

> TOPICONF WORKSHOP

> 25 MEI 2016



A GAME OF SWIFT

Thomas Dekker

Aron Kool

Kevin Rummler

Topicus Onderwijs

INHOUD

- > Introductie Swift
- > Uitleg basics Swift
- > Build a game [pt. 1]
- > Meer Swift
- > Build a game [pt. 2]
- > Afsluiting

INTRO

- > Geïntroduceerd in 2014
- > Vervanger van Objective-C
- > Inmiddels versie 2.2 (bijna 3)
- > Open source
- > Cross Platform (Mac, iOS, Linux, Windows?, Android?)

Gerucht: Google overweegt Swift te ondersteunen voor Android

Door Olaf van Miltenburg, donderdag 7 april 2016 21:10, 97 reacties • [Feedback](#)

Google, Facebook en Uber zouden overwegen Swift te gaan gebruiken. Google zou erover denken de door Apple ontwikkelde programmeertaal op lange termijn in te zetten, mede door de jarenlange strijd met Oracle over het gebruik van Java in Android.

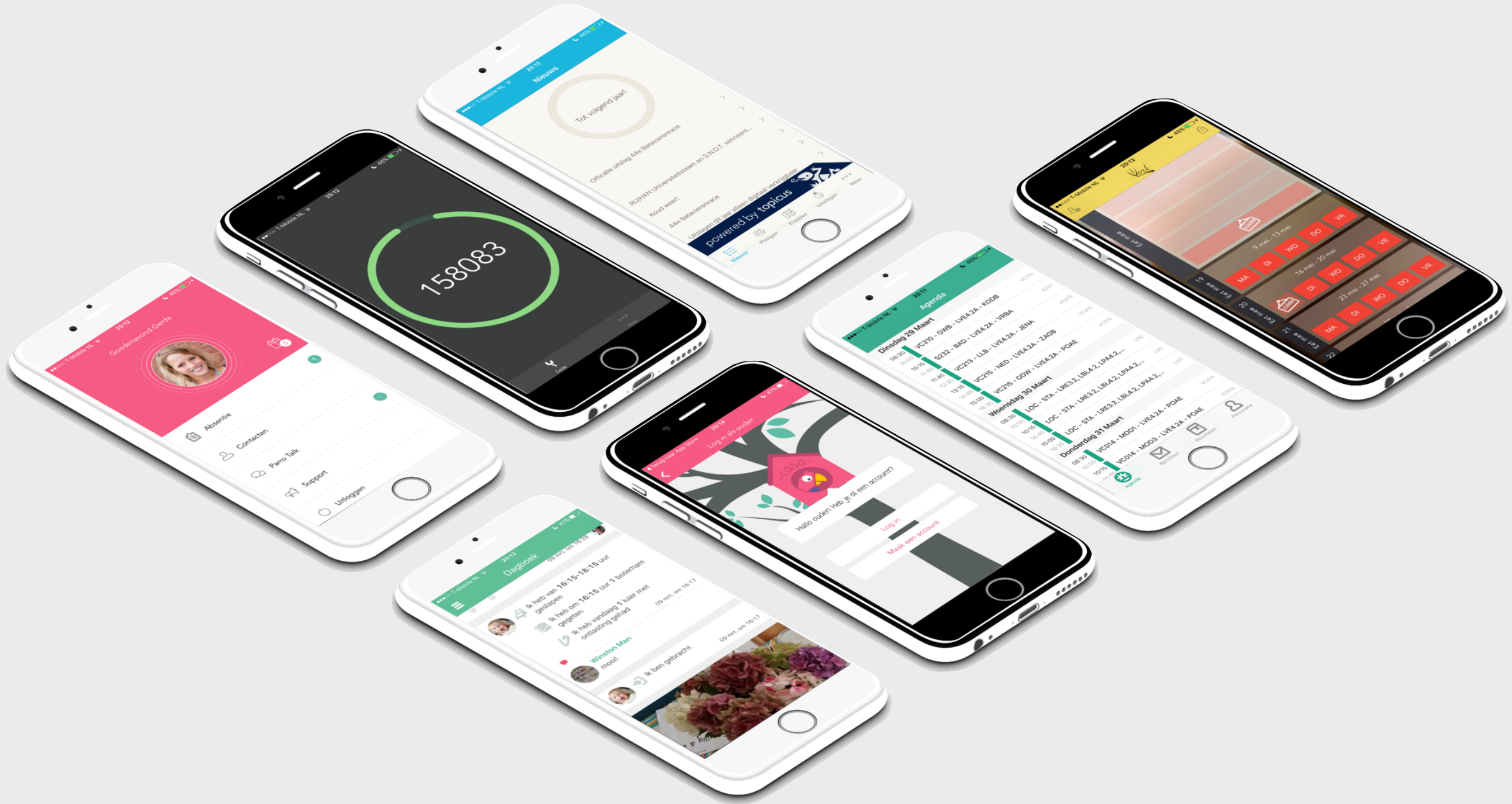
Aanvankelijk zou Swift Java dan ook nog niet vervangen, heeft The Next Web van bronnen [gehoord](#). Die bronnen spraken er wel van dat Swift een *first-class*-taal voor Android moet worden. Swift is ontwikkeld als alternatief voor Objective-C en vorig jaar opensource [gemaakt](#) door Apple. Uber en Lyft zouden volgens de site waarschijnlijk pas volgen als Google een besluit heeft genomen.

