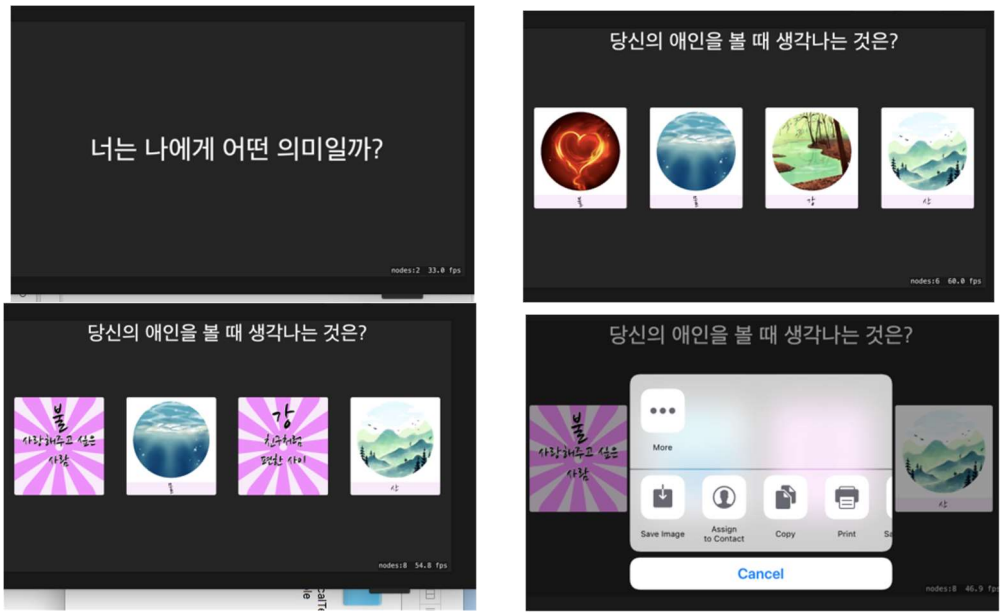
20 장. 심리 테스트 앱

사용자와 상호작용하는 모든 요소를 직접 만들어낸 후 SpriteKit 을 이용해 이를 보여주고 제어하게 된다. 즉, 게임을 만드는 것과 같다. 점수를 얻는다거나 하는 과정은 없지만 이번 예제를 만들어 보면서 게임 제작의 기초 과정을 배울 수 있다.



앱은 2 개의 화면으로 구성되어 있다. 심리 테스트 내용은 '너는 나에게 어떤 의미일까? 라는 것입니다. 애인을 생각할 때 무엇이 생각나는지 총 4 개의 보기가 준비되고 그 중 하나를 고르면 그 것이 어떤 의미인지 화면에 표시하고 효과를 준다. 또한 표시한 내용을 SNS 에 공유하는 기능도 구현한다. 생각했던 사람을 태그할 것인지는 사용자의 자유에 맡긴다.

20-1. SpriteKit

SpriteKit 은 iOS 와 macOS 에서 실행할 수 있는 게임을 쉽게 제작할 수 있게 하는 프레임워크이다.

SpriteKit 의 스프라이트란 화면에서 보여지는 여러 가지 그래픽 객체들을 의미한다. SpriteKit 은 이 스프라이트를 화면에 보여주기 위한 랜더링 관련 클래스와 객체가 상호 작용할 수 있게 하는 액션을 제공한다. 그래서 이 액션과 연결되는 메서드들을 만들면 객체들의 동작을 제어할 수 있게 되어 있다.

SpriteKit 은 애니메이션을 구현하고 제어할 수 있게 해주는 물리 엔진도 제공한다.

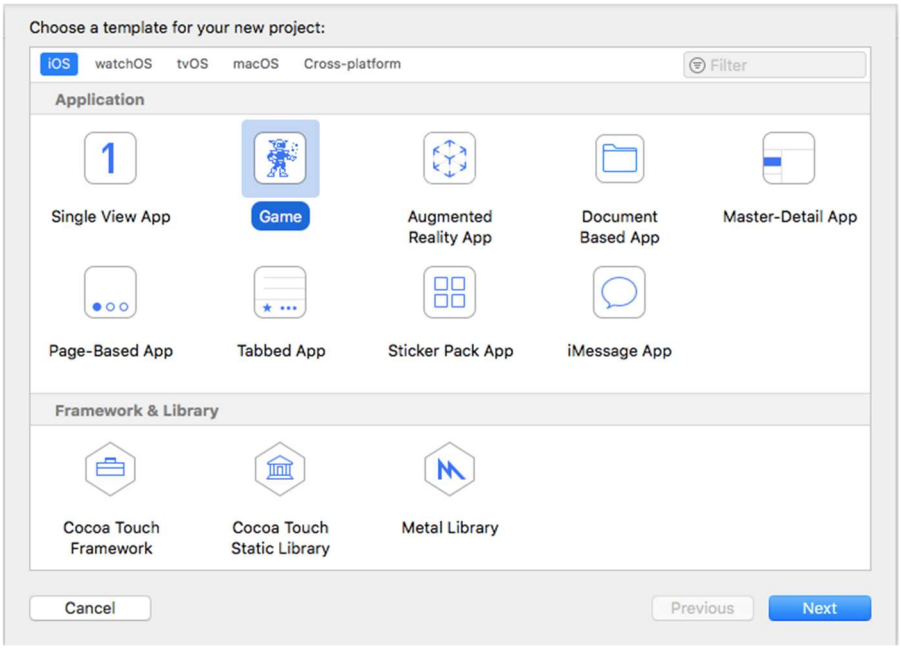
프레임이 갱신될 때마다 먼저 update 메서드가 실행된다. 그리고 SpriteKit 클래스를 통해 액션 평가, 물리 시뮬레이션과 constraint 를 적용한다. 이때 하나의 효과가 작용될 때마다 바로 didEvaluateAction, didSimulatePhysics, didApplyConstraints 메서드를 실행해 적용한다. 최종적으로 didFinishUpdate 메서드가 실행되면 SKView 가 전체 적용된 효과로 구현되는 장면을 랜더링하여 출력하게 된다. 프레임마다 이 과정을 반복한다.

