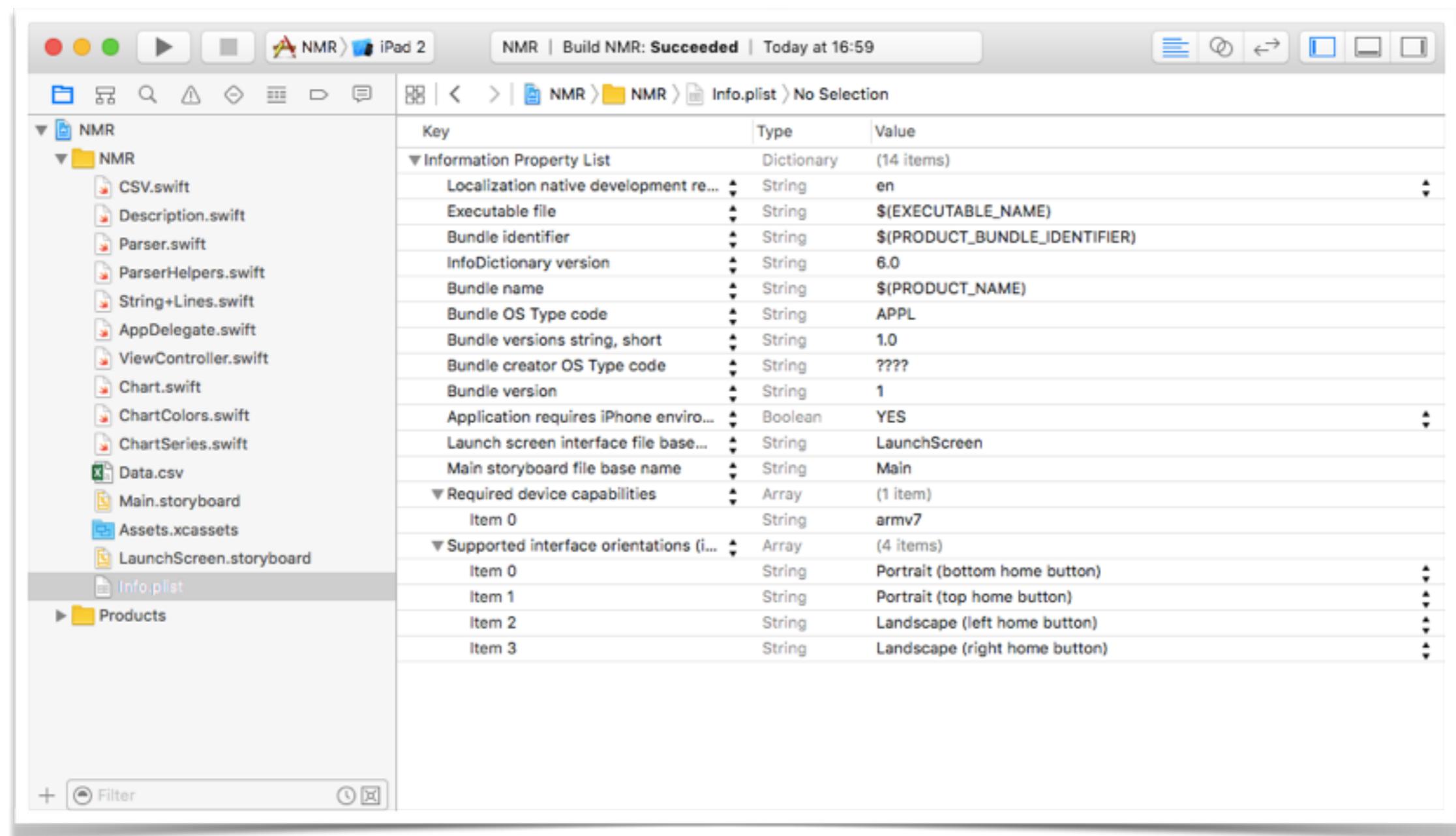
```
var backgroundTimeRemaining: NSTimeInterval { get } // bis zu Suspended
```

```
var preferredContentSizeCategory: String { get } // Große oder kleine Fonts
```

```
var applicationState: UIApplicationState { get } // Foreground, Background, Active
```

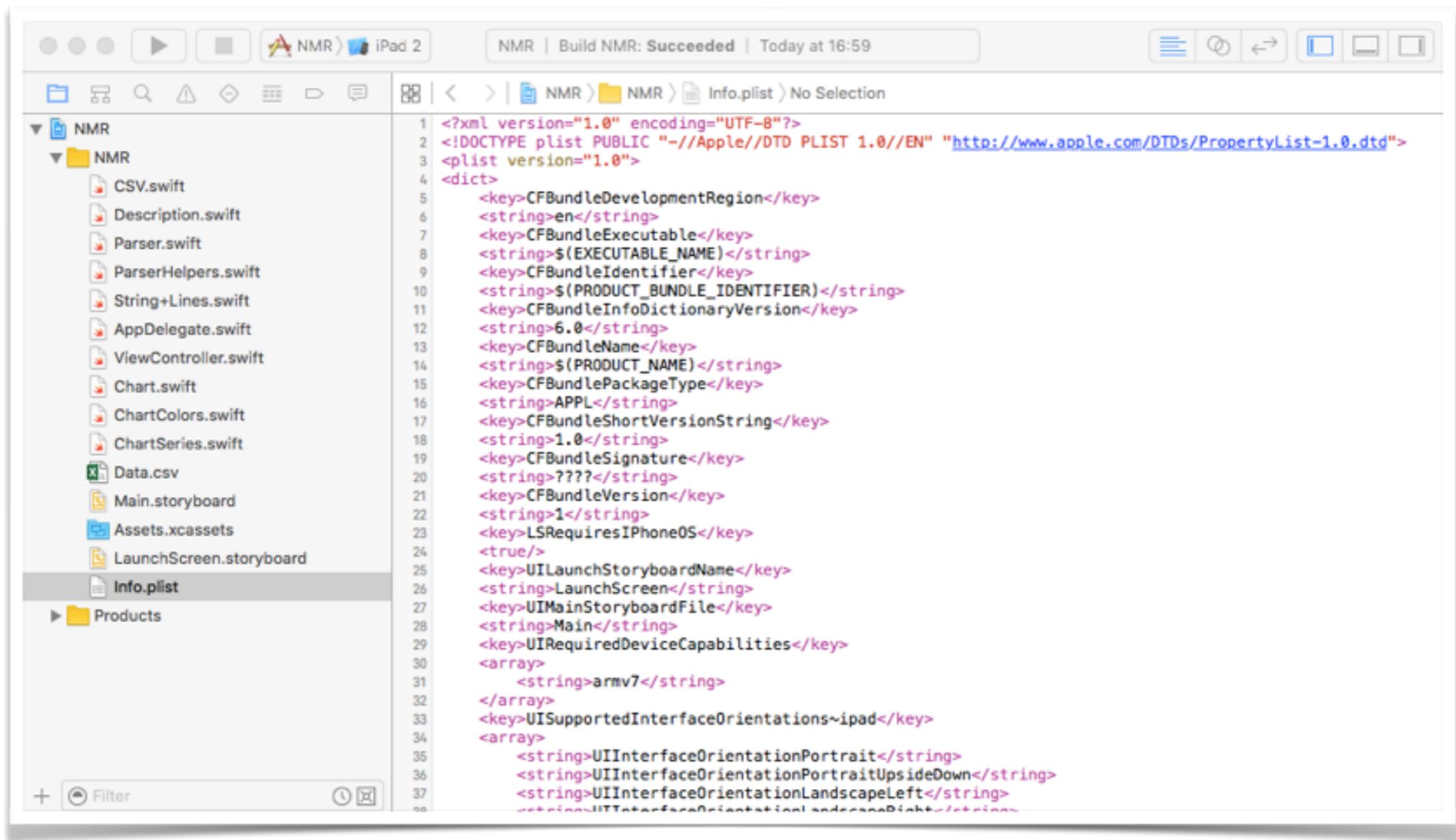
Info.plist

- Viele der Application Settings stehen in Info.plist
Können wir direkt (im Xcode Property List Editor) editieren.



Info.plist

- Viele der Application Settings stehen in Info.plist
Können wir direkt (im Xcode Property List Editor) editieren.
Oder sogar als raw XML.