The Next Web tekent aan dat er veel moet gebeuren voordat Google Swift kan gebruiken voor Android. Api's en sdk's moeten compleet herschreven worden en er is een runtime nodig. Als gevolg van de strijd met Oracle is Google bij Android N al afgestapt van zijn implementatie van Java-libraries naar Oracles opensource OpenJDK. Overigens zou Google volgens het gerucht ook Kotlin nog niet afgeschoten hebben. Deze programmeertaal is makkelijker te implementeren voor Google, maar zou volgens het concern niet snel genoeg compilen.

Swift wint in rap tempo aan populariteit. Onder andere IBM maakt het mogelijk om met Swift applicaties voor clouddiensten te ontwikkelen en draaien.

Lee

Mobi



OBJECTIVE-C

- > Bridging
- > Versie met generics etc
- > Geen reden om niet naar Swift te gaan

BEÏNVLOED DOOR FUNCTIONELE PROGRAMMEERTALEN

> Zoals Haskell en Scala

STRONGLY TYPED

Met zoveel mogelijk type inference

OPTIONALS ALS ONDERDEEL VAN DE TAAL

Iets kan niet nil zijn tenzij expliciet
aangegeven

VEEL SYNTACTIC SUGAR

Om de code clean te houden

GEEN PUNTKOMMA'S

Zoals in Javascript,
maar hier werkt het wel

DIT ZORGT VOOR

- > Snelle ontwikkeling
- > Compacte code
- > Robuuste software

SWIFT KENT

Functions

Interfaces

Variabelen

Operators

Tuples

Classes

Collecties

Constanten

Control Flow

Optionals

Generics

MAAR ALLES IS
NET EVEN ANDERS

Want Apple?

SWIFT PT. 1

LET

```
let pizza = Pizza()
```


VAR

```
var pizzaKeuze = Pizza()  
let pizza = Pizza()  
pizzaKeuze = pizza
```

OPTIONALS

```
var pizzaKeuze : Pizza?  
let pizza = Pizza()  
pizzaKeuze = pizza
```

OPTIONALS

```
pizzaKeuze?.naam = "Gekozen pizza"
```

OPTIONALS

```
let naam = pizzaKeuze?.naam
```

UNWRAPPING

```
if let pizzaKeuze = pizzaKeuze {  
    //pizzaKeuze is niet meer optional  
}
```

UNWRAPPING

```
if let vispizza = vispizzas.first {  
    //De eerste de beste vispizza  
}
```

FORCE UNWRAPPING

```
let unwrapped = pizzaKeuze!
```

UNWRAPPING

```
if let pizza = vispizzas.first, firstTopping = pizza.toppings.first {  
    //De eerste de beste vispizza en daarvan de eerste topping  
}
```


UNWRAPPING

```
if let firstTopping = visPizzas.first?.toppings.first {  
}
```

CASTING

```
if let topping = topping as? ExtraTopping, extraPrijs = topping.prijs {  
    //bijbetalen  
}
```

??

```
pizzaKeuze = visPizzas.first != nil ? visPizzas.first : anderePizza
```

??

```
pizzaKeuze = visPizzas.first ?? anderePizza
```

??

```
pizzaKeuze = vispizzas.first ?? vleespizzas.first ?? vegetarischePizzas.first ?? anderePizza
```