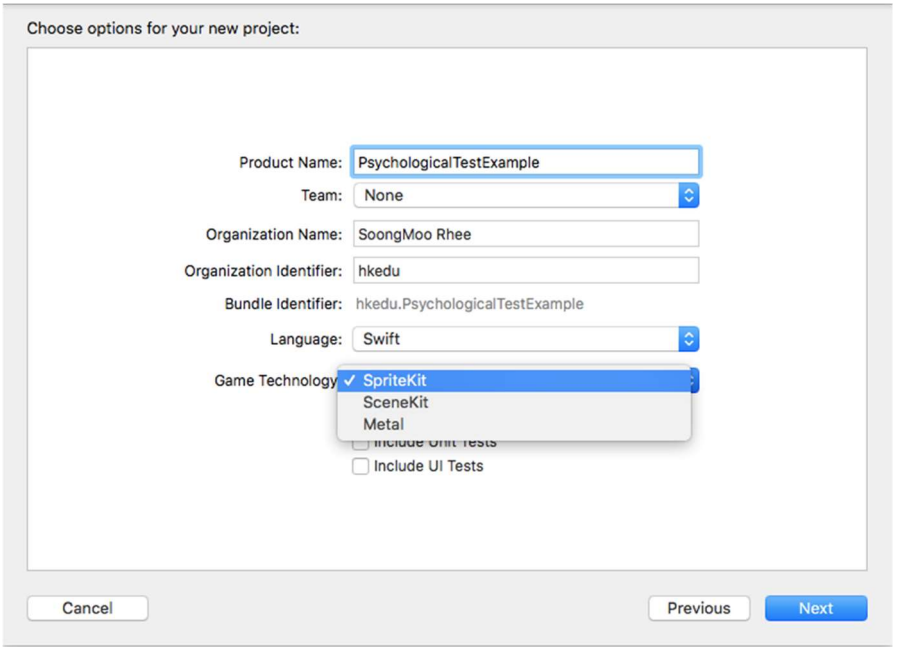
따라서 우리는 프레임마다 일어나는 이과정을 이해해서 각 절차에 맞게 적용할 효과만 지정해 주면 된다. 그러면 실행 결과는 SpriteKit 에서 원칙으로 담당해 화면에 보여주므로 신경 쓸 필요가 없다.

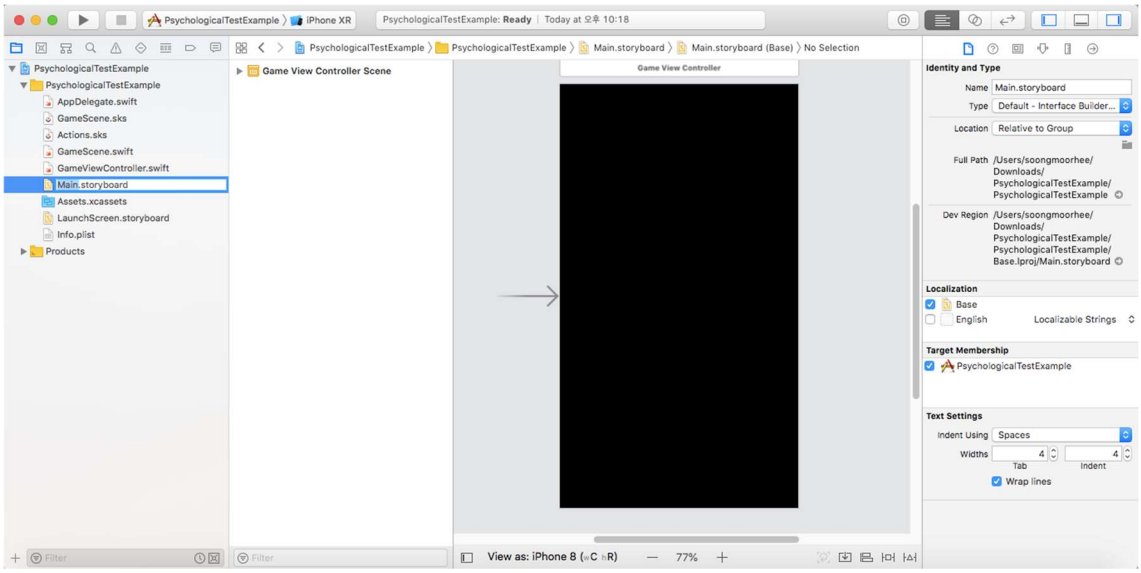
20-2. 화면 구성하기

1. Xcode 를 실행한 후 Game 을 선택한다.

프로젝트명은 ''PsychologicalTestExample" 라는 이름으로 새 프로젝트를 만든다







project navigator 에서 GameScene.swift 파일을 서택해서 didMove 메서드를 수정한다.

```
1   override func didMove(to view: SKView) {
2
3           // Get label node from scene and store it for use later
4           self.label = self.childNode(withName: "//helloLabel") as? SKLabelNode
5           if let label = self.label {
6               label.alpha = 0.0
7               label.run(SKAction.fadeIn(withDuration: 2.0))
8
9               label.text = "너는 나에게 어떤 의미일까?"
10              label.fontName = "AppleSDGothicNeo-Medium"
11              label.fontSize = 45
12          }
13  ...
14  }
```

9 행은 텍스트는 심리 테스트 앱에서 사용하려는 "너는 나에게 어떤 의미일까?" 라는 문장으로 교체한다.