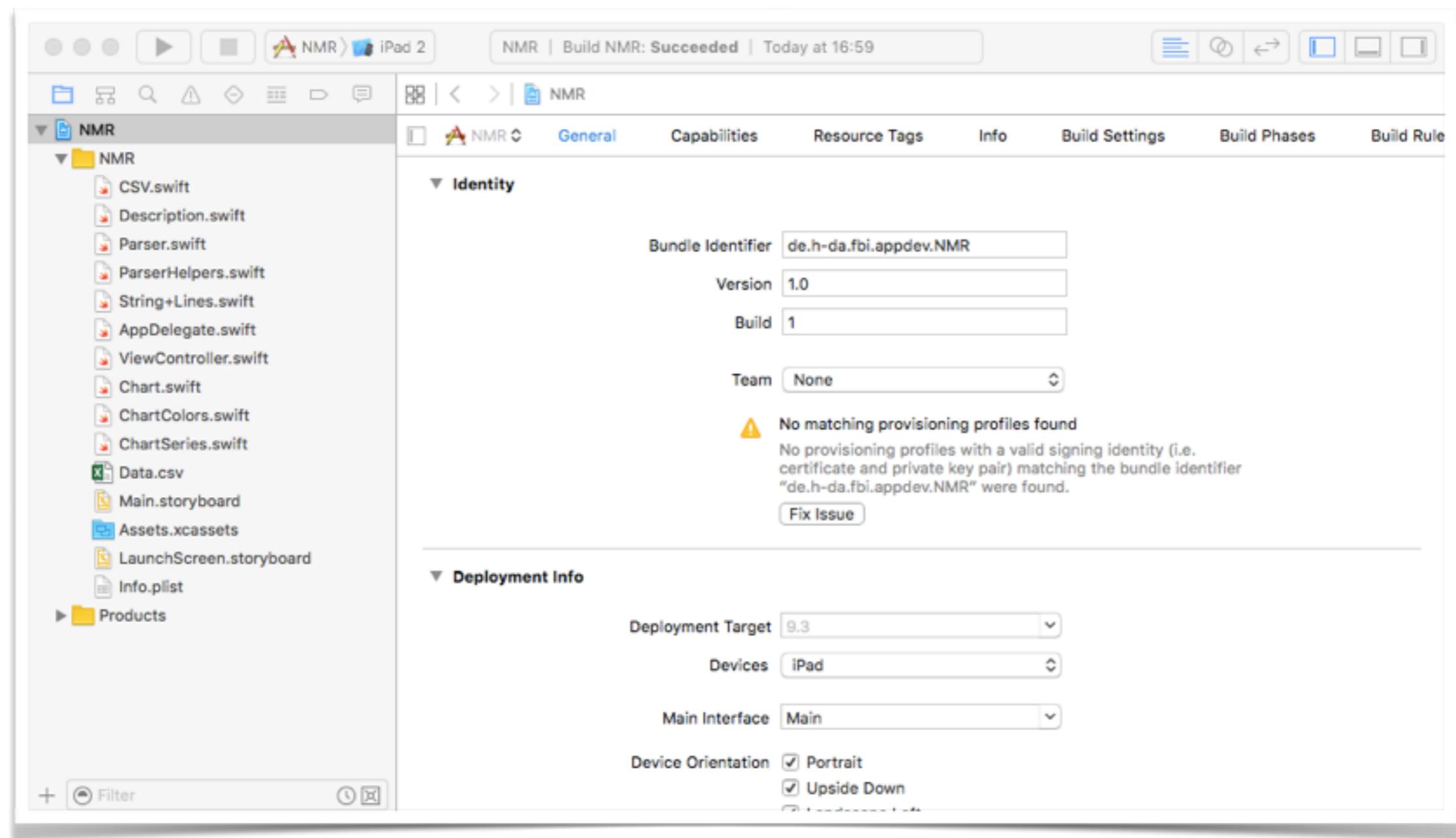


The screenshot shows the Xcode interface with the Project Navigator on the left and the Editor area on the right. In the Project Navigator, under the 'NMR' project, the 'Info.plist' file is selected and highlighted with a grey background. The Editor area displays the raw XML content of the Info.plist file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>CFBundleDevelopmentRegion</key>
    <string>en</string>
    <key>CFBundleExecutable</key>
    <string>$EXECUTABLE_NAME</string>
    <key>CFBundleIdentifier</key>
    <string>$PRODUCT_BUNDLE_IDENTIFIER</string>
    <key>CFBundleInfoDictionaryVersion</key>
    <string>6.0</string>
    <key>CFBundleName</key>
    <string>$PRODUCT_NAME</string>
    <key>CFBundlePackageType</key>
    <string>APPL</string>
    <key>CFBundleShortVersionString</key>
    <string>1.0</string>
    <key>CFBundleSignature</key>
    <string>????</string>
    <key>CFBundleVersion</key>
    <string>1</string>
    <key>LSRequiresiPhoneOS</key>
    <true/>
    <key>UILaunchStoryboardName</key>
    <string>LaunchScreen</string>
    <key>UIMainStoryboardFile</key>
    <string>Main</string>
    <key>UIRequiredDeviceCapabilities</key>
    <array>
        <string>armv7</string>
    </array>
    <key>UISupportedInterfaceOrientations~ipad</key>
    <array>
        <string>UIInterfaceOrientationPortrait</string>
        <string>UIInterfaceOrientationPortraitUpsideDown</string>
        <string>UIInterfaceOrientationLandscapeLeft</string>
        <string>UIInterfaceOrientationLandscapeRight</string>
    </array>
</dict>
</plist>
```

Info.plist

- Viele der Application Settings stehen in Info.plist
Können wir direkt (im Xcode Property List Editor) editieren.
Oder sogar als raw XML.
Normalerweise editieren wir die Info.plist Settings durch klick auf Project im Navigator.



Capabilities

- Einige Features müssen eingeschaltet werden

Dies sind Server und Interoperability Features.

Wie iCloud, Game Center, etc.

- Einschalten in Capabilities Tab
Innerhalb der Project Settings.

- Sehr umfangreich!

Nachschlagen in der Doku!

Viele benötigen allerdings eine volle Developer Mitgliedschaft.

Am besten aber schon jetzt damit vertraut machen.



Continuous Integration

- Warum Continuous Integration?

- Test mit mehreren OS Versionen und verschiedenen Hardware Modellen.

- Häufiger Build, Analyze und Tests.

- Schnelles und automatisches Erkennen von Problemen.

- Server-basiert (Auslagerung).

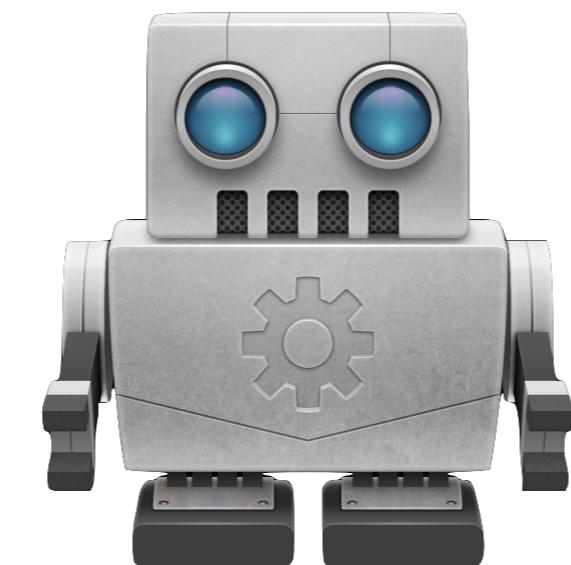
- Erstellen von Build und Test History für Projekte.

- Verteilen von Builds an das Team.

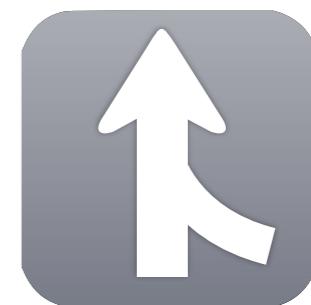
Continuous Integration



Scheme
Wie wird das Projekt gebaut?



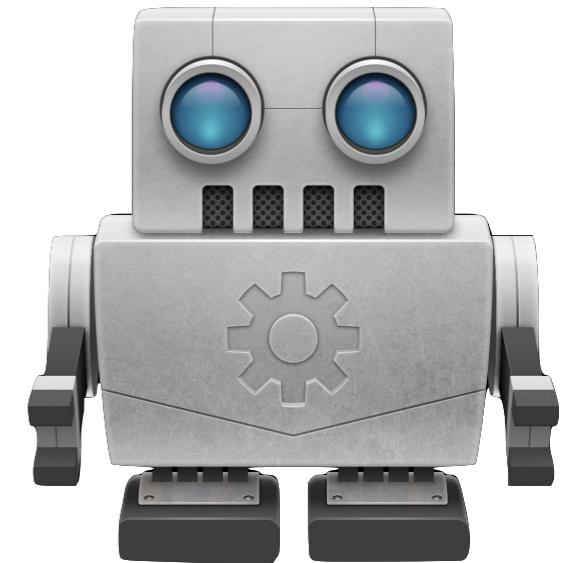
Bot
Analyze, Build, Test und
Archive nach Plan



Integration
Run the Bot

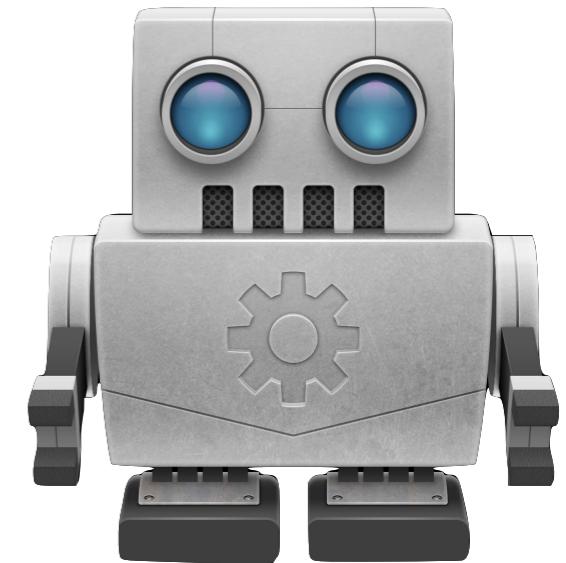
Continuous Integration

- **Was wird gebaut?**
Projekt und SCM Informationen
- **Wann wird gebaut?**
Periodisch, on-commit, manuell
- **Wie wird gebaut**
Projekt und SCM Informationen



Continuous Integration

- Was wird gebaut?
Projekt und SCM Informationen
- Wann wird gebaut?
Periodisch, on-commit, manuell
- Wie wird gebaut?
Shared Scheme
Static Analysis
Testing und Devices
Archives
- Notifications



Continuous Integration

- **Voraussetzung**

OS X (am 10.12 macOS) + Xcode Service + Xcode.

- **Xcode Service**

Bonjour Discovery.

Erstellen und Management von Bots welche OS X und iOS Projekte bauen.

Zugriffsrechte (Bot Creator und Viewer).

Server dem Development Team hinzufügen (Managed auch Profiles, Certificates, usw.).