PRAKTIJK

```
let aanduiding = roepnaam ?? voornaam ?? voorletters ?? "Onbekend"
```

PROTOCOLS

```
protocol Bestelbaar {  
    var prijs : Double { get }  
    func bestel()  
}
```

CLASSES

```
class Pizza : Entiteit, Bestelbaar {  
    var prijs : Double  
    let naam : String?  
  
    init(naam : String, prijs : Double) {  
        self.naam = naam  
        self.prijs = prijs  
    }  
  
    func bestel(){  
  
    }  
  
    func naamHoofdletters() -> String? {  
        return naam?.capitalizedString  
    }  
}
```


STRUCTS

```
struct Pizza {  
    let naam : String  
    let soort : PizzaSoort  
    let toppings : [Topping]  
}
```

STRUCTS

```
struct HttpHeaders {  
    static let Accept = "Accept"  
    static let ContentType = "Content-Type"  
}
```

STRUCTS

- > Dataobjecten
 - > IBAN
 - > Postcode
 - > Size
- > Member wise constructor met alle variabelen
- > Geen overerving
- > Lazy copy per referentie

STRUCTS

```
struct Pizza {  
    var naam : String  
  
    mutating func resetNaam() {  
        naam = "Standaard Pizza"  
    }  
}
```

STRUCTS

```
let margherita = Pizza(naam: "Margherita")
var standaardPizza = margherita
// 1 instantie van Pizza
standaardPizza.resetNaam()
// 2 instanties van Pizza
print(margherita.naam) // Margherita
print(standaardPizza.naam) // Standaard Pizza
```

ENUMS

```
enum PizzaSoort {  
    case Vis  
    case Vlees  
    case Vegetarisch  
}
```

ENUMS

```
enum PizzaSoort : String {  
    case Vis = "Vispizza"  
    case Vlees = "Vleespizza"  
    case Vegetarisch = "Vegetarische pizza"  
}
```

ENUMS

```
let soortNaam = "Vleespizza"
if let soort = PizzaSoort(rawValue: soortNaam) {
    switch soort {
    case .Vis: visMethode()
    case .Vlees: vleesMethode()
    case .Vegetarisch: vegetarischeMethode()
    }
}
```


ENUMS VALUES

```
enum PizzaSoort {  
    case VisPizza(Vis)  
    case VleesPizza(Vlees)  
    case VegetarischePizza(Groente)  
}
```

ENUM VALUES

```
let tonno = Pizza(naam: "Tonno")
tonno.soort = .VisPizza(Vis(naam: "Tonijn"))
let salmon = Pizza(naam: "Salmon")
salmon.soort = .VisPizza(Vis(naam: "Zalm"))
```

ENUM VALUES

```
switch soort {  
case .VisPizza(let vis): return vis.naam  
case .VleesPizza(let vlees): return vlees.naam  
case .VegetarischePizza(let groente): return groente.naam  
}
```

PRAKTIJK

```
public enum Optional<Wrapped> {  
    case None  
    case Some(Wrapped)  
}
```

COMPUTED PROPERTIES

```
var omschrijving : String {  
    return "\(self.naam) met \(self.toppings)"  
}
```

ARRAYS

```
let visPizzas = [Pizza(naam: "Tonno"), Pizza(naam: "Salmon")]
```

ARRAYS

```
var pizzas = [Pizza]()  
addVispizzas(pizzas)  
addVleespizzas(pizzas)  
addVegetarischePizzas(pizzas)  
let leeg = pizzas.isEmpty // true
```

ARRAYS

```
let eerste = pizzas.first // optional  
let tweede = pizzas[1] // niet optional  
let tweeTotEnMetVijf = pizzas[1..<5]
```


ARRAYS

```
let nietVegetarisch = visPizzas + vleesPizzas
```



I SMILE TO HIDE
HOW
COMPLETELY
OVERWHELMED
I AM.

GAME PART 1

> TopiPizza

> Basis Swift

> Model bouwen

> Compilerend spel zonder actie



Welcome to Xcode

Version 7.3 (7D175)



Get started with a playground

Explore new ideas quickly and easily.



Create a new Xcode project

Start building a new iPhone, iPad or Mac application.



