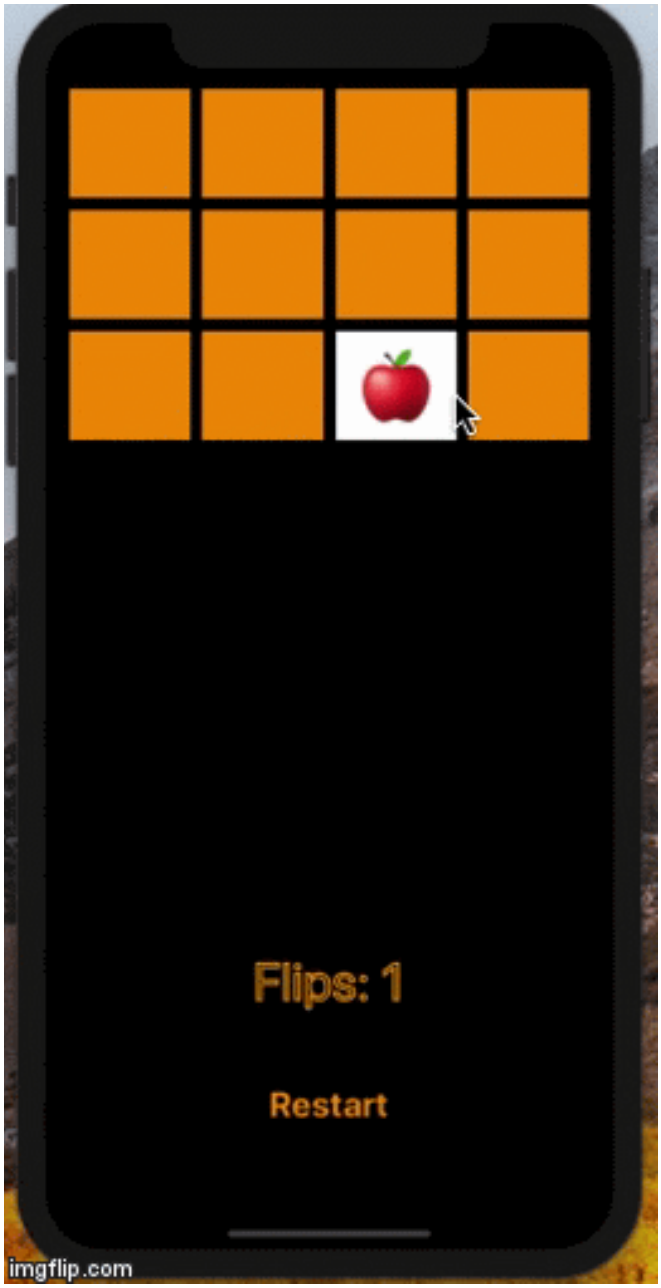
Carthago iOS workshop

Mobile concentration app (iOS)



Inhoudsopgave

Handige informatie

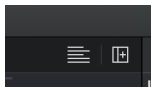
We gaan er tijdens de workshop vanuit dat deelnemers weinig tot geen ervaring hebben met iOS ontwikkeling. Om ervoor te zorgen dat je niet vast komt te zitten tijdens de workshop, hebben we het geheel opgedeeld in 5 projecten die met implementatie overeenkomen met alle voorgaande stappen (in stap 4 zijn dus bijvoorbeeld stappen 1, 2 en 3 geïmplementeerd).

Voor de mensen het eerst graag zonder stappenplan willen proberen hebben we per stap ook een beknopte beschrijving (met screenshot van het resultaat) opgezet.

Let op: de stapsgewijs opgebouwde repositories gaan er vanuit dat handleidingen gebruikt, dus doe dit alleen als je enige ervaring hebt met de swift syntax, storyboards en Xcode, we kunnen namelijk in mindere mate begeleiden omdat er hierbij afwijkingen op de normale code ontstaan waarbij we bij vragen moeten debuggen.

