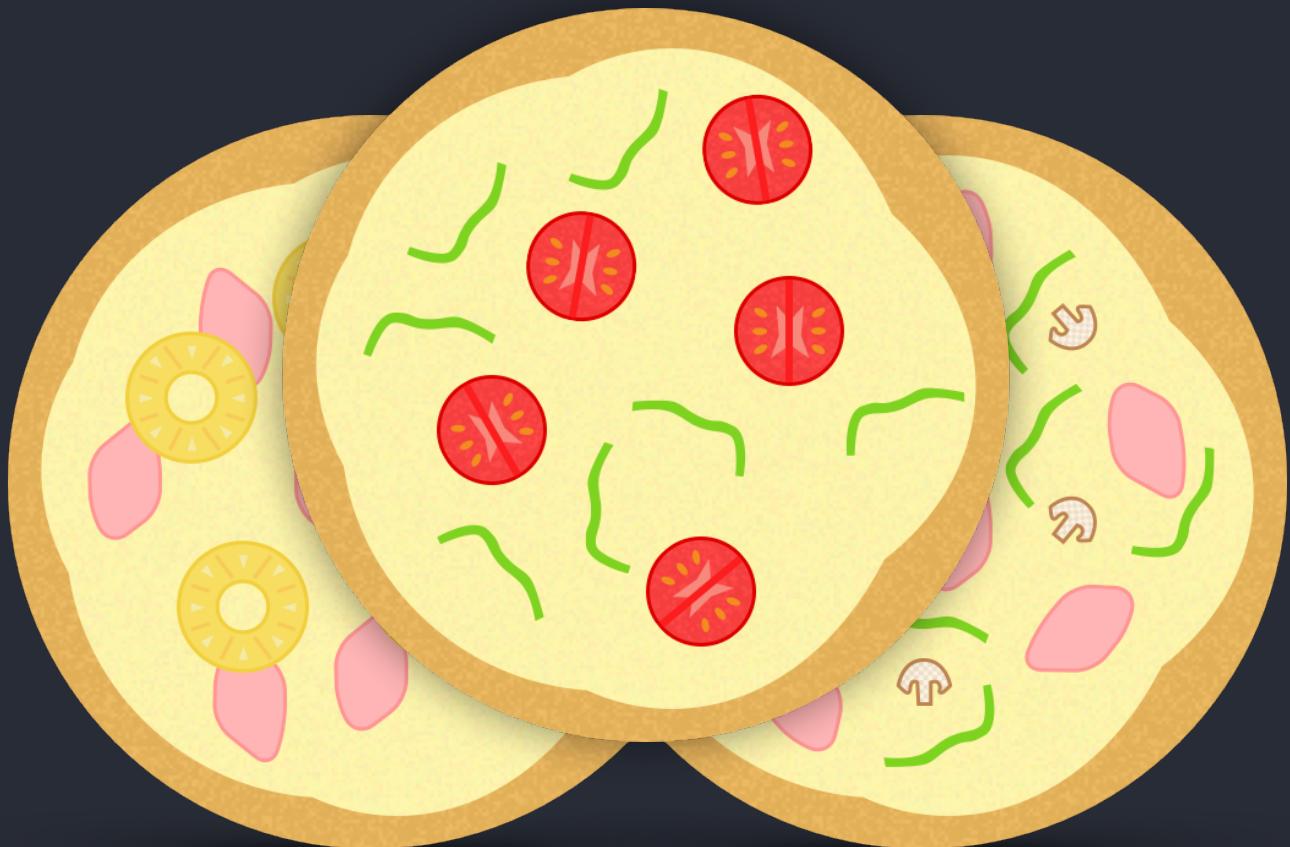


TopiPizza

Opdracht



A Game Of Swift

TopiConf Workshop
25 mei 2016

Thomas Dekker

Aron Kool

Kevin Rummel

Topicus Onderwijs

Beschrijving

In deze workshop gaan we een spel maken genaamd TopiPizza.

In een ruimte (die doet denken aan de extreem lege woonkamer van Steve Jobs) worden pizza's omhoog gegooid. Met de muis maakt de speler een snijlijn waardoor de pizza's in puntjes gesneden worden.

Het doel in dit simpele, maar verslavende spel is om pizza's te snijden. Het liefst zo veel mogelijk tegelijk. Elke gesneden pizzapunt levert punten op. Een combinatie van pizza's snijden binnen een seconde levert een bonus (*combo*) op.