Check out an existing project

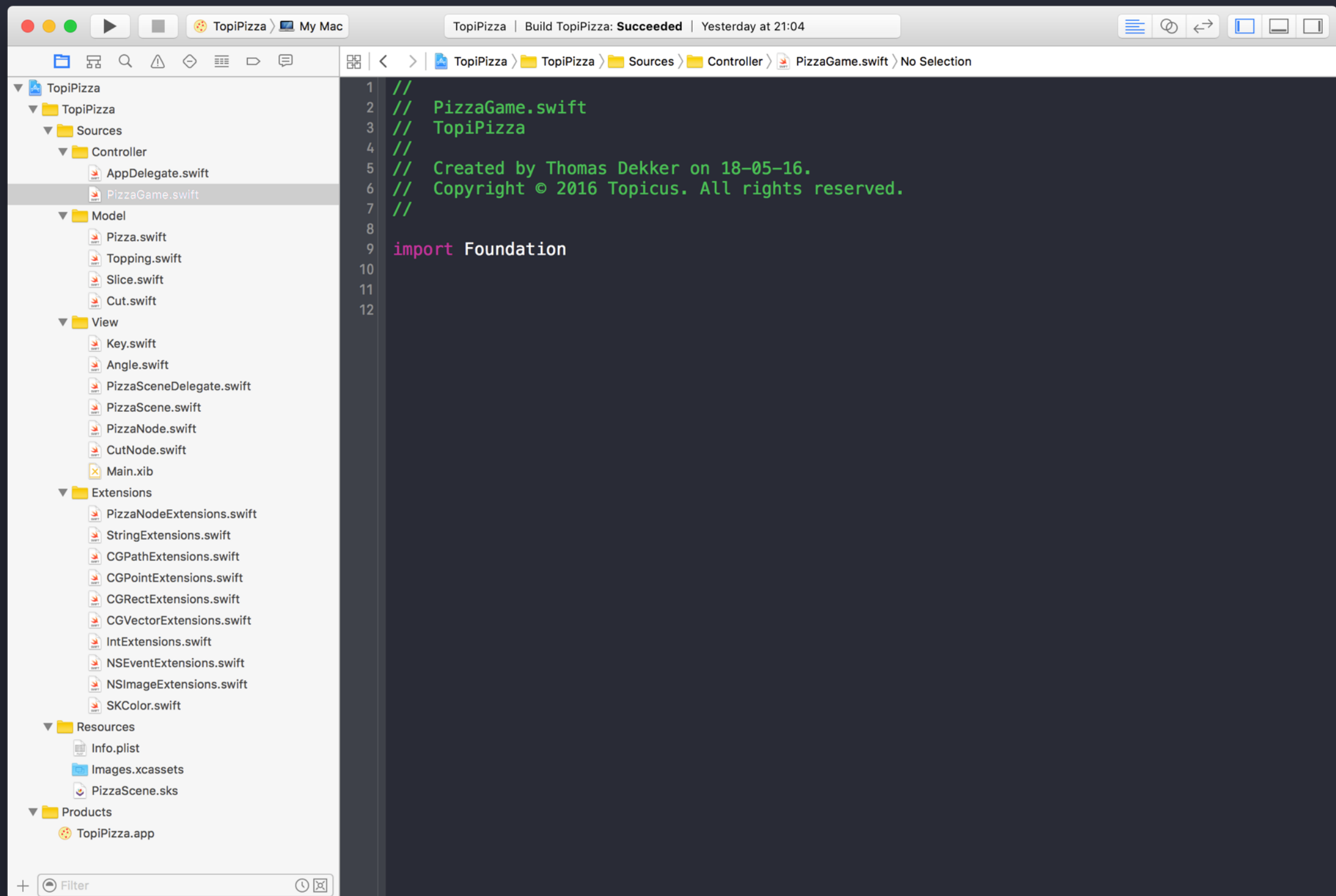
Start working on something from an SCM repository.