Github repository:
https://github.com/luuk-harmeling/CarthagoIOSWorkshop.git

Items koppelen door middel van drag-en-drop:
Vanuit het storyboard kan je een items koppelen aan de code met de volgende stappen:



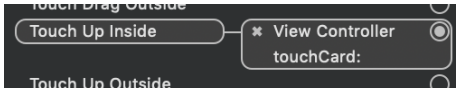
1. Zorg dat je editor en het storyboard beide open hebt, dit kan je doen met de knop boven aan je editor scherm (icon met het plusje)
2. Selecteer middels de breadcrumb de juiste files
3. Selecteer een element in het storyboard, houdt vervolgens control ingedrukt en de muis ingedrukt en sleep dan de cursor naar het juiste element in de code (dit kan ook een lege regel zijn waar het element gedefinieerd moet worden).

Verschillende connection types:
Bij het koppelen van een element uit het storyboard met de code verschijnt er een menu waarin de connectie wordt geconfigureerd. Zo kan je kiezen uit verschillende connectie types:

- Action: Geeft aan dat er een actie wordt uitgevoerd door middel van een functie wanneer er op dit element wordt gedrukt
- Outlet: Definieert een class variabele voor het geselecteerde element
- Outlet collection: Definieert een array van meerdere elementen van hetzelfde type als class variabele

Verwijderen connectie:
Naast het maken van een connectie, is het uiteraard ook mogelijk om de connectie te verwijderen. Dit is in sommige gevallen wel gewenst. Het verwijderen van een connectie kan je doen op de volgende manieren:

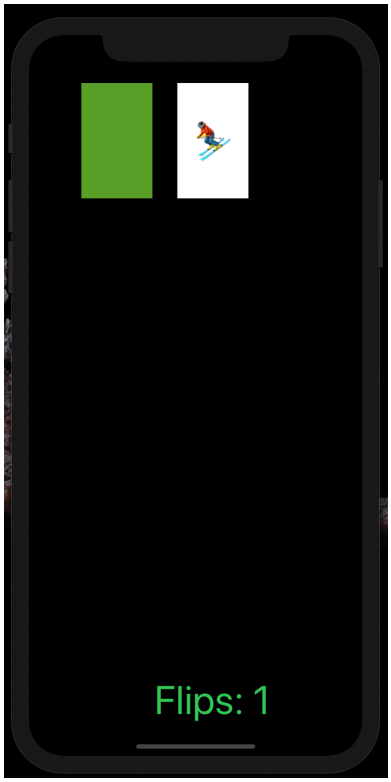
- Selecteer het element in het storyboard, klik vervolgens op het rondje met een pijltje erin (in het panel aan de rechterkant). Hier zie je de connecties die het element heeft. Voor de connectie is een klein kruisje te zien, hiermee kan je connectie verwijderen.
- Gebruik de rechtermuisknop (2 vingers op het touchpad) op het element in het storyboard, hiermee kan je snel de connecties zien. Net als bovenstaande punt kan je de connectie verwijderen door gebruik te maken van het kruisje.



Stap 1 - Eerste 2 kaarten, draaien & score

Tijdens de eerste stap is het de bedoeling dat je de volgende punten implementeert zodat de basis van de concentration app komt te staan.

- 2 kaarten maken via het Storyboard (en deze koppelen aan een functie voor interactie)
- De functie maken zodat de “kaart flip” wordt uitgevoerd
- Een score label maken die bij elke “flip” wordt opgehoogd



Een korte uitleg van een functie signiature in iOS:

```
func flipCard(withEmoji emoji: String, on button: UIButton)
```

- Func: Aanduiding dat het een functie is
- FlipCard: De naam van de functie
- WithEmoji: De externe naam van de parameter, deze wordt getoond wanneer je de functie aanroep maakt in de code
- Emoji: De interne naam van de parameter, dit is de naam welke binnen de functie gebruikt kan worden.
- String: Het datatype van de parameter

NB: Net als de meeste programmeertalen kent Swift ook accesssors en wordt de default gebruikt bij het weglaten van dit keyword.