Devices.

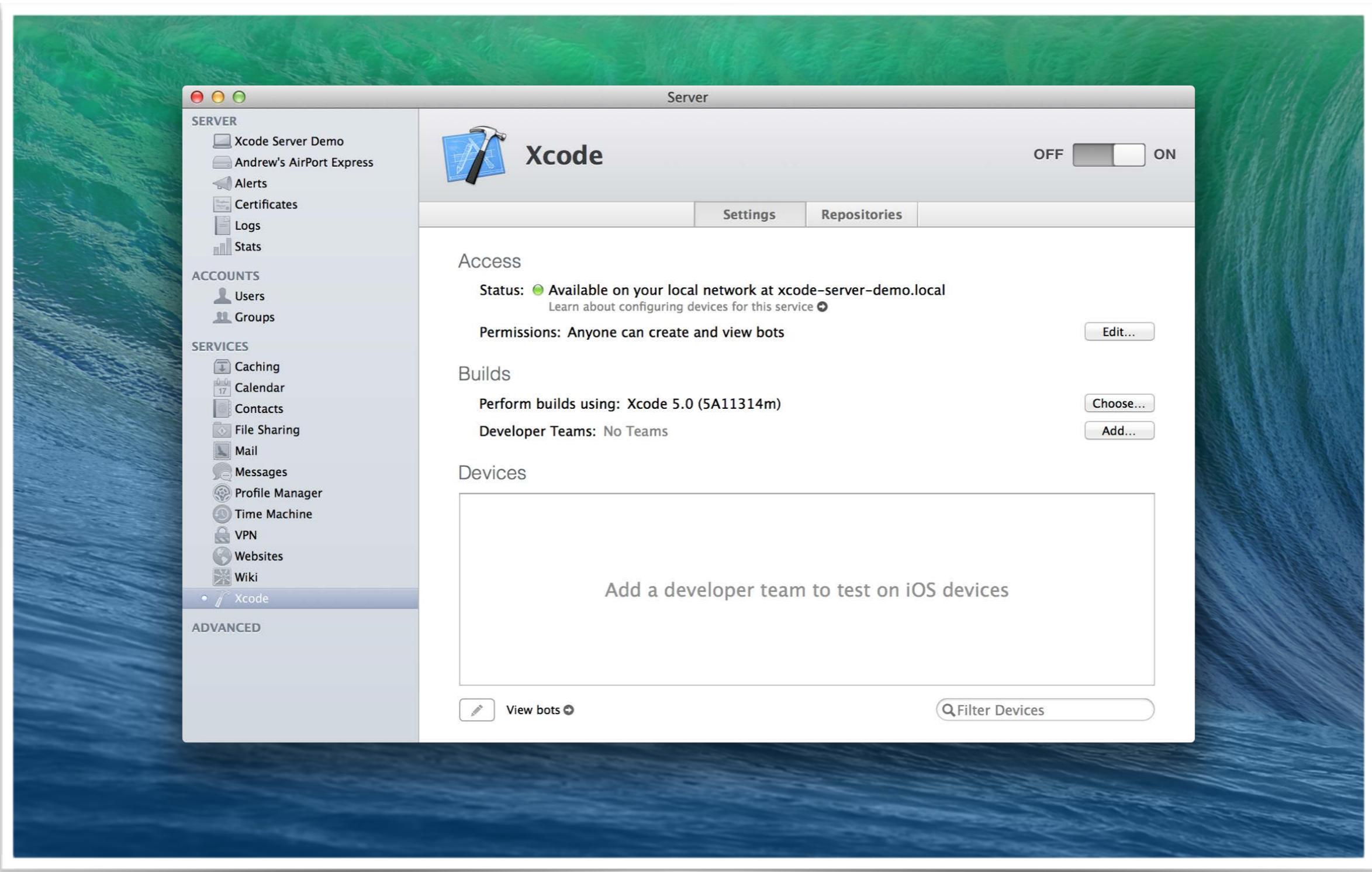
Hinzufügen von Devices zum Team.

Tests auf mehreren iOS Devices.

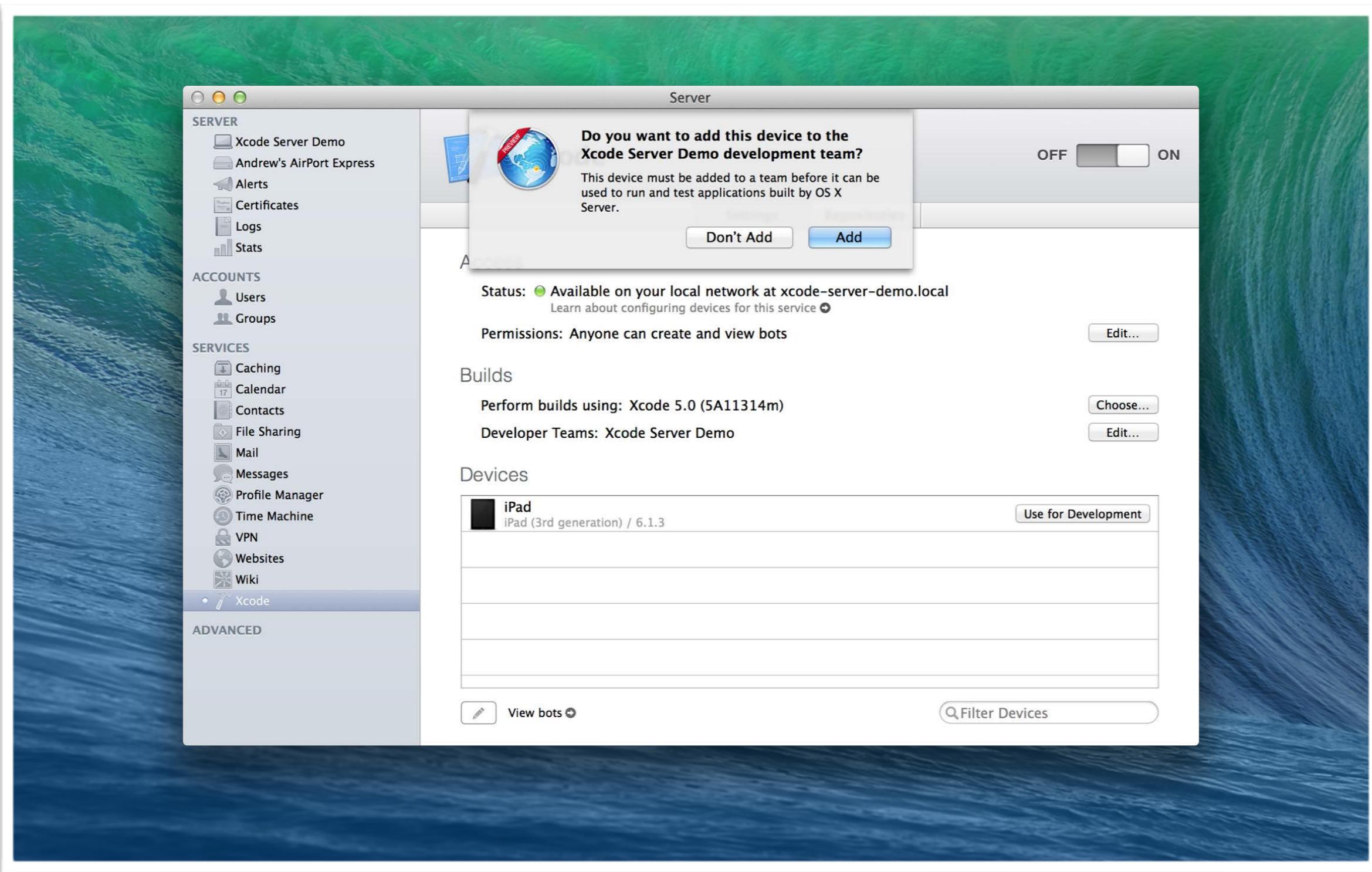
Hosten von Git Repositories

Verbindung zu nicht-lokalen Git Repos

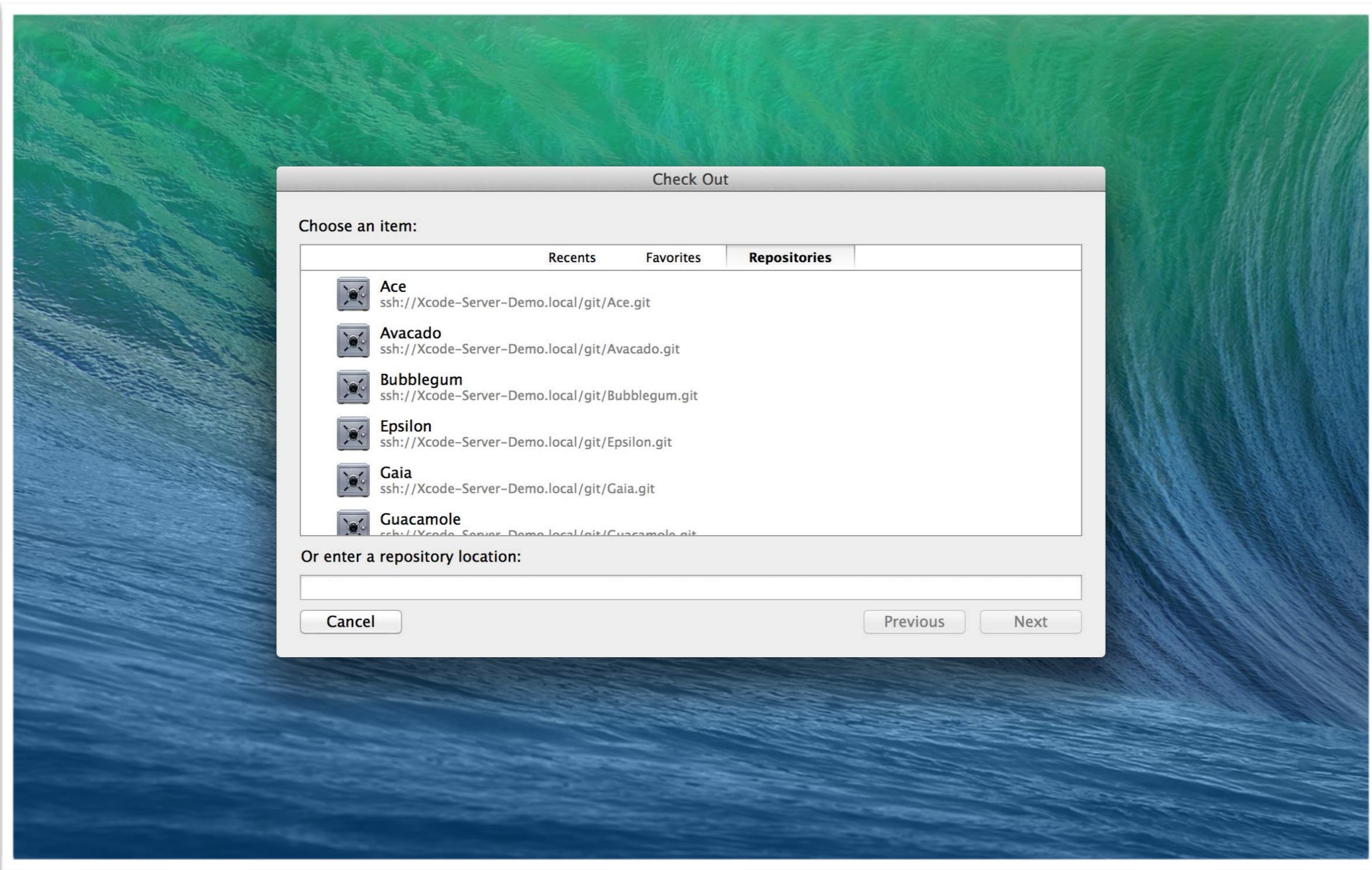
Continuous Integration



Continuous Integration



Continuous Integration



Continuous Integration

The screenshot shows the Xcode interface for a project named "Bubblegum". The main window displays the "Bubblegum" integration results. Key statistics are shown: 0 Errors, 1 Warning, 1 Analysis, and 4 Tests Failed. The "Build History" chart shows a series of bars for builds 268 through 298, with most builds being "No Issues" (yellow) and one build (286) showing analysis issues (blue). The "Test History" chart shows a series of bars for builds 268 through 298, with most tests being "Success" (green) and some showing failures (red). The "Integration Details" section lists a warning about suggested flavors and an analysis issue regarding the initialization of 'flavors'.