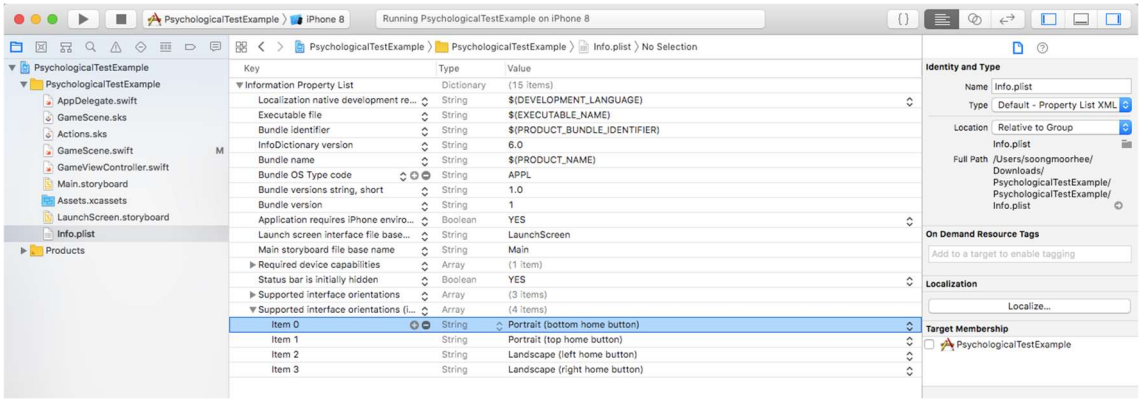
10 행은 SKLabelNode 를 만들 때 사용하던 폰트는 한글을 표현하기에 적당하지 않아 iOS 에서 기본적으로 지원하는 한글인 "AppleSDGothicNeo-Medium" 으로 변경한다.

11 행은 글자의 크기를 45 로 결정한다,

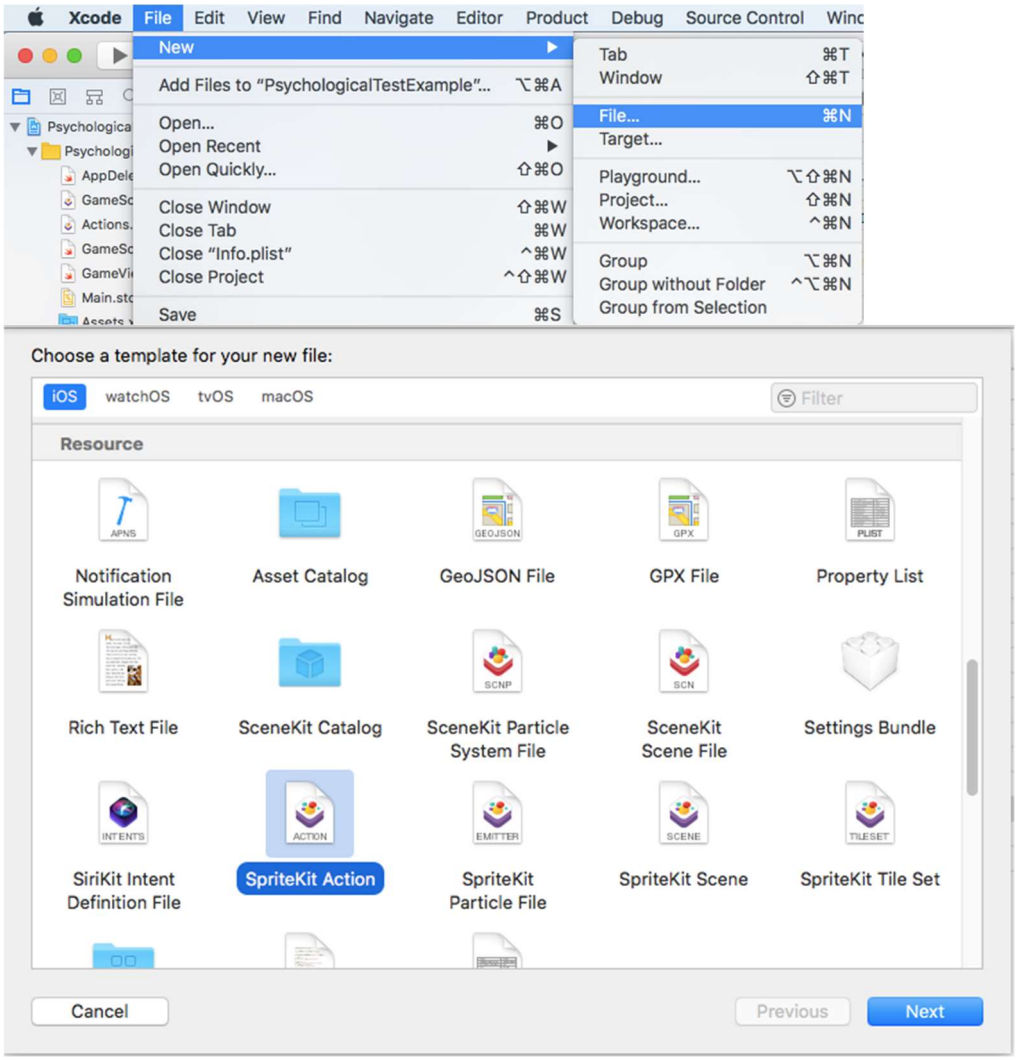
Navication 에서 info.plist 를 선택한다.

[Information Property List]에서 맨 아래에 있는 [Supported interface orientations]의 앞에 있는 화살표를 클릭하여 목록을 연다.

