UIKitDynamics - tchnij życie w swój interfejs!

Untitled Kingdom Ltd.

1. Wstęp

UIKitDynamics zostało dodane jako integralna część frameworku UIKit wraz z systemem iOS 7. Daje on możliwość wzbogacenia interfejsu naszej aplikacji - w dosłownie kilku liniach kodu - o rzeczywiste i dynamiczne zachowanie, które niegdyś wymagało dogłębnego zrozumienia procesów fizycznych oraz ponadprzeciętnych umiejętności programistycznych.

Ideą tego narzędzia jest kompozycja podstawowych zachowań opisanych w kolejnych działach - które mogą być dynamicznie dodawane i usuwane z obiektu animatora(dział 2. Animator) w trakcie działania aplikacji.

W systemie iOS 7 zastosowano to narzędzie w kilku miejscach i chociaż - na pierwszy rzut oka - nie jest to oczywiste, znacznie uatrakcyjnia ono ogólny odbiór nowego systemu, stanowiąc przysłowiową “wisienkę na torcie”. Dobrym przykładem jest animacja włączania kamery z zablokowanego ekranu - niegdyś była to na sztywno zapisana animacja, teraz ekran odbija się od dolnej krawędzi dynamicznie, odpowiednio do prędkości, jaką mu nadamy. Działanie tego narzędzia odnajdziemy również na liście wiadomości SMS/iMessage, gdzie poszczególne “chmurki” wiadomości reagują na prędkość i kierunek przesuwania się listy.

Warsztaty te mają na celu zapoznać Państwa z możliwościami tego niezwykłego narzędzia, a niniejszy dokument stanowi uzupełnienie oraz skrypt do warsztatów.

Aby dowiedzieć się więcej o którejś z omawianych niżej klas, polecam zapoznać się z oficjalną dokumentacją, do której odnośniki odnaleźć można na końcu każdego działu.

2. Animator

Obiekty klasy **UIDynamicAnimator** są sercem UIKitDynamics. Pośredniczą one pomiędzy silnikiem fizycznym a *“dynamicznymi obiektami”* (rozdział 3). Odpowiadają również za symulowanie procesów fizycznych zachodzących na ekranie oraz kontrolują układ współrzędnych niezbędny do obliczeń. Sam animator nie wpływa jednak w żaden sposób na zachowanie interfejsu. Niezbędnym jest dodanie do niego dynamicznych zachowań, wraz z przypisanymi do nich dynamicznymi obiektami.

Aby otrzymywać informacje o zmianie stanu animatora, musimy zaimplementować metody protokołu *UIDynamicAnimatorDelegate* oraz przypisać odpowiednią referencję. Metody w nim zawarte informują o zakończeniu wszystkich animacji oraz o tym, że animator zaraz wznowi działanie. Jest to niezwykle istotne przy definiowaniu dynamicznych przejść pomiędzy kolejnymi kontrolerami widoków.

Równoległe Może istnieć wiele obiektów tej klasy.

Link do dokumentacji: http://bit.ly/Hi8LAF

3. Dynamiczne obiekty

Dynamicznym obiektem (ang. **dynamic item**) nazywamy każdy obiekt dziedziczący z klasy NSObject, który implementuje protokół *UIDynamicItem*. Metody tego protokołu pozwalają animatorowi na zmianę położenia obiektu oraz jego obrót. Aby zapewnić poprawne działanie obiekt musi posiadać następujące atrybuty:

- center , typ : CGPoint - środek obiektu
- bounds, typ CGRect - rozmiar obiektu, nie jest zmieniany
- transform, typ CGAffineTransform - transformacja(obrót)

Domyślnie ten protokół implementują 2 klasy, UIView oraz UICollectionViewLayoutAttributes, jednak nic nie stoi na przeszkodzie aby zaimplementować go w innej klasie.

Ta sama instancja dynamicznego obiektu może należeć do wielu dynamicznych zachowań pod warunkiem że zachowania te działają w obrębie jednego animatora.