Bubblegum.xcodeproj

Bubblegum: Ready | Today at 1:06 PM

Project Yesterday, 6:41 PM Push Bubblegum Yesterday, 6:41 PM

Bubblegum Xcode Server Demo

- Integrate (298) Yesterday, 6:41 PM
- Integrate (297) Yesterday, 6:26 PM
- Integrate (296) Yesterday, 6:13 PM
- Integrate (295) Yesterday, 6:12 PM
- Integrate (294) Yesterday, 6:11 PM
- Integrate (293) Yesterday, 6:10 PM
- Integrate (289) Yesterday, 6:06 PM
- Integrate (288) Yesterday, 6:05 PM
- Integrate (287) Yesterday, 6:04 PM
- Integrate (286) Yesterday, 5:54 PM

Load More... 282 more integrations on server

Integration Results

Finished with test failures

Start: 6/7/13, 6:41 PM Duration: 53 seconds
2 commits by xcodeserverdemo...

0 Errors 1 Warning 1 Analysis 4 Tests Failed

Build History

■ Errors ■ Warnings ■ Analysis Issues

No Issues No Issues

268 269 270 271 272 273 274 275 276 277 278 281 282 283 284 285 286 287 288 289 293 294 295 296 297 298

Test History

■ Failure ■ Success

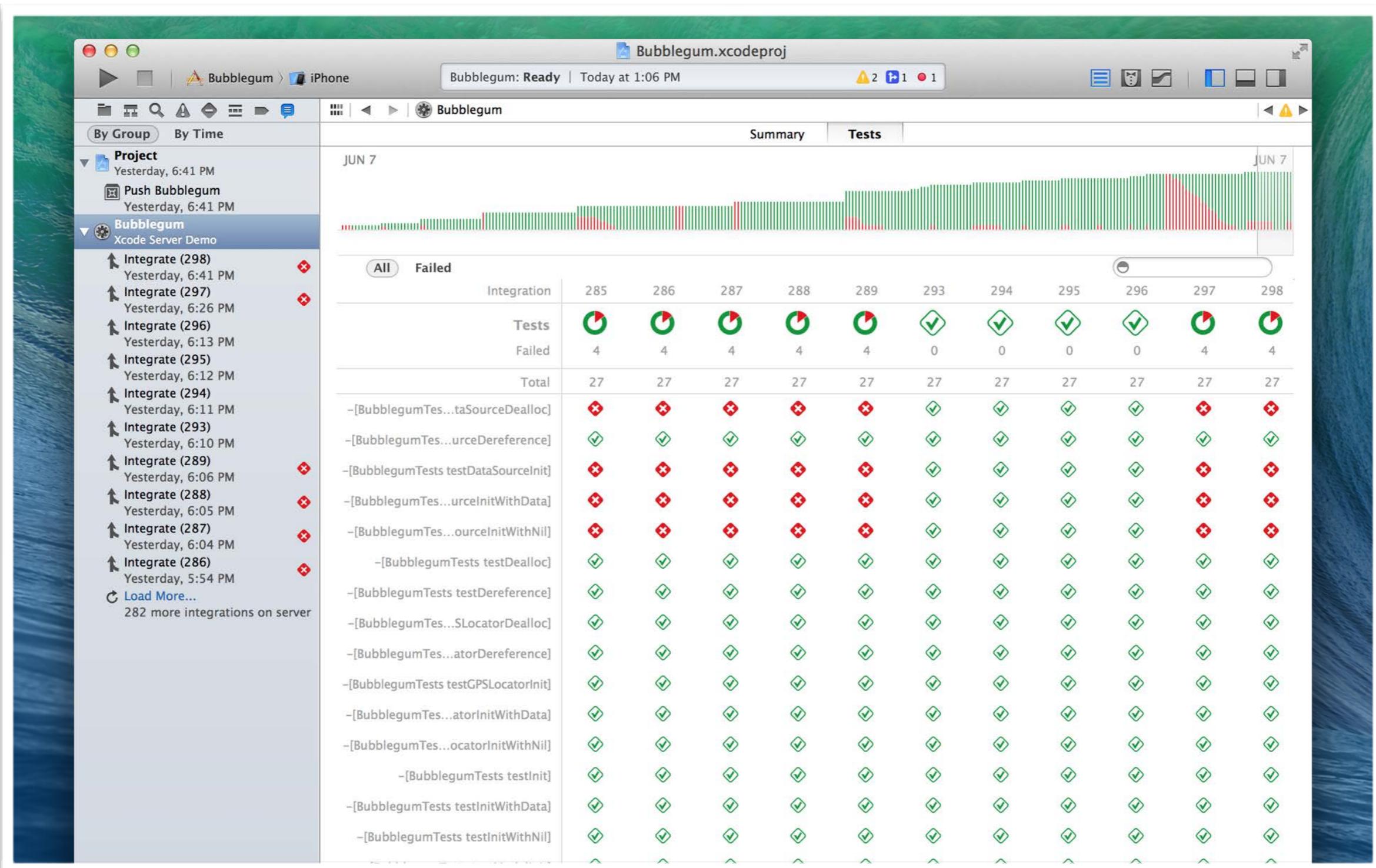
268 269 270 271 272 273 274 275 276 277 278 281 282 283 284 285 286 287 288 289 293 294 295 296 297 298

Integration Details

⚠ Warnings
Add suggested flavors here some day
Bubblegum — BubblegumFactory.m

✖ Analysis Issues
Value stored to 'flavors' during its initialization is never read
Bubblegum — BubblegumFactory.m

Continuous Integration

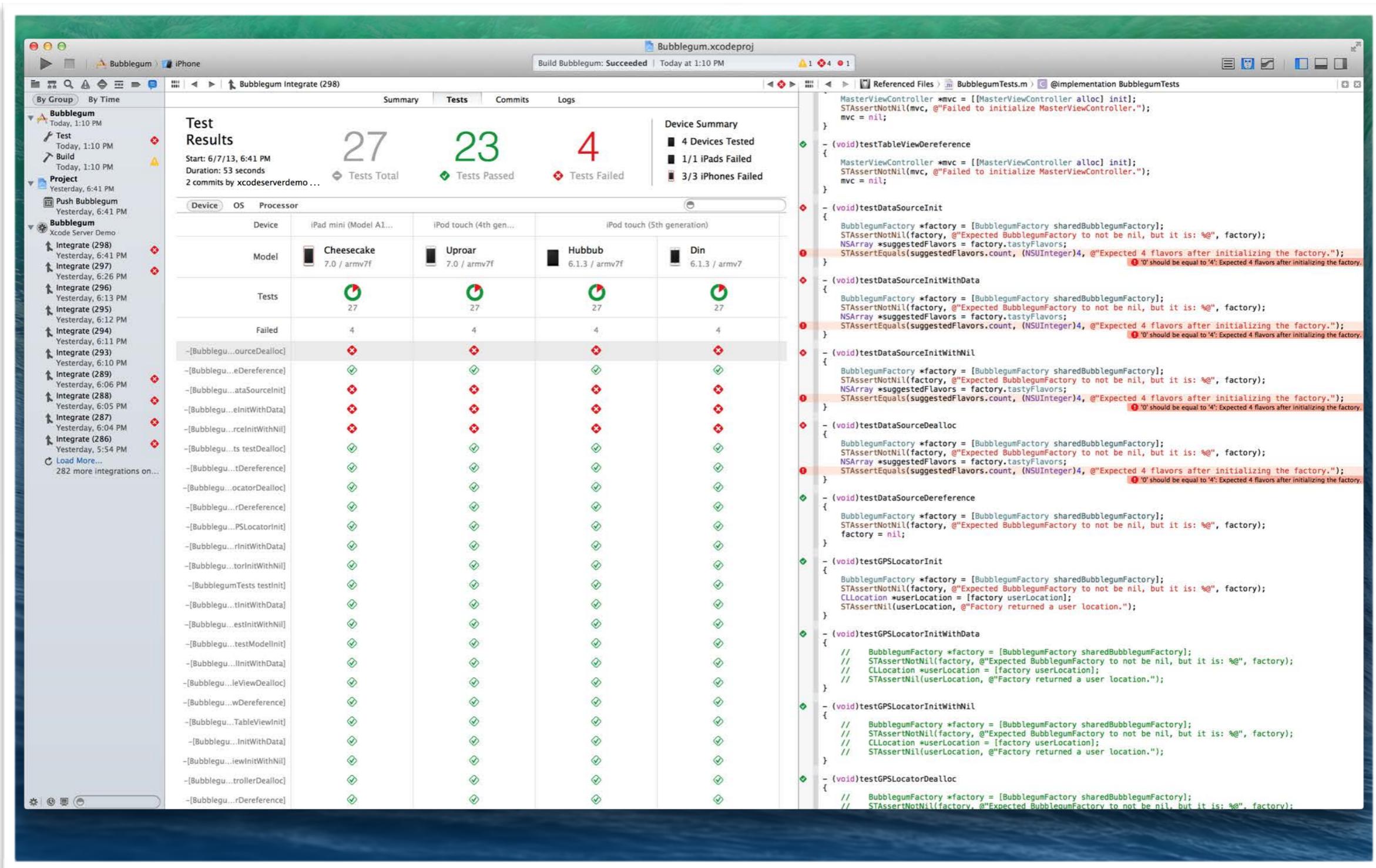


Continuous Integration

The screenshot shows the Xcode interface during a continuous integration build for the project 'Bubblegum.xcodeproj'. The main window displays the 'Test Results' for a build named 'Bubblegum Integrate (298)'. The summary indicates 27 total tests, with 23 passed and 4 failed. The test results are broken down by device: iPad mini (Model A1432), iPod touch (4th generation), iPod touch (5th generation), and Din (6.1.3 / armv7). The table below provides a detailed breakdown of the test results across these devices.