Stel gerust vragen tijdens het maken van de opdrachten. Vergeet ook niet dat er in de slides een behoorlijk aantal codevoorbeelden staan.

Model

Het model voor het spel bestaat uit een klein aantal entiteiten.

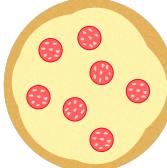
Topping

Er zijn zes verschillende toppings voor op een pizza. Elke soort topping is een vast aantal punten waard. De tabel hieronder bevat hiervan een overzicht.

Topping	Benaming	Punten
	Ham	3
	Mushroom	1
	Pepper	2
	Pineapple	2
	Salami	3
	Tomato	1

Pizza

Er zijn een aantal pizzasoorten. Elke soort heeft een of meer toppings. Bovendien heeft elke soort een afbeelding die met een naam wordt aangegeven. In de onderstaande tabel staat een overzicht.

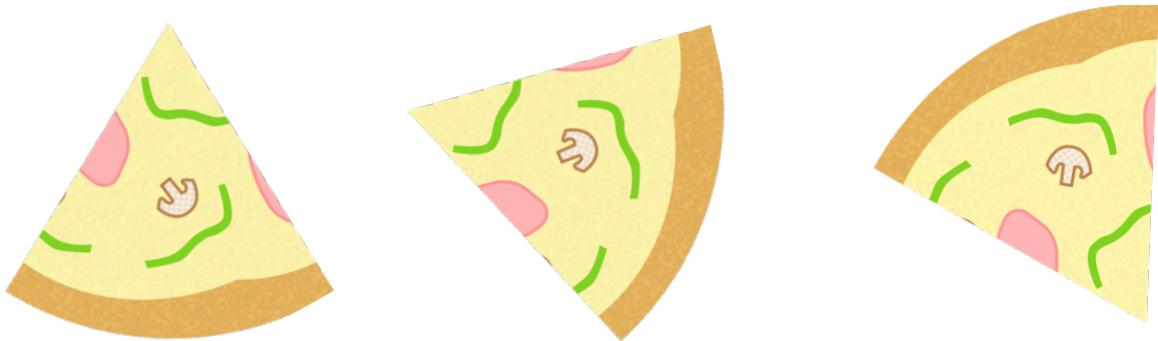
Pizza	Toppings	Afbeeldingsnaam	Voorkomen
 Tomaat	Tomaat	TomatoPizza	Vaak
 Paprika	Tomaat, Paprika	PepperPizza	Regelmatig
 Salami	Salami	SalamiPizza	Vaak
 Champignon	Champignon, Paprika, Ham	MushroomPizza	Vaak
 Hawaii	Ananas, Ham	HawaiianPizza	Regelmatig
 Verbrand		BurntPizza	Weinig

Het aantal punten dat een pizza waard is wordt bepaald door de toppings die er op zitten. De som van de punten van de toppings vermenigvuldigd met 10 is het aantal punten dat een pizza waard is. Een verbrande pizza is altijd -100 punten waard.

Slice

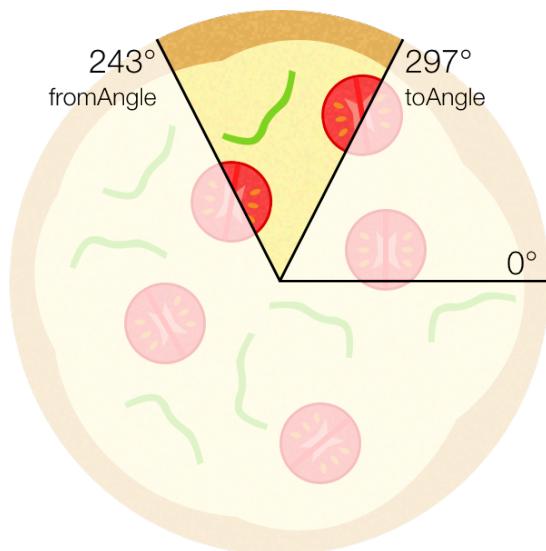
Als een pizza eenmaal gesneden is, valt deze uiteen in twee pizzapunten, vanaf nu *slices* genoemd.

Een slice wordt gedefinieerd door een begin- en eindhoek ten opzichte van de volledige pizza. Een slice kan zelf ook weer gesneden worden in twee slices.