Ważna uwaga: **aby zapewnić poprawność działania, obiekt(widok) który dodajemy do animatora musi być jednym z podwidoków jego widoku odniesienia.**

Link do dokumentacji: http://bit.ly/18h0YIM

4. Zachowania (Behaviours)

Działy 5 - 10 opisują podstawowe dostępne zachowania które łączy kilka wspólnych cech:

1. Wszystkie opisane klasy w działach 5 - 10 dziedziczą z klasy UIDynamicBehaviour
2. Obiekty każdej z klas(zachowań) mogą dodawać podzachowania co pozwala na grupowanie ich jako jedno zachowanie
3. Do każdego z zachowań można przypisać blok który wykonywany będzie przy każdym kroku animacji

Link do dokumentacji: http://bit.ly/1dth8a6

5. Złączanie obiektów

Obiekty klasy **UIAttachmentBehavior** pozwalają na zdefiniowanie połączenia pomiędzy dwoma dynamicznymi obiektami lub dynamicznym obiektem i punktem zaczepienia. W praktyce oznacza to utworzenie niewidzialnej linii łączącej te 2 elementy której właściwości możemy modyfikować. Są to: długość, sprężystość oraz częstotliwość drgań wyrażona w herzach

Ważna uwaga: **Początkowo punkt zaczepienia każdego obiektu jest zdefiniowany w jego centrum (zmienna center) jednak można to zmienić, definiując przesunięcie od środka obiektu.**

Link do dokumentacji: http://bit.ly/1arh2MQ

6. Kolizje

Zachowania klasy **UICollisionBehavior** pozwalają dodać efekt kolizji do przypisanych do niego dynamicznych obiektów. Warto zauważyć, że w obrębie jednego animatora może istnieć wiele obiektów tej klasy. Każdy z nich może obsługiwać zderzenia innych obiektów.

Parametrami, które możemy definiować, są: układ odniesienia animatora - czyli widok w którym zachodzą kolizje oraz ich rodzaj. Obecnie dostępne są 3 rodzaje kolizji:

1. Kolizje wszystkiego ze wszystkim, czyli kolizje pomiędzy obiektami oraz widokiem odniesienia // **UICollisionBehaviorModeEverything**
2. Kolizje tylko z widokiem odniesienia - dynamiczne elementy ignorują się // **UICollisionBehaviorModeBoundaries**
3. Kolizje jedynie pomiędzy dynamicznymi obiektami // **UICollisionBehaviorModeItems**

Nowo utworzony obiekt tej klasy nie posiada zdefiniowanych krawędzi, z którymi mogłyby zachodzić kolizje. Aby je utworzyć mamy 2 możliwości.

Jeżeli chcemy aby wewnętrzne krawędzie (bounds) naszego widoku odniesienia brały udział w kolizjach musimy ustawić wartość parametru **translatesReferenceBoundsIntoBoundary** na YES.

Drugą jest możliwość definiowania dowolnych linii, z którymi będą zachodziły kolizje. Linie te mogą przyjmować praktycznie dowolne kształty, dzięki zastosowaniu krzywych beziera lub prostych określonych dwoma punktami.

Ważna uwaga: **Początkowe położenie obiektu dodanego do animatora nie może przecinać się z żadnym elementem z którym może zachodzić kolizja.**

Mamy również możliwość otrzymywania informacji o kolizjach, które zaszły. Do tego celu należy zaimplementować metody zdefiniowane w protokole **UICollisionBehaviorDelegate**.

Link do dokumentacji: http://bit.ly/1a3RBgl

7. Grawitacja

Obiekty klasy **UIGravityBehavior** zmieniają pozycję swoich elementów na podstawie wektora, który określa siłę grawitacji. Jego domyślna wartość to [0.0, 1.0], jednak jest to modyfikowalny atrybut o nazwie gravityDirection. Na potrzeby tego zachowania zdefiniowano nową jednostkę imitującą przyspieszenie ziemskie o nazwie UIKit gravity, którego wartość wynosi 1000 punktów / s^2. Dodatkowo, wektor siły możemy określić wywołując metodę setAngle: magnitude: która pozwala na określenie kąta (w radianach) oraz siły grawitacji.

Link do dokumentacji: http://bit.ly/16wM9rd

8. Popychanie

Obiekty klasy **UIPushBehavior** zmieniają pozycję przypisanych do nich elementów poprzez “popychanie” ich z jednostajną siłą. Początkowo siła popychania ma ustawioną wartość **nil** która oznacza brak zdefiniowanej siły.

Tak jak w przypadku grawitacji siłę popychania ustalamy poprzez wektor lub kąt i siłę. Zdefiniowano również na potrzeby tego zachowania jednostkę reprezentującą siłę o nazwie UIKit Newton która wynosi 100 punktów / s ^ 2.

Siła przykładana jest do środka widoku (center) ale tak samo jak w przypadku złączania obiektów jest to modyfikowalny parametr.