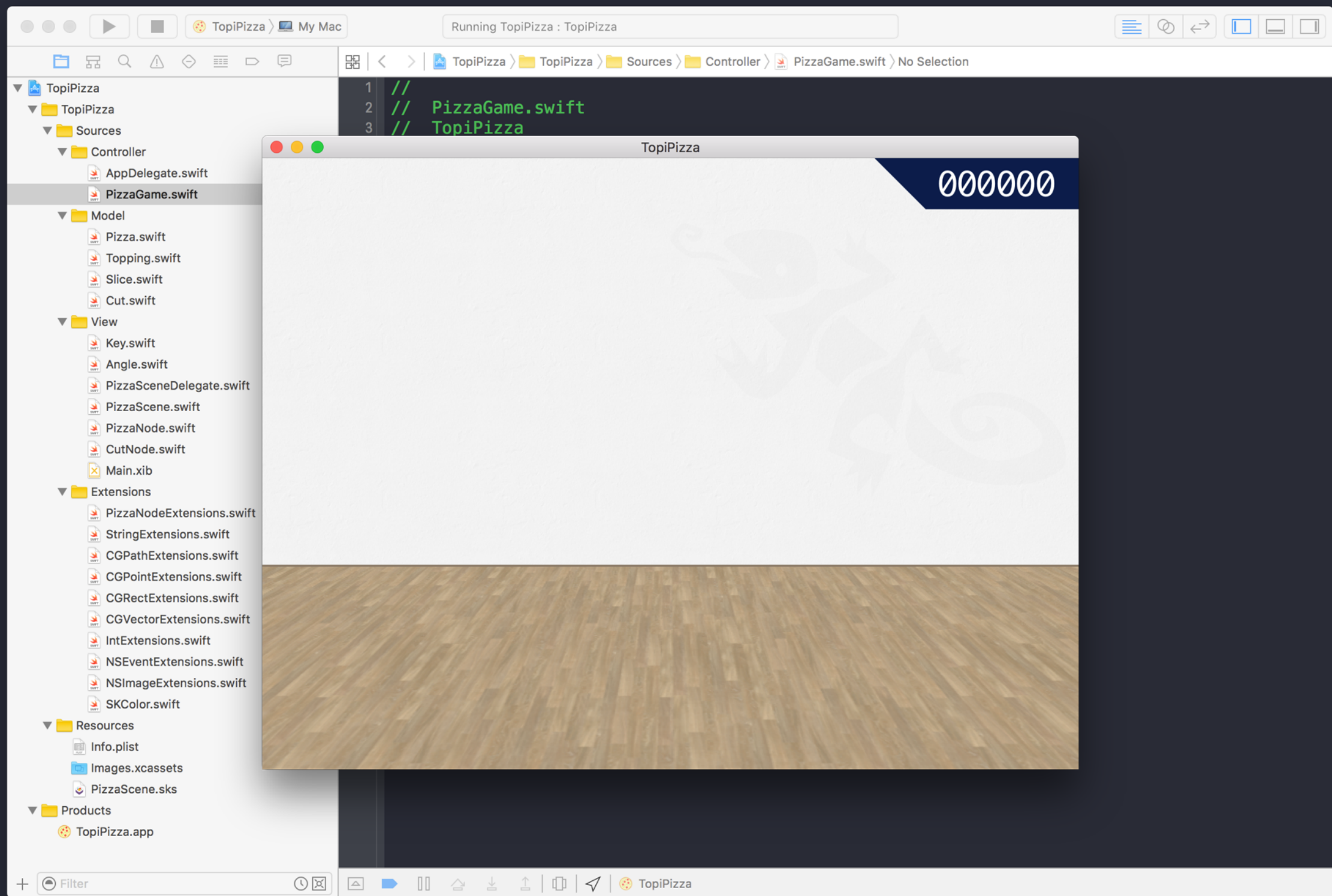


TopiPizza

~/Development/Topicus

Open another project...





SOURCES



<https://github.com/topicusonderwijs/TopiPizza>

SWIFT PT. 2

DICTIONARIES

```
var pizzas = [String : Pizza]()  
pizzas["Margherita"] = Pizza(naam: "Margherita")
```

CLOSURES

```
let visPizzas = pizzas.filter { (pizza) -> Bool in  
    return pizza.soort == .VisPizza  
}
```

CLOSURES

```
let visPizzas = pizzas.filter { return $0.soort == .VisPizza }
```

CLOSURES

```
let visPizzas = pizzas.filter { $0.soort == .VisPizza }
```

CLOSURES

```
let pizzaNamen = pizzas.map({ return $0.naam })
```

CLOSURES

```
let alfabetisch = pizzaNamen.sort { (pizza1, pizza2) -> Bool in  
    return pizza1 < pizza2  
}
```

CLOSURES

```
let alfabetisch = pizzaNamen.sort { $0 < $1 }
```

CLOSURES

```
let alfabetisch = pizzaNamen.sort(<)
```


EXTENSIONS

```
extension Pizza {  
    func heeftTopping(topping : Topping) -> Bool {  
        return toppings.contains({ return $0.id == topping.id})  
    }  
}
```