Een slice heeft dus drie eigenschappen:

- de pizzasoort (**Pizza**) waar de slice van gemaakt is
- de beginhoek (**fromAngle**)
- de eindhoek (**toAngle**)



Opdrachten

Het doel van de workshop is het afmaken van de implementatie voor TopiPizza. De grafische elementen zijn platform-specifiek en zijn daarom al gegeven.

De volgende bestanden moeten voor de opdrachten worden aangepast:

 Controller

 PizzaGame.swift

 Model

 Pizza.swift

 Topping.swift

 Slice.swift

 Extensions

 PizzaNodeExtensions.swift

 StringExtensions.swift

Alle andere bestanden in de workspace hoeven niet worden aangepast, op hier en daar commentaar verwijderen na. Uiteraard is bekijken hoe de implementatie werkt prima.

Deel 1

Dit eerste deel van de workshop behandelt voornamelijk de basis van Swift en het model van het spel. Het resultaat is een compilerend spel waar nog geen actie in plaatsvindt.

- ➡ Clone het project van TopiPizza.

De benodigde bestanden zijn te vinden op GitHub:

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

- ➡ Open de workspace van TopiPizza.

In principe is Xcode redelijk goed te begrijpen. In de zogeheten *source list* staan alle bestanden voor de sources en resources. Het draaien van een target kan met de *run*-knop (▶). Het stoppen van een target gebeurt met de stop-knop (■).

- ➡ Implementeer de verschillende modelentiteiten.

Zie ook de beschrijving van het model. Implementeer het volgende:

- **Pizza** (**enum**)
- **Topping** (**enum**)
- **Slice** (**struct**)

De **Pizza** moet onder andere een aantal *computed properties* krijgen:

- **topping** : De toppings die op de pizza zitten.
- **image** (**String**) : De naam van het plaatje voor de pizza.
- **score** (**Int**) : Het aantal punten dat het snijden van de pizza geeft, op basis van **topping**.

Maak bovendien een statische **random** functie die een willekeurige **Pizza** meegeeft. Houdt er eventueel rekening mee dat sommige pizza's minder vaak voorkomen in het spel. Gebruik hiervoor de meegeleverde **Int.random** functie.

De **Slice** moet onder andere een aantal *stored properties* krijgen:

- **pizza** : De pizzasoort (**Pizza**) van de slice.
- **fromAngle** : De beginhoek van de slice.
- **toAngle** : De endhoek van de slice.

Gebruik voor de hoeken de meegeleverde **Angle** (**enum**). Deze heeft *enum values* om onafhankelijk te zijn van of er gerekend wordt met graden of radialen.

- ➡ Verwijder commentaar.

Nu het model is geïmplementeerd kunnen er een aantal regels in de code actief worden gemaakt.

In de **PizzaSceneDelegate** op regel 31 staat een declaratie voor een functie die buiten commentaar gezet moet worden. Deze declaratie kon niet bestaan omdat **Slice** nog niet bestond.

Verwijder vervolgens regel 114 in **PizzaNode** en zet de regel 115 buiten commentaar. Op die manier wordt de **image** property van **Pizza** in gebruik genomen.

- ➡ Maak de entiteit voor de spellogica.

Maak de **PizzaGame** (**class**) en laat deze **PizzaSceneDelegate** (**protocol**) implementeren. De beschrijving van welke functie wanneer wordt uitgevoerd staat in het protocol zelf.

Merk op dat de inhoud voor de logica pas in het tweede deel aan bod komt.

- ➡ Verwijder commentaar.

De **PizzaGame** moet vervolgens als **delegate** van de **PizzaScene** worden toegewezen. Dit is voor gedaan in de **AppDelegate** op regel 20 en 31. Verwijder hier het commentaar om de regels mee te compileren.

- ➡ Draai de applicatie om te verifiëren dat er geen compilatiefouten zijn.

Het spel zou nu moeten tonen. Het is nu mogelijk om sneden te maken, maar er verschijnen nog geen pizza's. Dit komt in het volgende deel.

Deel 2

Het doel van het tweede deel van de workshop is het daadwerkelijk snijden van de pizza's en het bepalen van de score die de speler heeft gehaald.

- ➡ Maak een extension voor **PizzaNode**.

Voeg een *computed property* toe genaamd **slice** die een **Slice?** geeft als resultaat. Zet de node om in een **Slice** als deze een slice is (**isSlice** in **PizzaNode**) en anders is de property **nil**.

Deze wordt gebruikt door de scene om door te geven aan de delegate. Deze kan vervolgens de pizza en eventueel de slices omzetten in punten.