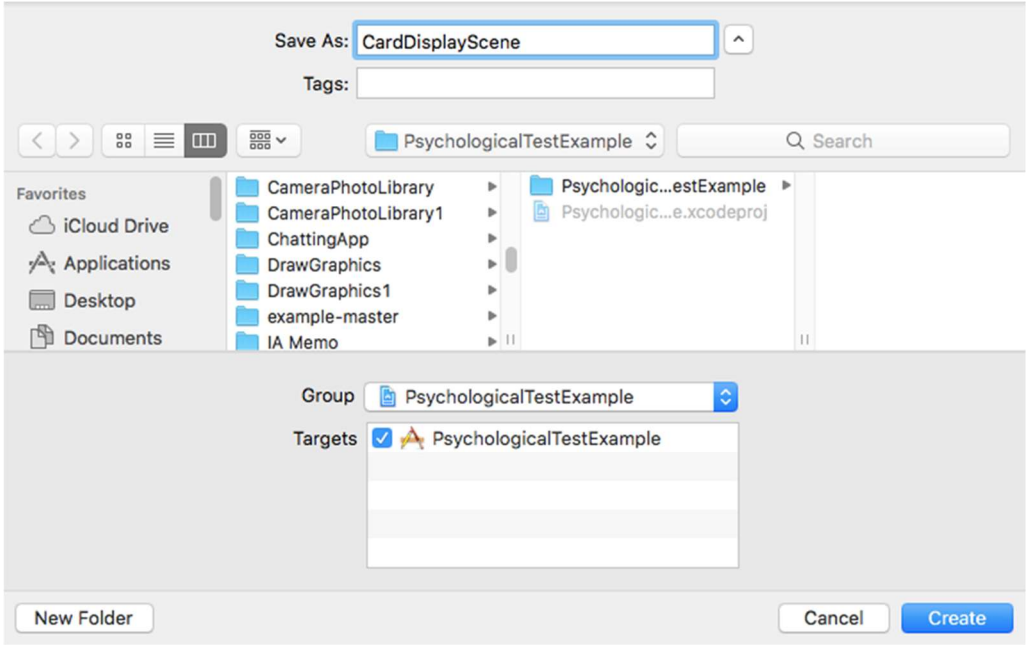
[item0]의 설정 값이 'Portraint'인 것을 확인할 수 있다. 해당 항목 옆에 있는 <->버튼을 클릭해서 이 목록을 제거 한다.



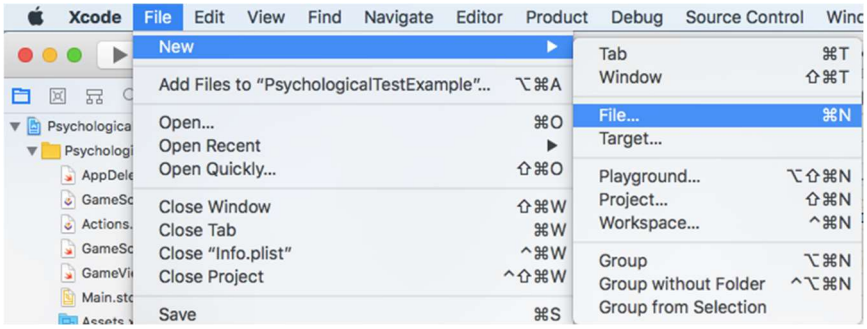
이제 앱을 실행시키면 가로로 실행되는 것을 확인할 수 있다.



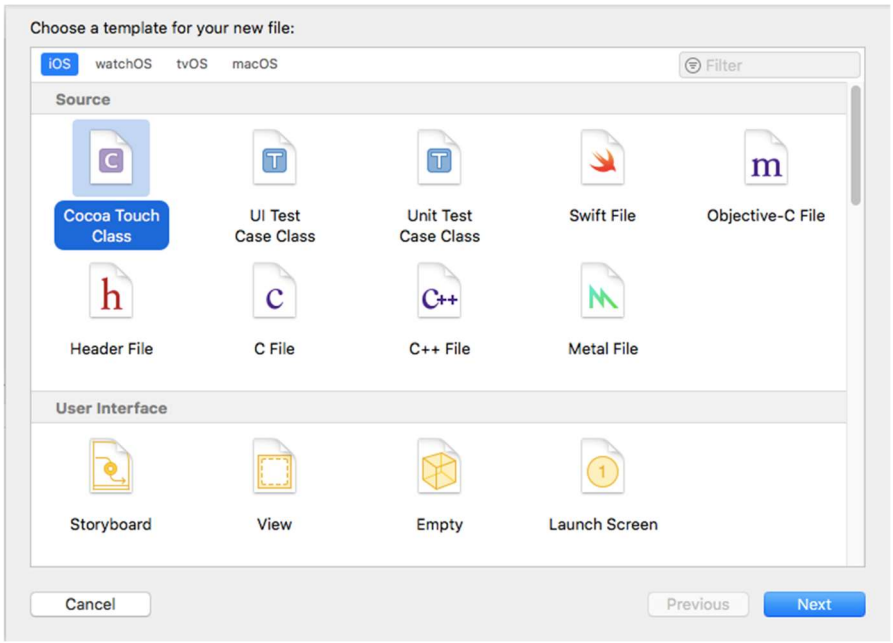
파일명을 'CardDisplayScene'으로 입력하고 [Create]버튼을 클릭한다.



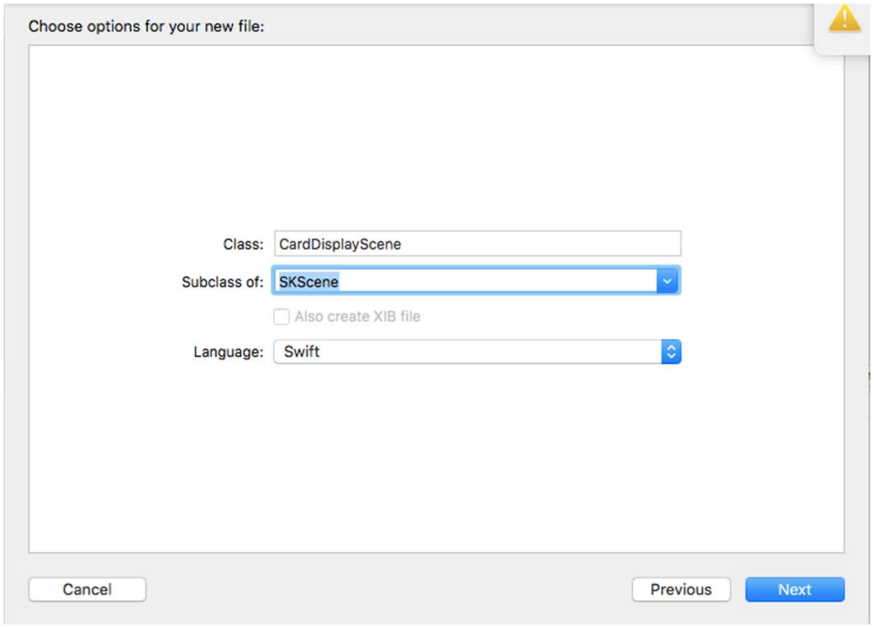
[CardDisplayScene.sks] 파일이 추가되면 다시



선택한 후 동일한 이름의 swift 파일을 추가한다. 편집 화면을 연다.