Test	iPad mini (Model A1432)	iPod touch (4th gen...)	iPod touch (5th generation)	Din
Tests Total	27	27	27	27
Failed	4	4	4	4
-[Bubblegum sourceDealloc]	✗	✗	✗	✗
-[Bubblegum eDereference]	✓	✓	✓	✓
-[Bubblegum ataSourceInit]	✗	✗	✗	✗
-[Bubblegum elnItWithData]	✗	✗	✗	✗
-[Bubblegum rceInitWithNil]	✗	✗	✗	✗
-[Bubblegum ts testDealloc]	✓	✓	✓	✓
-[Bubblegum tDereference]	✓	✓	✓	✓
-[Bubblegum oocatorDealloc]	✓	✓	✓	✓
-[Bubblegum rDereference]	✓	✓	✓	✓
-[Bubblegum PSLocatorInit]	✓	✓	✓	✓
-[Bubblegum rInitWithData]	✓	✓	✓	✓
-[Bubblegum tInitWithNil]	✓	✓	✓	✓

Continuous Integration



Continuous Integration

The screenshot shows the Xcode interface for managing continuous integration bots. The window title is "xcsdemo.wwdc/xcode/bots/bubblegum2". The navigation bar includes "Xcode", "Bots", and "Bubblegum". The main area displays a table of build history for the "Bubblegum" bot, with the "Archives" tab selected. The table has columns for Type, Integration, Date, Name, and Size. Most entries are for Product type, while some are for Archive. The Date column shows dates from June 7, 2013. The Name column lists file names like "Bubblegum.ipa" and "Archive.xcarchive.zip". The Size column shows file sizes like "100 KB" and "185 KB". The "Archive" column contains small icons representing each build. The "Logs" column is empty. The "Integrate" button is located in the top right corner of the table header.

Type	Integration	Date	Name	Size
Product	298	6/7/2013	Bubblegum.ipa	100 KB
Archive		6/7/2013	Archive.xcarchive.zip	185 KB
Product	297	6/7/2013	Bubblegum.ipa	99 KB
Archive		6/7/2013	Archive.xcarchive.zip	185 KB
Product	296	6/7/2013	Bubblegum.ipa	99 KB
Archive		6/7/2013	Archive.xcarchive.zip	185 KB
Product	295	6/7/2013	Bubblegum.ipa	99 KB
Archive		6/7/2013	Archive.xcarchive.zip	185 KB
Product	294	6/7/2013	Bubblegum.ipa	99 KB
Archive		6/7/2013	Archive.xcarchive.zip	185 KB
Product	293	6/7/2013	Bubblegum.ipa	99 KB
Archive		6/7/2013	Archive.xcarchive.zip	185 KB
Product	289	6/7/2013	Bubblegum.ipa	99 KB
Archive		6/7/2013	Archive.xcarchive.zip	185 KB
Product	288	6/7/2013	Bubblegum.ipa	99 KB
Archive		6/7/2013	Archive.xcarchive.zip	185 KB

Continuous Integration

- Bubblegum 19 hrs ago
- Bubblegum Nightly 1 min ago
- Guacamole 1 day ago
- Guacamole Nightly 21 hrs ago
- Nightly Frameworks Just now
- Proctor 19 hrs ago
- Salsa 1 day ago
- Unit Test Bot 1 day ago
- Zed 1 day ago
- Ace 21 hrs ago

Integration 298

0 1 1 4

! Errors ⚠ Warnings ⚡ Analysis ✘ Failed Tests

1 Committer

Add a warning to remind myself to implement this method some day.

Andrew Nesbitt Yesterday at 6:41 PM 1 file modified

4 Devices

Hubbub Din Uproar Cheesecake

Continuous Integration



On commit
Build, Test und Static Analysis
für jede Code Änderung



Nightly
Build um 3 Uhr Nachts



Per Branch
Erstellen eines neuen
Bots pro Branch

Games

Basic Platform Game

Was wird benötigt?

- Game Art
- Audio Art (Music und Sound Effekte)
- Level Map
- CCLayer Subclass (Physics Stuff)
- CCSprite Subclass (Player Logics)

Physic Engine

Zentraler Ankerpunkt des Spiels (no engine, no game...)

Warum keine fertige Engine verwenden?

- Tuning - Verhalten der Engine sollte einstellbar sein. Eine fertige Engine wird nicht die Ergebnisse bringen wie von einem Super Mario, Sonic, etc. Spiel gewohnt
- Komplexität - existierende Engines sind überladen mit Features. Eine Engine sollte das machen was gebraucht wird, nicht mehr und nicht weniger (auch bzgl. Ressourcen)

Games

Was muss eine Physics Engine können?

Movement

- Gravity (resistive)
- Running, jumping, friction, etc. (applied)

Collision Detection

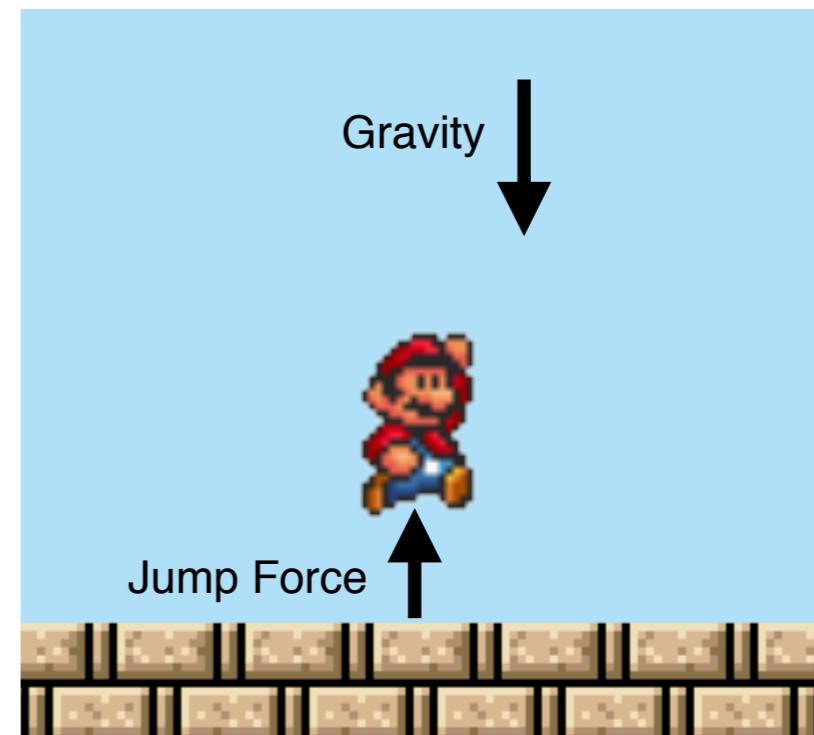
- Erkennen und abarbeiten von Kollisionen des Players und anderen Objekten im Spiel

Gravity und Jump Force

- Interaktion über Zeit

Collision Detection

- Verhindert Fallen durch den Boden
- Schaden bei Spikes/Gegnern/Feuer...