- ➡ Verwijder commentaar.

Verwijder opnieuw commentaar, dit keer in **PizzaScene**. Regel 131 tot en met 148 staat in commentaar omdat het gebruik maakt van de **slice** property in **PizzaNode**.

- ➡ Implementeer de spellogica voor het gooien van pizza's.

Het vlees van het spel moet nu worden gemaakt.

De functies in **PizzaGame** worden vanaf nu allemaal aangeroepen door de **scene**. Implementeer het tweeluik dat nodig is om het spel te laten draaien. Eerst moeten de pizza's worden gegooid. Vervolgens moeten de gesneden pizza's omgezet worden in punten.

Implementeer met name eerst de **scene(:didUpdateTime:)** functie. Roep de **throwPizza** functie van de **scene** hierin aan om een pizza te gooien. Gebruik de **time** property van **scene** om de verlopen tijd in de gaten te houden. Voeg ook een aantal *properties* aan **PizzaGame** toe om de staat bij te houden.

Het is de bedoeling dat er een aantal pizza's tegelijk gegooid worden met intervallen van een willekeurig aantal seconden tussen verschillende worpen. Gebruik hiervoor de **Int.random** functie die is meegeleverd.

Het gooien verloopt in principe als volgt:

- Gooi 1 **Pizza.random()**
- Wacht **random(500, 4000)** milliseconden
- Gooi 2 **Pizza.random()**
- Wacht **random(500, 4000)** milliseconden
- Gooi 3 **Pizza.random()**
- (...)
- Gooi 8 **Pizza.random()**
- Wacht **random(500, 4000)** milliseconden
- Gooi 1 **Pizza.random()**
- (...)

Om de implementatie hiervan te versimpelen kun je altijd gewoon een enkele pizza gooien. Let wel dat je dan geen *combo* kan implementeren.

- ➡ Implementeer de spellogica voor het bepalen van de score.

Bij het starten van het spel zie je nu de pizza's de lucht in gaan. Het snijden werkt ook. Er worden alleen nog geen punten gerekend.

Implementeer de `scene(:didCutPizza:intoSlices:)` functie in `PizzaGame`. Hierin moeten de punten van een enkele snede te worden bepaald. Gebruik hiervoor de `score` property van `pizza`. De slices worden voor nu nog niet meegenomen in de score.

Stel de score in de `scene` in door de `score` property hiervan in te stellen. Dit is het totaal aantal behaalde punten dat door `PizzaGame` moet worden bijgehouden.

- ➡ Implementeer een combo voor de score.

Het is de bedoeling dat de vermenigvuldiging van de score elke keer wordt verdubbeld als er binnen een seconde nog een snede is. Een mogelijk verloop is als volgt:

Tijd	Multiplier	<code>pizza.score × Multiplier</code>	Score
500	1	20×1	$0 + (20 \times 1) = 20$
600	2	30×2	$20 + (30 \times 2) = 80$
800	4	20×4	$80 + (20 \times 4) = 160$
1100	8	30×8	$160 + (30 \times 8) = 400$
3000	1	-100×1	$400 + (-100 \times 1) = 300$

Bij een continue combo loopt de score behoorlijk op. Maar als er een verbrande pizza gesneden wordt tijdens deze combo loopt de score net zo hard weer terug.

Implementeer dit in de `scene(:didCutPizza:intoSlices:)` functie in `PizzaGame`.

- ➡ Maak een `extension` voor `String`

Zorg ervoor dat het aantal punten wat in beeld staat altijd met `0` gevuld wordt tot een maximum van 6 tekens. Maak hiervoor een functie in een `extension` van `String` die *left padding* verzorgt. Gebruik deze in de `PizzaGame`.

000480

Bonus

Het spel kan uiteraard naar eigen hartenlust worden aangepast. Hieronder een aantal ideeën.

- ➡ Verhoog de score op basis van hoe klein een slice is (kleiner = moeilijker = meer punten).
- ➡ Zorg ervoor dat de `random` functie in **Pizza** rekening houdt met het voorkomen van de pizza
- ➡ Houd een high-score bij, bijvoorbeeld voor de hoogst behaalde combo.
- ➡ Houd levens bij die vervallen als er een verbrande pizza gesneden wordt.
- ➡ Genereer een pizza met random topping en idem afbeelding.
- ➡ Behaal een score van 999999 met de officiële regels

Uiteraard kun je nog veel meer doen. Indien wat hulp nodig hebt, helpen we je graag.