파일명은 CardDisplayScene.swift 로 한다.



추가된 장면으로 이동하기 위해 기존 장면에서 touchesBegan 메서드의 내용을 변경한다,

GameScene.swif 파일을 선택한 후 GameScene 클래스의 touchesBegan 메서드의 기존 코드를 모두 지우고 다음과 같이 변경 한다,

didMove, touchesBegan, update 메서드 빼고 다른 모든 메서드는 삭제한다.

```
1   override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
2           if let label = self.label {
3               let fadeAction = SKAction.fadeOut(withDuration: 1.0)
4
                label.run(fadeAction, completion: {
5                   let sceneTransition = SKTransition.push(with:
6   SKTransitionDirection.left, duration: 1.0)
7                   let cardScene = CardDisplayScene()
                    cardScene.size = (self.view?.bounds.size)!
8                   self.view?.presentScene(cardScene, transition: sceneTransition)
9               })
10          }
    }
11
12
13
```

4 행은 이번 장에서 다음 장으로 넘어갈 때는 label 노드를 페이 아웃시킬 것이다. 따라서 장면이 이동하는 시간을 설정해야 한다. 노드에는 fadeAction 이라는 상수를 선언하고 SKAction 클래스의 인스턴트에 fadeout(withDuration:) 메서드를 통해 페이드 아웃 효과를 사용하게 하고 초 단위의 시간을 설정한다.

7 행은 페이드 아웃 효과의 실행은 run 메서드가 담당한다. 액션이 종료되면 completion 블럭의 코드가 실행되는데 이때 장면의 전환이 실행되면 된다.

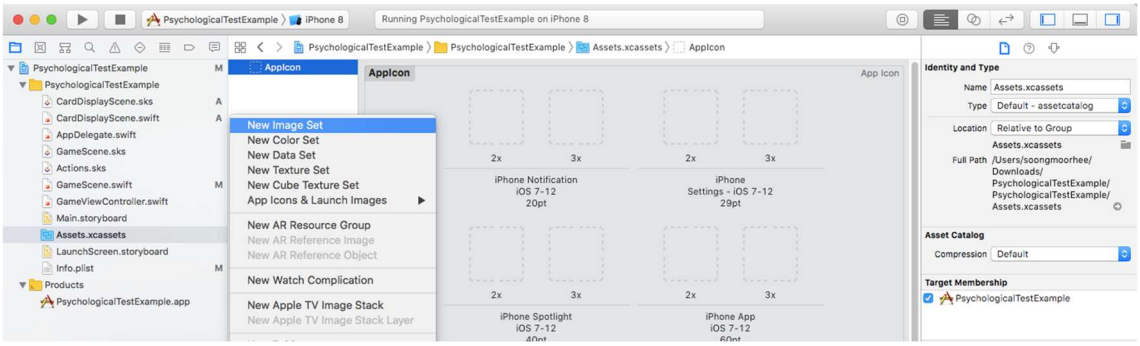
기존 장면이 왼쪽에서 밀리면서 다음 장면이 나오도록 할 것이다. 장면의 전환 효과는 SKTransition 클래스를 이용해 페이드 아웃 액션을 만들 때와 같은 방법으로 할당하며, 이때 사용하는 push 메서드는 SKTransitionDirection 클래스의 인스턴스에 up, down, left, right 총 4 가지로 장면이 전환되는 방향을 지정할 수 있다. 예를 들면 left 로 지정하면 장면이 왼쪽으로 밀리면서 전환된다.

8 행은 전환 효과를 만든 후에는 CardDisplayScene 클래스를 이용해 새롭게 등장할 장면에 이용할 오브젝트를 생성하고 새로 등장하는 장면의 화면 크기를 현재 화면 크기와 동일하게 설정한다, 크기를 지정해주지 않으면 다음

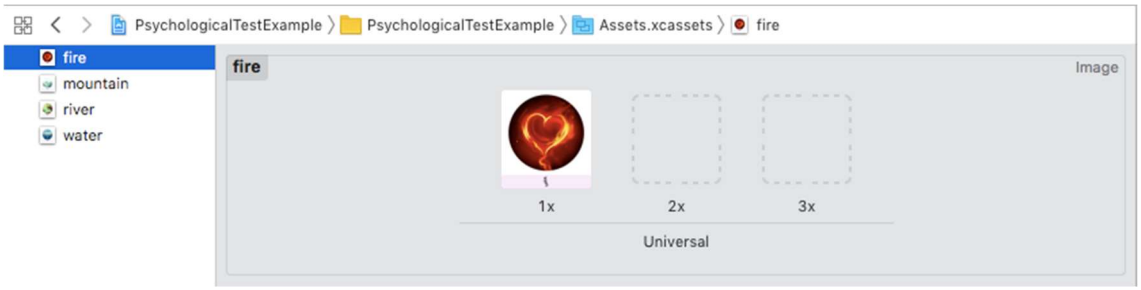
장면에 추가할 요소들을 정확한 위치에 추가할 수 없다.
10 행운 SKView 의 presentScene 메서드를 이용해서 장면의 전환을 실행한다.

효과가 적용되었으니 실행해서 확인해 보자. 앱을 실행하고 타이틀 화면을 터치하면 다음 장면으로 전환이 된다.