To zachowanie można włączać i wyłączać na żądanie poprzez własność *active.*

Warto wspomnieć iż początkowo wartość siły jest równa [0.0, 0.0], a prędkość jaką osiągnie widok jest uzależniona od 2 czynników:

1. Jego gęstości
2. Trybu działania tego zachowania

Wyróżniamy 2 tryby działania:

- UIPushBehaviorModeContinuous - obiekt przyśpiesza stopniowo zgodnie ze zdefiniowanym wektorem siły
- UIPushBehaviorModeInstantaneous - w momencie dodania do animatora obiekt uzyskuje docelową prędkość i nie zmienia jej

Link do dokumentacji: http://bit.ly/1hgjnxj

9. Przyciąganie

Obiekty klasy UISnapBehavior definiują zachowanie przyciągania do zdefiniowanego punktu. Po dodaniu tego zachowania do animatora następuje animacja przyciągania do punktu która konczy się “drżeniem” wokół punktu.

Link do dokumentacji: http://bit.ly/1hgjdGq

10. Indywidualne własności widoków

Obiekty klasy UIDynamicItemBehavior pozwalają definiować niskopoziomowe własności widoków(obiektów) takie jak:
- **elasticity** - elastyczność obiektu, wartości od 0 do 1, 0 - oznacza brak elastyczności
- **friction** - współczynnik tarcia - 0 oznacza brak tarcia podczas przesuwania obiektów po powierzchni
- **density** - gęstość, domyślna wartość to 1
- **resistance** - opór jaki stawia obiekt podczas przemieszczania się, 0 - oznacza brak oporów
- **angularResistance** - opór kątowy, wpływa na podatność obiektu na obrót
- **allowsRotation** - pozwala na zablokowanie rotacji obiektu

Link do dokumentacji: http://bit.ly/HlZ97a

11. Uwagi i porady

1. UIKitDynamics nie służy do tworzenia gier, do tego celu należy używać frameworku SpriteKit który również został dodany w iOS7

2. Pomimo iż jest to świetne narzędzie, należy używać go z rozwagą, ponieważ jest on stosunkowo drogi pod względem obliczeniowym - w szczególności kolizje - co w praktyce przekłada się na większe zużycie energii

3. Przed dodaniem zachowań do animatora należy zapewnić “rozsądny” (cyt. WWDC) stan początkowy animowanych obiektów.
Przykładem nierozsądnego stanu początkowego może być dodanie widoku do zachowania kolizji (UICollisionBehavior) którego początkowe położenie koliduje z innymi elementami. Spowoduje to nieprzewidywalne zachowanie elementów widoku

4. Obsługiwane są jedynie rotacje 2D a zewnętrzne zmiany wykonywane na widokach / obiektach są ignorowane

5. Grupowanie zachowań w jedno nie powoduje zwiększenia obciążenia CPU

6. Dodanie na raz wielu obiektów UIDynamicItemBehavior modyfikujących tą samą własność do drzewa obiektów animatora może mieć nieoczekiwane efekty - pod uwagę brana jest ostatnii obiekt przy przechodzeniu tego drzewa metodą pre-order.

7. Animator nigdy nie zmienia rozmiaru animowanego obiektu. Jeżeli chcemy zmienić rozmiar dynamicznego obiektu należy usunąć go z animatora, dokonać zmian a następnie dodać z powrotem.

8. Ta sama instancja obiektu UIDynamicBehaviour może być dodana tylko raz do animatora.

12. Dodatkowe źródła

Polecam zapoznać się z niżej wymienionymi artykułami na temat UIKitDynamics, można tam odnaleźć interesujące pomysły na jego zastosowanie oraz niestandardowe implementacje.

- http://www.codigator.com/tutorials/uikit-dynamics-ios7-tutorial/
- http://blog.bignerdranch.com/3899-uikit-dynamics-and-ios-7-building-uikit-pong/
- http://www.objc.io/issue-5/collection-views-and-uidynamics.html
- http://www.raywenderlich.com/50197/uikit-dynamics-tutorial
- http://www.teehanlax.com/blog/introduction-to-uikit-dynamics/
- http://www.teehanlax.com/blog/implementing-a-bouncy-uicollectionviewlayout-with-uikit-dynamics/
- http://www.doubleencore.com/2013/09/ios-7-uikit-dynamics/

Przykładowe projekty przeglądowe UIKitDynamics:

- https://github.com/croath/DynamicCatalog
- https://developer.apple.com/library/ios/samplecode/DynamicsCatalog/Introduction/Intro.html

Dla osób będących posiadających konta w Apple Developer Program polecam 2 nagrania z WWDC 2013 (https://developer.apple.com/wwdc/videos/) : **Advanced Techniques with UIKit Dynamics** oraz **Getting Started with UIKit Dynamics.**