Opbouw van de functie is zo, dat wanneer je het aanroept het heel logisch leest:
"Roep functie flipCard aan met emoji (withEmoji) van kaart (on UIButton)

Opstarten

- Clone het project van de repository (zie hoofdstuk Handige Informatie)
- Open het project in de folder **stap 1**

1.1 Opmaken user interface

- Open de main.storyboard class
- Om de kaarten straks te kunnen zien maak je eerst de achtergrond van de **view** zwart, dit kun je doen door deze te selecteren en middels de **attributes inspector** een background color te geven
- Maak een nieuwe button door in de **object library** (Command + Shift + L (of + button)) te zoeken op "button" en deze op je view te slepen
  - Maak de achtergrond van de button wit
  - Verhoog de lettergrootte middels de attributes inspector op 45 en zorg ervoor dat er é én willekeurige emoji (hotkey selectie paneel : control + command + spatie) in komt te staan
  - Maak de button kaartvorming door hem middels de **size inspector** op 80 * 130 px (breedte*hoogte)) in te stellen (dit kan ook door middel van slepen)

1.2 Koppelen UI aan de code

- Open de ViewController.Swift
  - Vul de functie touchCard aan met de volgende punten
    - Koppel als eerst de zojuist gemaakte knop door middels van drag-and-drop aan de functie toe (als het gelukt is en je gaat met de muis over de connecties van de functie zul je zien dat de button oplicht (dit kun je ook zien in de **connections inspector,** die zich rechtsboven bij alle inspectors bevindt)
    - Roep de flipCard functie aan in touchCard en implementeer vervolgens de missende logica in de else statement van de flipCard functie
    - Geef bij het aanroepen van de touchCard dezelfde emoji mee als de emoji die je aan de kaart hebt gegeven
- Run vervolgens links bovenin de editor, op een iPhone X (of >) simulator de code om vervolgens een kaart te zien die omkeert (en terug) wanneer je er op klikt

1.3 Toevoegen tweede kaart & score label

Let op: In deze stap gaan we een kaart toevoegen, de manier hoe, voelt (hopelijk) alleen niet goed, echter gaan we dit in stap 2 generieker oplossen.

- Kopieer de in stap 1.1 gemaakte kaart (button)
- Bekijk vervolgens in de **connections inspector**, hier zul je zien dat je niet alleen de kaart hebt gekopieerd, maar ook de associatie tussen de kaart en de touchCard functie

- Haal vervolgens de associatie tussen de tweede kaart en de touchCard functie weg
- Maak een nieuwe functie aan voor de tweede kaart, koppel de tweede kaart aan de functie middels drag & drop en zorg ervoor dat deze functie de flipCard functie aanroept.
- Maak vervolgens op het storyboard een label aan door middels de Object Library voor een UILabel te kiezen en maak maak hem op met de volgende punten:
  - Zet de text grootte op 45, kies een kleur voor de text en zet de default text op : 'Flips: 0' in de attributes inspector
  - Koppel het label in de ViewController aan de aangemaakte variabele flipCountLabel middels drag & drop

In de onderstaande snippet code uit de ViewController.swift zie je de @IBOutlet, dit is de referentie naar de in het storyboard aangemaakte flips label. Zoals je ziet wordt de text van deze variabele in de didSet functie van de **variabele** flipCount geüpdatet. Dit wil zeggen dat elke keer als er iets met flipCount gebeurd, zal de text van het flipCountLabel automatisch aangepast worden. Dit noemen we in swift een **computed property.**

```
// Deze variabele registreert de hoeveelheid flips
var flipCount: Int = 0{
    didSet {
        flipCountLabel.text = "Flips: \(flipCount)"
    }
}

// Deze variabele representeert de score
@IBOutlet weak var flipCountLabel: UILabel!
```

- Om stap 1.3 af te ronden moeten we alleen nog zorgen dat het de flipCount een increment krijgt wanneer je een kaart draait. Maak deze aanpassing in de code.

Je kunt nu verder gaan met **stap 2**

- Als alles is gelukt kun je verder werken met je eigen code, wanneer je tegen problemen aangelopen bent. Kun je er ook voor kiezen om de "Stap 2" te openen als nieuw project in xCode en hiermee verder te werken.