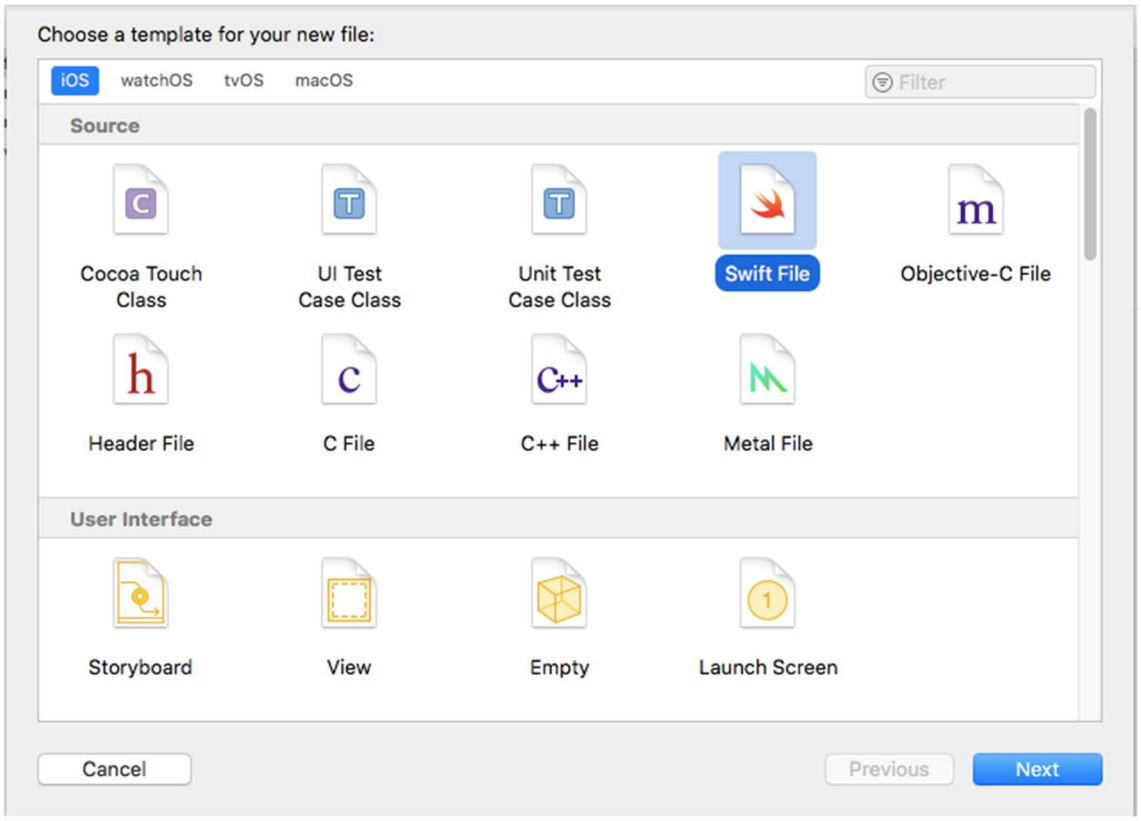
새로운 장면으로 전환이 된다면 이장면에 카드 4 장이 나타나야 한다.
Project navigator 에서 [Assets.xcassets]폴더를 이동한다. < - >버튼을 클릭해서 제거한다.



그리고 find.png 파일을 찾아서 1x 에 해당하는 곳에 드래그 앤 드롭해 넣어주면 fire 에셋 생성이 끝난다.
같은 방법으로 mountain, river, water 에셋도 생성한다,



이미지 에셋이 준비되었으니 이와 연관된 코드를 작성할 차례이다. [File] -> [New] -> [File]을 선택하고 [iOS] -> [Source] -> [Swift File]을 선택해 프로젝트에 card.swift 파일을 새로 만들어서 추가한다.




```
1  import Foundation
2  import SpriteKit
3
4  class Card: SKSpriteNode {
5      required init?(coder aDecoder: NSCoder) { //
6          fatalError("init(coder:) has not been implemented")
7      }
8      init(cardName: String) {
9          let texture = SKTexture.init(imageNamed: cardName)
10         super.init(texture: texture, color: UIColor.white, size: texture.size())
11     }
```

```
12  }
13
```

5~7 행은 스프라이트 노드를 초기화한다. 초기화되지 않을 경우 에러 메시지가 나타난다.

9 행은 노드를 생성할 때는 SKTexture 클래스의 init(imageNamed:)메서드를 이용해 텍스처를 할당하게 한다.

11 행은 texture 상수를 이용해 SKSpriteNode 클래스의 init 메서드를 이용해 텍스처를 입히고, 색상을 설정하고 이미지 크기를 설정한다.

그럼 만들어진 카드의 인스턴스를 설정해서 화면에 출력해보자.

CardDisplayScene 클래스의 didMove 메서드를 구한다,

```
1   import SpriteKit
2
3   class CardDisplayScene: SKScene {
4       static var cardScale: CGFloat = 0.0
5
6       override func didMove(to view: SKView) {
7           let width = self.view?.bounds.width
8
9           let titleLabel = SKLabelNode(fontNamed: "AppleSDGothicNeo-Medium")
            titleLabel.name = "titleText"
10          titleLabel.text = "당신의 애인을 볼 때 생각나는 것은?"
11
12          titleLabel.fontSize = 30
13          titleLabel.position = CGPoint(x: self.frame.midX, y: self.frame.maxY-
14  titleLabel.frame.height)
15          self.addChild(titleLabel)

16          let fireCard = Card(cardName: "fire")
17          let waterCard = Card(cardName: "water")
18          let riverCard = Card(cardName: "river")
19          let mountainCard = Card(cardName: "mountain")
20
21          CardDisplayScene.cardScale = 0.2 * width! / fireCard.size.width
22
23          fireCard.xScale = CardDisplayScene.cardScale
            fireCard.yScale = CardDisplayScene.cardScale
24          waterCard.xScale = CardDisplayScene.cardScale
25          waterCard.yScale = CardDisplayScene.cardScale
26          riverCard.xScale = CardDisplayScene.cardScale
27          riverCard.yScale = CardDisplayScene.cardScale
28          mountainCard.xScale = CardDisplayScene.cardScale
```

```
29              mountainCard.yScale = CardDisplayScene.cardScale
30
31              let MidX = self.frame.midX
32              let MidY = self.frame.midY
33              fireCard.position = CGPoint(x: MidX / 4, y: MidY)
34              waterCard.position = CGPoint(x: MidX * 3 / 4, y: MidY)
35              riverCard.position = CGPoint(x: MidX * 5 / 4, y: MidY)
36              mountainCard.position = CGPoint(x: MidX * 7 / 4, y: MidY)
37
38              self.addChild(fireCard)
39              self.addChild(waterCard)
                self.addChild(riverCard)
40              self.addChild(mountainCard)
41          }
42  }
43
44
45
46
```