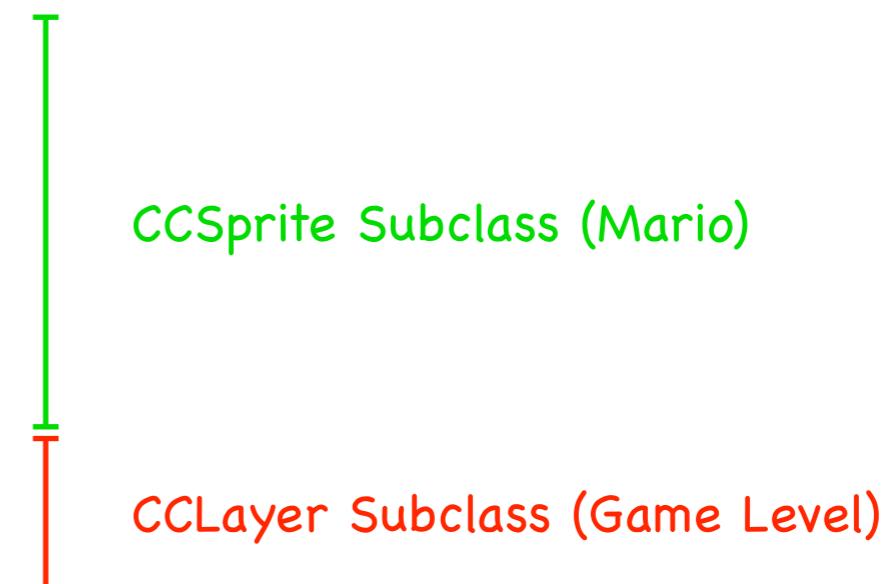
Mario Physics

- Mario (seine Klasse) hat seine eigenen Members

- Current Velocity
- Acceleration
- Position
- etc.

- Jede Änderung folgt diesem Algorithmus (in jedem Frame!)

1. Jump oder move action?
2. If YES, apply forces
3. Apply gravity
4. Compute velocity
5. Apply velocity & update position
6. Collision check
7. If collision, resolve (move back / damage)



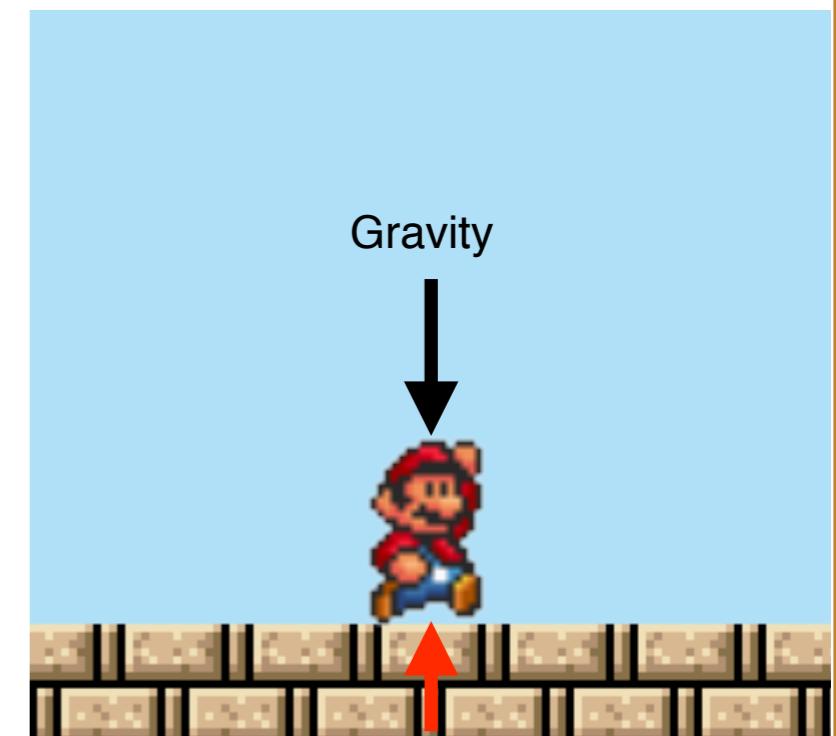
- 1 bis 6 beeinflusst nur den Player (alle Informationen im Objekt)
- ab 6 müssen Wände, Gegner, Boden, Hindernisse, etc. berücksichtigt werden

Fighting Gravity

- Gravity drückt Mario nach unten
- Collision Detection drückt Mario nach oben (bis knapp über den Boden)
- Anders ausgedrückt: Berührt Mario noch den Boden?
 - If NO, disable jump (!)

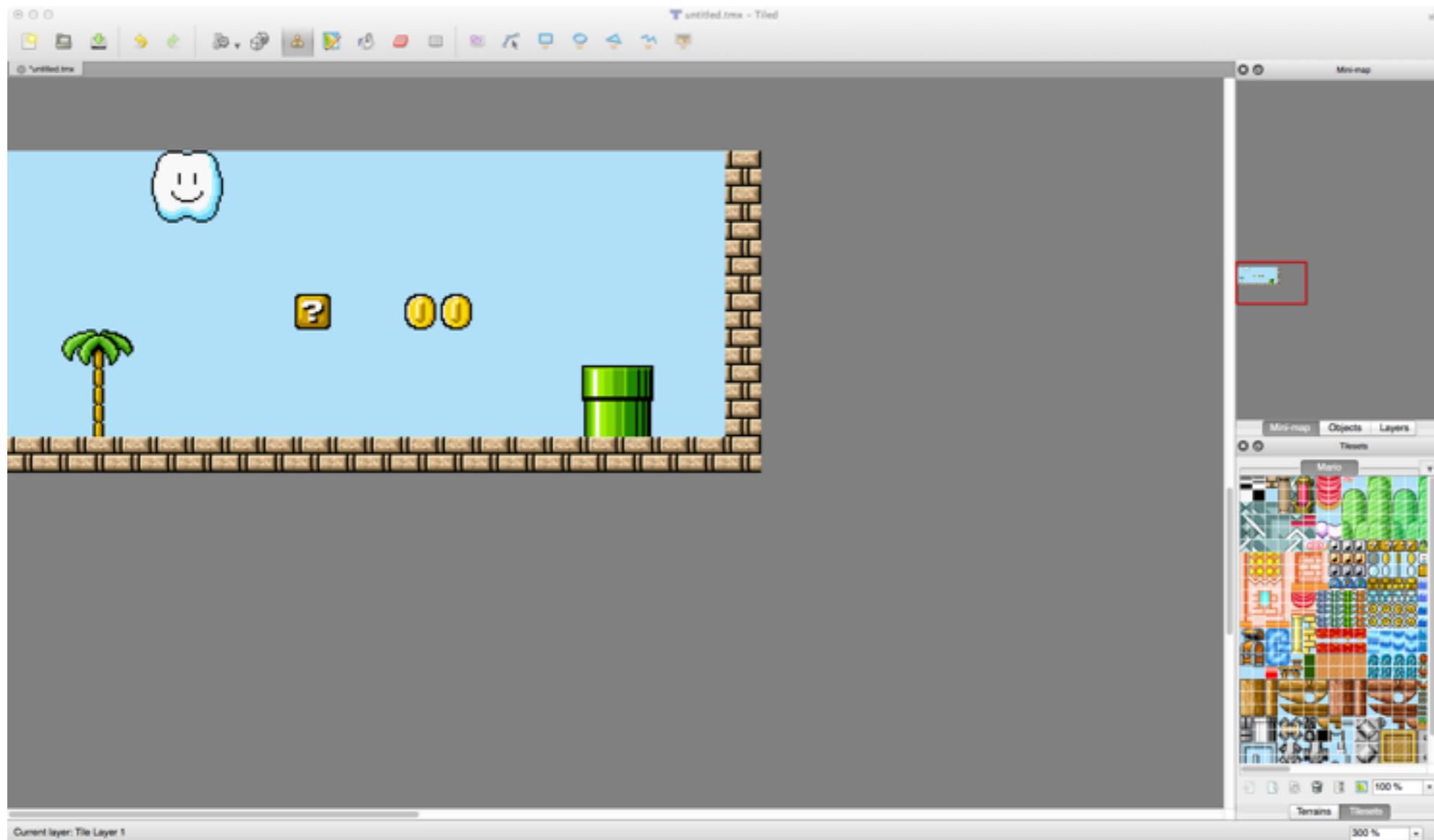
Position

- Wenn Mario seine eigene Position setzen würde, würde er sich selbst in eine Kollision bewegen (Wand, Boden, etc.)
- Der Game Level Layer würde ihn daraus wieder herausdrücken, immer wieder und wieder und wieder...
- Abhilfe: Mario hat eine Desired Position, die vom Game Level Layer geprüft wird und falls diese (nach collision detection) ok ist, updated der Game Level Layer die Position von Mario



Games

TMXTiledMap



Map hat

- hazards - Objekte die Mario schaden/töten
- walls - Hindernisse die Mario nicht durchdringen kann
- background - Eye candy

GameLevelLayer

- Braucht einen Pointer auf die Map

```
@interface GameLevelLayer() {  
    CCTMXTiledMap *map  
}
```

- Die Map muss geladen werden, ggf. Background (Himmel, etc.) setzen sofern nicht in der TileMap vorhanden

```
CCLayerColor *sky = [[CCLayerColor alloc] initWithColor:ccc4(100, 100, 250, 255)];  
[self addChild:sky];  
  
map = [[CCTMXTileMap alloc] initWithTMXFile:@"level.tmx"];  
[self addChild:map];
```

GameLevelLayer

- Braucht einen Player (Mario)

```
#import "Player.h"
```

...