Stap 2 - Generieke oplossing, model &

In deze stap zal het al meer op een echt spel gaan lijken, we gaan namelijk werken aan de volgende punten:

- Een set (Array) van 12 kaarten aanmaken
- De kaarten “Face-down” hebben bij start
- Alle kaarten koppelen op de touchCard functie

2.1 Aanmaken array van 12 kaarten t.b.v. het spel

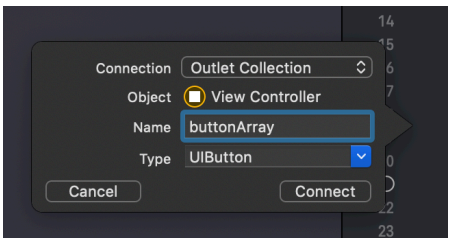
Om straks een werkende concentration game te krijgen moeten we de volgende punten m.b.t. de opmaak doen:

1. We gaan voorbereiding leggen om de emoji's in de code i.p.v. het storyboard toe te wijzen zodat we de startposities niet weten (en random kunnen instellen)
2. We willen een dozijn kaarten hebben en ze moeten "face-down" starten

Voorbereiding Emoji's vanuit code i.p.v. storyboard

- Verwijder de emoji's uit de bestaande kaarten in het storyboard
- Zet de background color van de kaarten op "Carthago groen"

Dozijn kaarten aanmaken



- Verwijder de connectie tussen de kaarten en stukken code
- Dupliceer de bestaande kaarten tot je een totaal van 12 kaarten hebt en centreer ze op het scherm
- Selecteer nu 1 kaart en koppel deze door middel van drag-en-drop in de ViewController als **outlet Collection** (Dit maakt een Array (collection) aan)
- Voeg vervolgens met behulp van drag-en-drop de rest van de kaarten toe aan de hierboven gemaakte lijst.
- Door over het bolletje voor de kantlijn te hoveren met de muis kan je controleren of alle kaarten zijn toegevoegd aan de lijst.

2.2 Generieke functie voor het draaien van de kaarten

- De touchSecondCard functie zal redundant worden en mag weg
- De code in touchCard kan voor nu blijven zoals het is
- Omdat we in de voorgaande stap alle referenties van de kaarten hebben verwijderd, moeten we ze nu opnieuw toevoegen (drag & drop) aan de touchCard functie
- Wanneer de app nu wordt gedraaid in de simulator zul je zien dat elke kaart hetzelfde zal reageren. Het enige wat nu ontbreekt is dat er verschillende paartjes zijn, dit gaan we in stap 3 regelen.

Stap 3 - Koppelen concentration engine

Voor stap 3 is het van belang dat er een aantal files ge-edit worden en extra files worden toegevoegd aan het project. Deze staan grotendeels ingevuld in de “Supporting files” folder vanaf stap 3. Wanneer je met je eigen implementatie verder gaat moet je deze in je project copy pasten.
Dit kun je doen door in je eigen applicatie in de root (of in supporting files) via rechtermuis klik “Add files” te selecteren en de files **Card.swift** en **Concentration.swift** op je file system te selecteren. De aanpassingen aan de ViewController.class moet je zelf ook doen (hiervoor kan je de ViewController.class uit stap 3 gebruiken)

Hieronder gaan we kort in op de toegevoegde classes en wat ze gaan doen m.b.t. het spel.

Card.swift
Deze class representeert een spelkaart en de bijbehorende eigenschappen, zo heeft de kaart een “unieke” identifier en kent hij zijn status.

Concentration.swift
Deze class representeert een instantie van het spel concentration. Deze class heeft een lijstje van alle kaarten in het spel, en bevat de beslis logica voor wat er met het spel moet gebeuren waneer een kaart gedraaid wordt. Tevens maakt deze class paren en moet deze de kaarten ook kunnen schudden zodat ze op willekeurige volgorde komen te liggen.