4 행은 수치정보를 다를 cardScale 이라는 변수를 정의한다. 이는 비율에 맞춰 카드 이미지를 배치하는 데 사용한다.

7 행은 width 이라는 변수를 정의한다. 역시 비율에 맞춰 카드 이미지를 배치하는 데 사용한다.

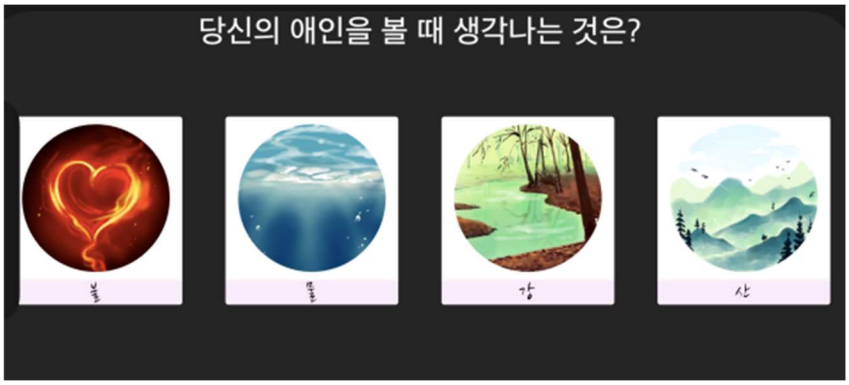
9~16 행은 title_abel 이라는 상수를 만들고 [Label]정보를 다룬다.

18~21 행은 4 개카드 이미지를 다룰 상수를 각각 정의한다.

23~32 행은 화면에 크기에 따라 비율을 조절해주는 코드를 추가한다. 이 때 계산된 비율은 이 후 애니메이션을 적용할 때 필요하기 때문에 클래스 속성으로 저장해둔다.

34~39 행은 카드의 위치를 지정한다. Fire, water, river, mountain 순서로 화면의 너비에 따라 적당한 위치를 계산한다.

41~44 행은 4 개 카드 이미지를 추가한다,



20-3. 카드 선택하기

카드를 터치해도 아무런 반응이 없다면 SpriteKit 을 사용하는 의미가 없다.

우선 카드를 터치하면 커졌다가 다시 작아지는 효과를 구현해보자.

```
1  class Card: SKSpriteNode {
2      required init?(coder aDecoder: NSCoder) { //
3          fatalError("init(coder:) has not been implemented")
4      }
5      init(cardName: String) {
6          let texture = SKTexture.init(imageNamed: cardName)
7          super.init(texture: texture, color: UIColor.white, size:
8  texture.size())
9          isUserInteractionEnabled = true
10     }
11 override func touchesBegan(_ touches: Set<UITouch>, with event:
12 UIEvent?) {
13         let liftUp = SKAction.scale(to: CardDisplayScene.cardScale *
14 1.2, duration: 0.2)
15         run(liftUp)
16     }
17
18     override func touchesEnded(_ touches: Set<UITouch>, with event:
19 UIEvent?) {
20         let dropDown = SKAction.scale(to:
21 CardDisplayScene.cardScale, duration: 0.2)
```

```
22              run(dropDown)
23
24          }
25  }
26
27
```

11 행을 추가한다. 카드 노드가 터치 이벤트를 인식할지를 결정하는 프로퍼티인 isUserInteractionEnabled 를 true 로 변경했기 때문에 이제부터는 터치 등의 이벤트를 카드 노드가 인식할 수 있다.

그래서 옮겨온 2 개의 터치 관련 메서드가 동작할 수 있다.

13 행부터 26 행을 추가한다.

```
1   class Card: SKSpriteNode {
2       ...
3       override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
4           let liftUp = SKAction.scale(to: CardDisplayScene.cardScale * 1.2, duration:
    0.2)
5           run(liftUp)
6
7           let rotateCCW = SKAction.rotate(toAngle: CGFloat(-M_1_PI / 10),
8   duration: 0.1)
            let rotateCW = SKAction.rotate(toAngle: CGFloat(M_1_PI / 10), duration:
    0.1)
9           let rotate = SKAction.sequence([rotateCCW, rotateCW])
            let rotateRepeat = SKAction.repeatForever(rotate)
10          run(rotateRepeat, withKey: "rotate")
11      }
12      ...
13  }
14
```

8~13 행 추가한다.

8 행은 흔들리는 효과를 주려면 카드가 양쪽으로 반복해서 회전하는 형태로 만들면 된다. 따라서 회전 액션을 만드는 rotate 메서드를 사용한다. 첫 번째 파라미터는 라디안으로 계산된 회전 각도 그리고 애니메이션의 진행 시간이다.

9 행은 반대쪽으로도 회전하는 액션을 하나 추가

10 행은 회전 반복하기 위해서는 sequence 메서드를 이용해 하나의 연속적인 액션을로 만들면 된다.