```
Player *mario;
```

- Mario muss noch hinzugefügt werden in der init Methode

```
mario = [[Player alloc] initWithFile:@"mario_stand.png"];
mario.position = ccp(120, 60);
[map addChild:mario z:20];
```

Objekte in der realen Welt

- Statt eine komplexe State Machine zu erstellen, lieber an der realen Welt orientieren
- Gravity zieht Objekte immer nach unten
- Kräfte werden nicht ein-/ausgeschaltet. Eine Kraft wirkt auf einen Körper, das Momentum setzt sich fort bis eine andere Kraft auf den Körper wirkt
- Wenn Mario springt, wird die Schwerkraft nicht abgeschaltet, stattdessen wird eine Kraft angewendet, welche die Schwerkraft kurzzeitig überkommt
- Eine Kraft die einen Körper bewegt, wird von der Reibung "bekämpft"

- Wenn sich Mario bewegt (weil wir eine Kraft auf ihn anwenden, die ihn sich bewegen lässt), bleibt er nicht stehen wenn wir aufhören die Kraft auf ihn wirken zu lassen
- Er wird durch die Reibung verlangsamt und bleibt dann irgendwann stehen
- Das anwenden kumulativer Kräfte ist wichtig und ermöglicht dynamisches Verhalten (Steinboden vs. Eis, etc.)

- Anwendung ist deutlich einfacher, Objekte müssen nicht konstant nach ihrem Zustand abgefragt werden, stattdessen folgen die Objekte den Gesetzen der Physik

Game Level Layer

- Muss geupdated werden mit einigen Dingen...
 1. Initialisierung Velocity mit 0.0
 2. Erstellung von Gravity - Vector der Mario um x Pixel pro Sekunde zum Boden drückt
 3. Multiplikator (scale factor) um die Beschleunigung für jeden Time Step zu skalieren (variable Frame Rate führt zu konstanter Beschleunigung)
 4. Nach der Berechnung der Schwerkraft für diesen Time-Step, Addition zu aktuellen Velocity. Daraus folgt die Velocity für einen einzigen Time-Step (variable FPS...)
 5. Verwendung von ccpAdd für neue Position von Mario

Collision Detection

- Kann sehr einfach sein durch einfache Bounding Box oder komplex durch 3D Mesh
- Keep it simple!
- Um eine Kollision zu erkennen, muss eine Query auf die TileMap erfolgen (auf die Tiles die Mario umgeben)
- Danach schauen ob sich Mario's Bounding Box mit der Bounding Box des Tiles schneidet
 - `CGRectIntersectsRect` - gibt an ob sich zwei Rechtecke überschneiden
 - `CGRectIntersection` - gibt das überschneidende Rechteck zurück
- Die Bounding Box von Mario soll wahrscheinlich etwas kleiner sein, als sie tatsächlich ist
 - Warum? Weil Mario von Transparenz umgeben ist und sich die Kollision etwas überlappen soll

Get it up and running

- Game Level Layer braucht Methoden um Kollisionserkennung durchzuführen
 - Methode die Koordinaten der 8 umgebenden Tiles zurück gibt
 - Methode die feststellt ob eine Kollision mit einem der 8 Tiles besteht
 - Methode um die Kollision abzuarbeiten (inkl. Priorität)
- Hilfsmethoden
 - Berechnung der Tile-Position von Mario
 - Methode die Tile-Koordinaten in Rechteck für 2D OpenGL Koordinaten umrechnet

Games

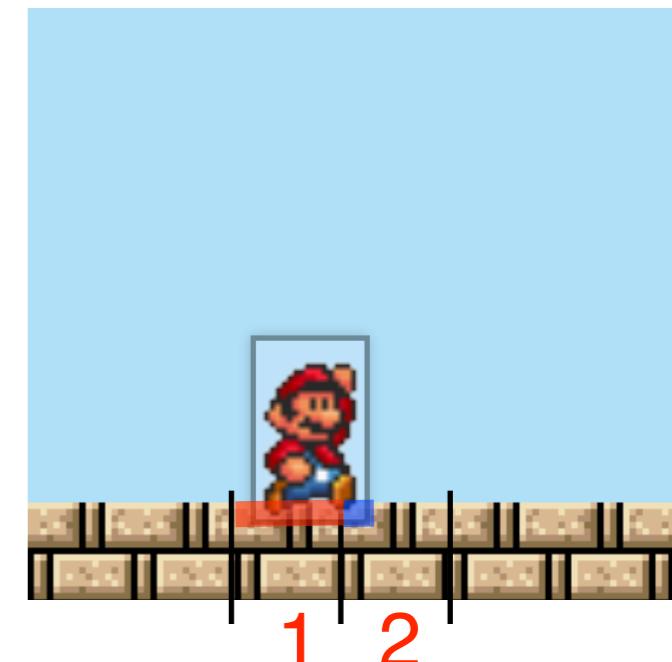
- ➊ Mario vs. hazards
 - ➊ Wenn Mario mit einem hazard Tile kollidiert, stirbt er
- ➋ Mario vs. walls
 - ➊ Wenn Mario mit einem wall layer kollidiert, muss weitere Bewegung in diese Richtung verhindert werden
- ➌ Mario vs. background
 - ➊ nothing here, move along...

Games

1. Empfang der Tile-Koordinaten für die Input Position (Mario Position)
2. Array erstellen welches Tile-Informationen beinhalten wird (als Return value)
3. 9-Loop (für 9 Tiles, wobei eigentlich nur 8 gebraucht werden sollten). Positionsberechnung der Tiles und speichern in tilePos
4. Ermitteln der GID für ein Tile an Position
5. Berechnung der 2D World Space Koordinaten für jedes Tile (CGRect) und speichern in NSDictionary. Collection von Dictionaries wird über das Array (s. 2) zurückgegeben
6. Mario Tile aus den Array entfernen und Tiles nach Priorität sortieren (erst adjazente Tiles)

1	2	3
4		5
6	7	8

5	2	6
3		4
7	1	8



Die Bearbeitung von Tile 1 im Bild (Kollision rot), führt automatisch zur Lösung der Kollision in Tile 2 (Kollision blau)

Games

- Um auf Walls Zugriff zu haben, müssen Sie dem Game Level Layer hinzugefügt werden

```
CCTMXLayer *walls; // Interface  
...  
walls = [map layerNamed:@"walls"]; // init  
...  
[self getSurroundingTilesAtPosition:player.position forLayer:walls]; // update
```

- Daran denken, dass Mario nur seine desired position setzt, nicht seine echte Position!

- Die echte Position wird vom Game Level Layer gesetzt und zwar erst nachdem auf alle Kollisionen aller Tiles geprüft wurde!

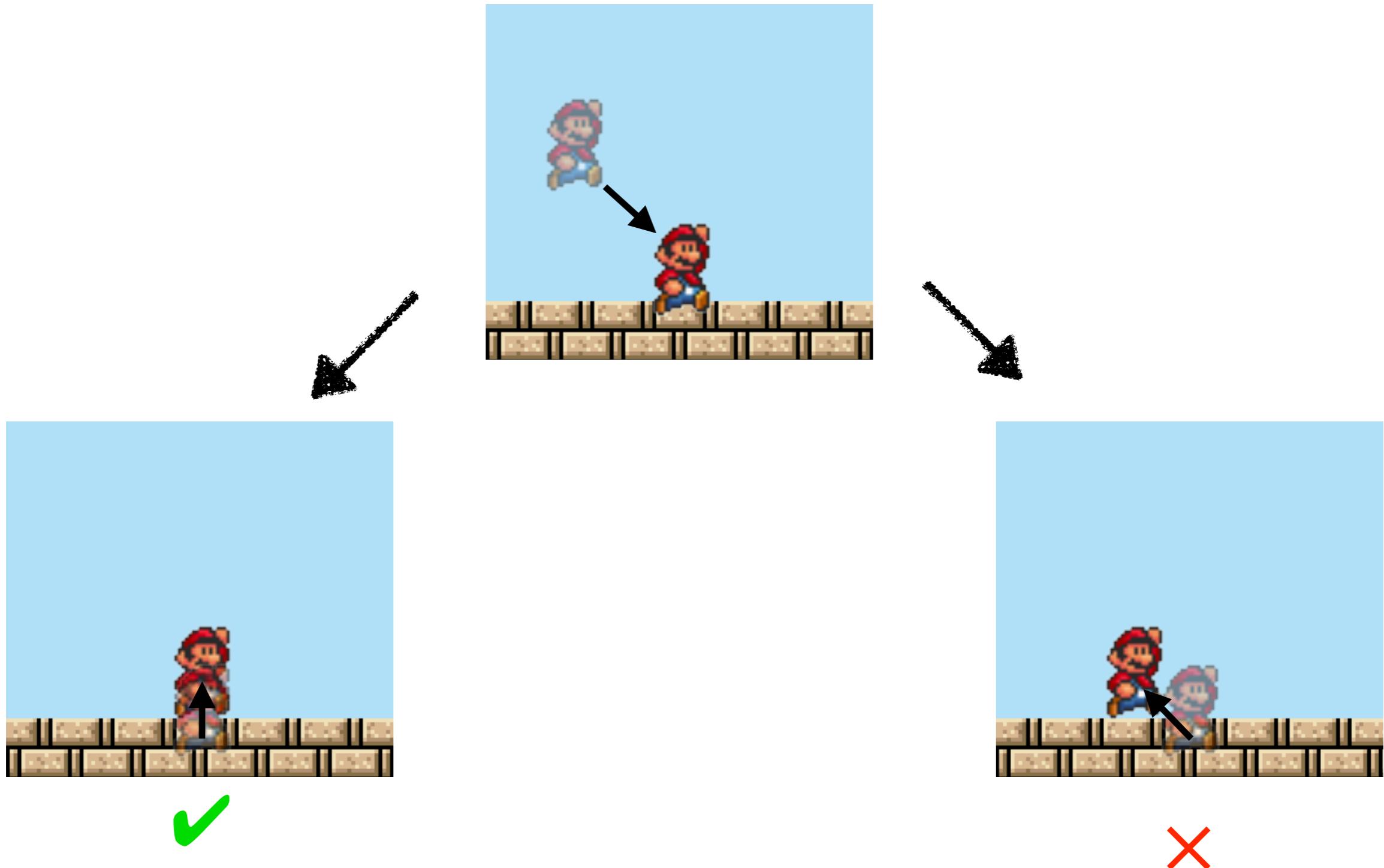
- Ansonsten setzt Mario seine Position vielleicht beliebig hin und her, springt dadurch im Level und landet vielleicht außerhalb der Map (Crash!!!)