ViewController.swift
In deze class hebben we een aantal wijzigingen doorgevoerd zodat we kunnen focussen op het werkend maken van het geheel (en features toevoegen)

- Toegevoegde variabelen:
  ```
  private var emojiChoices: Array<String>
  private var emoji = [Int:String]()
  ```
- Toegevoegde functies:
  ```
  func emoji(for card: Card) -> String
  func updateViewFromModel()
  ```
- Verwijderde functies
  ```
  func flipCard(withEmoji emoji: String, on button: UIButton)
  ```

3.1 Wijzigingen om concentration model te gebruiken

- In de Concentration class hebben we in de initializer (init) een todo staan, we moeten naast het toevoegen van de kaart paren ook zorgen dat de volgorde veranderd om het speelveld random te maken (hint: check de functies die beschikbaar zijn voor arrays)
- In de ViewController class moeten we de touchCard functie aanpassen zodat hij de volgende punten uitvoert:
  - Het score label updaten bij een flip (dit gebeurde voorheen in de flipCard functie die nu weg is)
  - De index van de geselecteerde kaart ophalen (hint: check de array functies) en controleren dat deze daadwerkelijk een resultaat oplevert (hint: optional if let)
  - De game updaten met de gekozen kaart
  - De view updaten met de wijzigingen die voort zijn gekomen uit de laatste interactie

Als je alle bovenstaande stappen hebt doorlopen, heb je nu een werkende situatie net zoals stap 2, maar nu is de Concentration.swift class het model die de state van de kaarten bijhoudt. Ook zullen er nu paartjes van 2 emoji's, random op het speelveld te vinden moeten zijn.

In de volgende stap gaan we de laatste 2 stappen uitvoeren, namelijk aangeven wanneer een gebruiker heeft gewonnen en het herstarten van een spel. Mochten er mensen zijn die alles binnen de tijd af krijgen, zijn er nog een aantal suggesties opgenomen om aan te werken.

Stap 4 - Completing the game!

Net zoals de vorige stappen kun je met een “schone lei” beginnen in door stap 4 te openen in xCode of je eigen codebase te gebruiken voor de implementatie.

Einde van het spel
Het spel heeft in de huidige staat geen duidelijke eind positie, zo blijft het laatste paartje in beeld staan, krijg je geen feedback dat alle paren zijn gevonden en is er geen mogelijkheid om het spel opnieuw te starten. In deze stap ga je zelf proberen deze elementen toe te voegen aan het spel, bij elk element staat wel een hint om je op weg te helpen.
Mocht je echt vastzitten dan kan je altijd op hulp vragen of stap 5 uit de repository downloaden.

- Laatste paartje weghalen
  - De kaarten worden nu pas weggehaald wanneer een andere kaart wordt aangeklikt, dit is niet mogelijk om te doen wanneer er geen kaarten meer over zijn
  - De status van de kaart zou kort na het omdraaien bijgewerkt moeten worden
- "You won" text tonen wanneer het laatste paartje is weggehaald zodat je weet dat het spel is afgelopen
  - Hint: UIAlertController
- Mogelijkheid tot het opnieuw starten van het spel
  - Voeg een UIButton toe en geef deze de tekst restart
  - Koppel de Button aan de code en zorg ervoor dat het spel hiermee opnieuw wordt geïnitialiseerd

Extra opdrachten

Bij tijd over zou je eens kunnen proberen om één of meerdere van onderstaande punten te implementeren in het spel.

- Constraints (meerdere devices)
  - Met contstraints leg je de afmetingen en positie van een element in het storyboard vast. Dit kan je doen per device en per oriëntatie. Door gebruik te maken van constraints zorg je ervoor dat je app er altijd goed uit ziet. Een mooie uitdaging zou zijn om het spel met de juiste verhoudingen in landscape mode te laten werken.
- Topscores
  - Door het toevoegen van topscores kan je andere mensen uitdagen om je score te verbeteren. Breidt de UIAlertController uit de vorige stap verder uit, zodat de gebruiker de mogelijkheid heeft om zijn naam op te geven.
  - Navigeer naar een nieuw scherm waar de 10 beste scores staan
- Meerdere kaarten (moeilijkheidsgraden)
  - Om wat meer diversiteit aan het spel toe te voegen zou je ervoor kunnen kiezen om een startscherm toe te voegen waar de gebruiker de gewenste moeilijkheid selecteert.
  - Houdt er wel rekening mee dat de topscores ook per moeilijkheid moet worden bijgehouden dan!
- Animaties van de kaart (moeilijk)
  - In de huidige situatie wordt simpelweg de kleur van de kaart aangepast waardoor je het gevoel krijgt dat de kaart draait. Om het nog mooier te maken zou je animaties kunnen toevoegen waarmee de kaart echt wordt gedraaid.