PRAKTIJK

```
extension String {  
    func firstCharacterUppercase() -> String {  
        let uppercase = self  
        //TODO: daadwerkelijke implementatie  
        return uppercase  
    }  
}
```

EXTENSIONS

```
extension Array where Element:Topping {  
    var asString : String {  
        return map({ return $0.naam }).joinWithSeparator(", ")  
    }  
}
```

PRAKTIJK

```
extension SequenceType where Generator.Element == String {  
    public func joinWithSeparator(separator: String) -> String  
}
```

OPERATORS

```
func ==(lhs: Topping, rhs: Topping) -> Bool {  
    return lhs.id == rhs.id  
}
```

```
func !=(lhs: Topping, rhs: Topping) -> Bool {  
    return lhs.id != rhs.id  
}
```

MOGELIJKHEDEN

```
let toekomst = (5.days + 3.months).fromNow  
let verleden = (2.days + 3.years).ago
```

TUPLES

```
func getBestelling() -> (eten: [Pizza], drinken: [Drank]) {  
    let pizzas = [Pizza(naam: "Margherita"), Pizza(naam: "Tonno") ]  
    let dranken = [Drank(naam: "Gifkikker"), Drank(naam: "Cola")]  
    return (eten: pizzas, drinken: dranken)  
}
```

TUPLES

```
let bestelling = getBestelling()  
print(bestelling.eten)  
print(bestelling.drinken)
```


TUPLES

```
var pizzas = [String : Pizza]()  
pizzas["Margherita"] = Pizza(naam: "Margherita")  
for entry in pizzas{  
    print(entry.0) // key  
    print(entry.1) // value  
}
```

TUPLES


```
var pizzas = [String : Pizza]()  
pizzas["Margherita"] = Pizza(naam: "Margherita")  
for (key, value) in pizzas{  
    print(key) // key  
    print(value) // value  
}
```

FOR

```
for index in 0.. $<10$  {  
}
```

FOR

```
for _ in 0..<10 {  
    //Doe iets  
}
```



```
if let _ = pizzaKeuze {  
    //Er is een keuze gemaakt  
}
```

TRY

```
do {  
    try bestel(pizza)  
} catch _ {  
  
}
```

TRY

```
do {  
    try bestel(pizza)  
} catch {  
  
}
```

TRY

```
func bestel(pizza : Pizza) throws {  
    throw NotImplemented() // Implementatie van ErrorType  
}
```


TRY

```
let pizzas = try? fetchFromServer()
```

PROPERTY OBSERVERS

```
var pizzaKeuze : Pizza? {  
    didSet {  
        print("\n(pizzaKeuze) wordt \n(newValue)")  
    }  
    didSet {  
        print("\n(oldValue) werd \n(pizzaKeuze)")  
        //Update UI?  
    }  
}
```

LAZY

```
var pizzaService : PizzaService {  
    if let pizzaService = internalPizzaService {  
  
        return pizzaService  
    } else {  
  
        let service = PizzaService()  
        service.url = "https://localhost/api/"  
        service.method = "GET"  
        internalPizzaService = service  
        return service  
    }  
}
```

LAZY

```
lazy var pizzaService : PizzaService = {  
    let service = PizzaService()  
    service.url = "https://localhost/api/"  
    service.method = "GET"  
    return service  
}()
```

GENERIC

```
class Fetcher<T:Topping> {  
    func get() -> [T] {  
        return []  
    }  
  
    func create() -> T {  
        return T()  
    }  
}
```

WHERE

```
if let pizzaKeuze = pizzaKeuze where !pizzaKeuze.toppings.isEmpty {  
}
```

WHERE

```
switch pizza.naam {  
case let x where x.hasPrefix("Kinder"): bestelKleinePizza()  
case let x where x.hasSuffix("XL"): bestelGrotePizza()  
default: bestelNormalePizza()  
}
```

??

```
let vragen : [Vraag]?
```


GAME PT.2

- > Gooien van pizza's
- > Snijden van pizza's
- > Berekenen en tonen van score
- > Spelen van het verslavende spel
- > Bonus?

MEER SWIFT?

The Swift Programming Language

(eBook van Apple)

Coding Together

(Stanford op iTunes University)