Game Level Layer

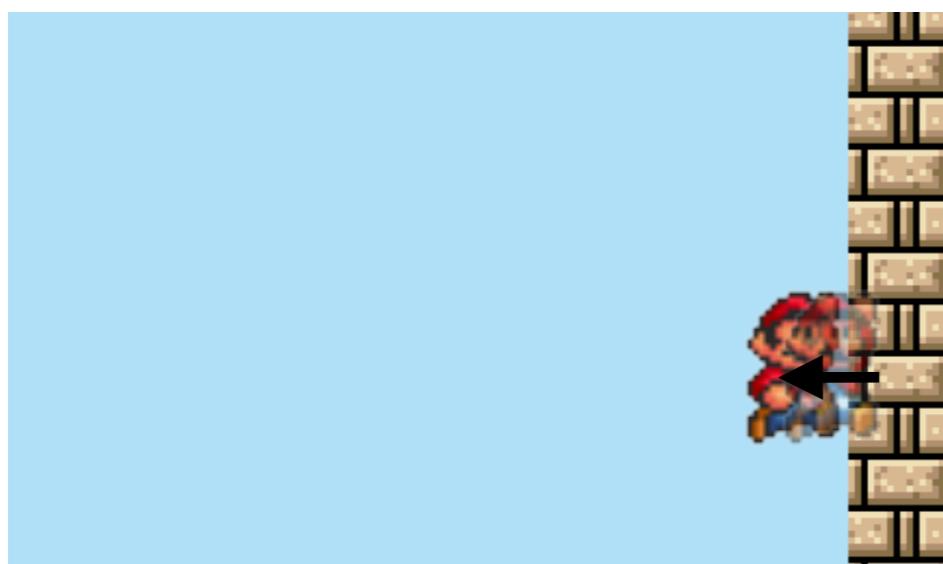
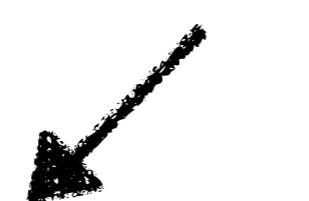
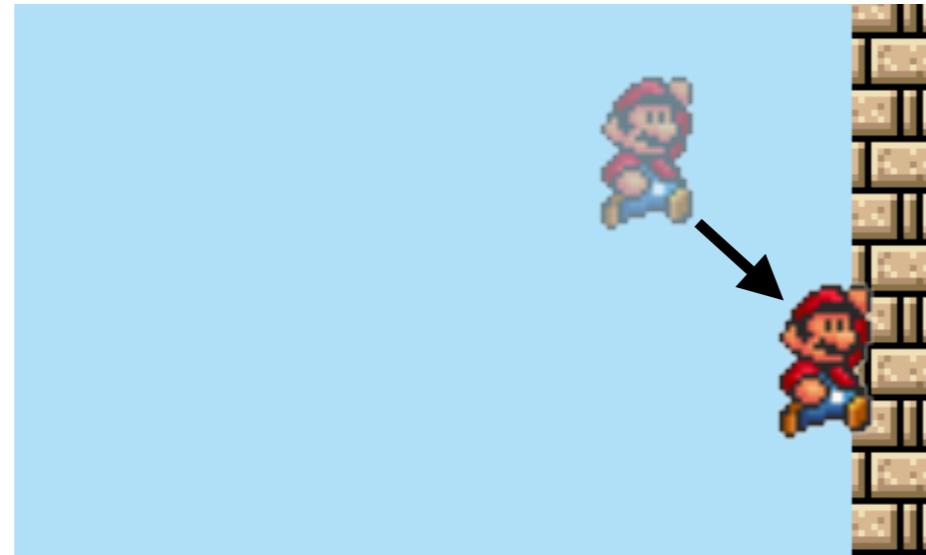
checkAndResolveCollisions fügt alles zusammen

1. Erhalten der umgebenden TileSets, Loop über alle Tiles, jede bemerkte Kollision behandeln (durch Veränderung des desiredPosition Attributes von Mario)
2. In jedem Loop, erhalten der Bounding Box für Mario. Wenn Mario durch eine Kollision bewegt wird, müssen die restlichen Tiles vielleicht nicht mehr geprüft werden!
3. GID erhalten aus Dictionary. Falls 0 (kein Tile an dieser Position), all done, move to next Tile
4. Falls ein Tile an der Position existiert, CGRect dafür erhalten. CGRect für Mario und Tile auf Kollision prüfen
5. Prüfen mittels CGRectIntersectsRect...

- Probleme bei der Kollisionserkennung



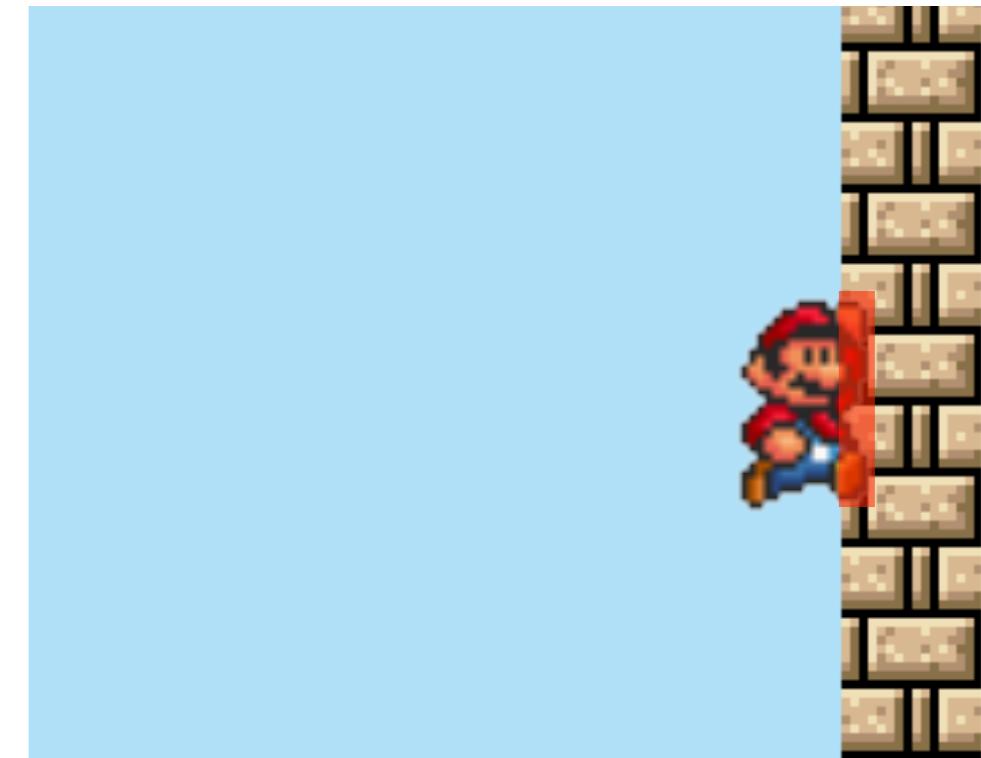
- Probleme bei der Kollisionserkennung



- Probleme bei der Kollisionserkennung



Breiter als hoch = vertikal



Höher als breit = horizontal

Games

6. Index des aktuellen Tiles erhalten, Index nutzen um Position zu erhalten. Adjazente Tiles individuell behandeln. Mario bewegen durch Addition/Subtraktion der Höhe/Breite des Collision Rectangles. Diagonale Tiles, Sonderfall wie auf den vorherigen Folien gezeigt
7. Beachten ob die Tiles über/unter Mario liegen, dementsprechend Addition oder Subtraktion wählen
8. Position von Mario setzen





• Mario is cheating... ?!?

- Mario wird vom Boden gestoppt, aber über die Zeit fällt er durch den Boden durch!
 - In jedem Frame wird Mario nach unten beschleunigt (durch die Schwerkraft)
 - Velocity wird irgendwann so groß, dass sie das Tile überkommt (Bewegung durch ein ganzes Tile in einem einzigen Frame)
 - Wenn eine Kollision behandelt wird, sollte die Velocity in dieser Dimension resetet werden. Mario bewegt sich nicht mehr, also sollte dies durch die Velocity widergespiegelt werden
 - Vorsicht! Wenn das nicht gemacht wird, kann das Spiel sich völlig unvorhersehbar verhalten
 - Innerhalb der CCSprite Subclass ein Property (`onGround`) dafür verwenden

Games

What's next?

- Mario kann noch gar nicht rennen und springen...
- Spikes, Feuer, Gegner, etc.
- Scoring, Winning, Losing, usw.
- Abgründe - aus dem Display fallen (Crash!)
- Hinzufügen von Audio/Sound
- und viel, viel mehr...

Ein paar Tools

- Tiled - Tile Editor: <http://www.mapeditor.org>
- Kobolt Kit - Open Source Sprite Kit Engine: <http://koboldkit.com>
- Cocos2d-x - <http://www.cocos2d-x.org>
- Natürlich Sprite Kite