11 행은 연속적인 액션을 반복하기 위해서 repeatFoever 메소드를 사용해서

다시 SKAction 객체를 생성한다.
12 행은 생성된 rotateRepeat 액션을 run 메서드를 통해 실행하면 완성이다. 이때 중요한 것은 파라미터 withKey 의 값을 꼭 지정해야 하나는 것이다, 이는 이 회전 작업을 식별하는 데 사용되는 유일한 값이기 때문이다.

```
1  class Card: SKSpriteNode {
2      ...
3       override func touchesEnded(_ touches: Set<UITouch>, with event:
4  UIEvent?) {
5              let dropDown = SKAction.scale(to:
6  CardDisplayScene.cardScale, duration: 0.2)
7              run(dropDown)
8
9              removeAction(forKey: "rotate")
10             let rotate = SKAction.rotate(toAngle: 0, duration: 0)
               run(rotate)
11     }
12
13     ...
14 }
```

10 행은 removeAction 메소드의 toKey 파라미터에 'rotate'라는 키 값을 설정해 액션을 찾아서 제거하게 된다. 그럼 바로 흔들리는 효과는 중단 된다.
11 행은 좀 더 확실하게 하기 위해 우선 동작을 멈춘 후 카드를 0 도로 회전시켜서 이전 상태와 동일하게 바꿔준다.

20-4. 테스트 결과 화면 만들기

각각의 결과 값이 될 새로운 이미지 에셋을 만든다.

새로운 에셋 4 개를 추가하고 각각 '기존에 있던 엣세명_back'이라고 추가해 이름을 지정한다,

```
1  class Card: SKSpriteNode {
2      var front: SKTexture
3      var back: SKTexture
4
5      required init?(coder aDecoder: NSCoder) { //
6          fatalError("init(coder:) has not been implemented")
7      }
8      init(cardName: String) {
9          self.front = SKTexture(imageNamed: cardName)
10         self.back = SKTexture(imageNamed: cardName+"_back")
11         super.init(texture: self.front, color: UIColor.white, size: self.front.size())
12         isUserInteractionEnabled = true
13     }
14     ...
15 }
```

16	

2~3 행을 추가하며 SKTexutre 클래스 타입인 front, back 변수를 추가한다

8~14 행은 가존의 init 내용을 지우고 새로운 변수를 클래스에 추가한다. Init 메서드에서 각각 앞, 뒤면의 텍스처를 생성해 저장하는 용도로 사용한다, 이렇게 초기화하면 앞면은 그대로 나오고 뒷면의 텍스처는 보이지 않지만 만들어져 있는 상태가 된다.

```
1  class Card: SKSpriteNode {
2
3      ...
4          override func touchesEnded(_ touches: Set<UITouch>, with event:
5  UIEvent?) {
6          let dropDown = SKAction.scale(to: CardDisplayScene.cardScale, duration:
7  0.2)
8          run(dropDown)
9
10         removeAction(forKey: "rotate")
11         let rotate = SKAction.rotate(toAngle: 0, duration: 0)
12         run(rotate)
13
14         self.texture = self.back
15
16     }
17     ...
18 }
```

14 행을 추가 한다.

```
1  class Card: SKSpriteNode {
2
3      ...
4          override func touchesEnded(_ touches: Set<UITouch>, with event:
5  UIEvent?) {
6          let dropDown = SKAction.scale(to: CardDisplayScene.cardScale, duration:
7  0.2)
8          run(dropDown)
9
10         removeAction(forKey: "rotate")
11         let rotate = SKAction.rotate(toAngle: 0, duration: 0)
12         run(rotate)
13
           self.texture = self.back
```

```
14
15              let fold = SKAction.scaleX(to: 0.0, duration: 0.4)
16              let unfold = SKAction.scaleX(to: CardDisplayScene.cardScale, duration:
17  0.4)
                run(fold, completion: {
18                      self.texture = self.back
19                              self.run(unfold)
20              })
21      }
22      ...
23  }
24
25
```

16~22 행 추가한다.

16 행, 17 행은 크기를 변경하는 액션이다. 터치했을 때 사용했던 scale 과는 다르게 X 축으로만 크기를 변경하는 scaleX 를 사용한다.

뒤집는 효과를 연출하려면 카드의 너비를 0 으로 만들었다가 다시 원래의 너비(CardDisplayScene.cardScale)로 만들어주면된다.

18 행~20 행은 sequence 메서드를 사용하면 연속으로 액션을 실행할 수 있자만 적절한 시점에 텍스처를 변경할 수 없기 때문에 적용된 효과가 어색하다. 그래서 시퀀스를 사용하지 않고 complete 블럭을 이용해서 텍스터를 변경한 후 unfold 액션을 실행하도록한다.

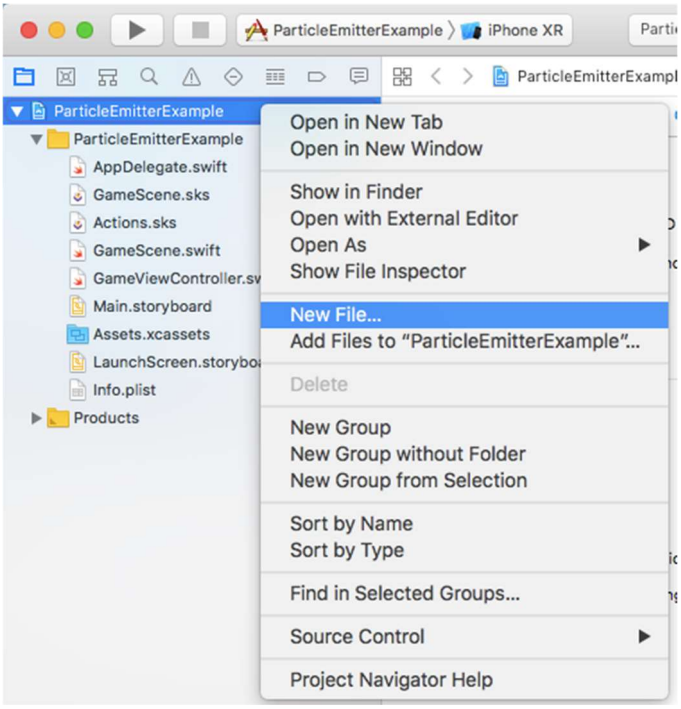
20-4. 특수 효과 적용하기

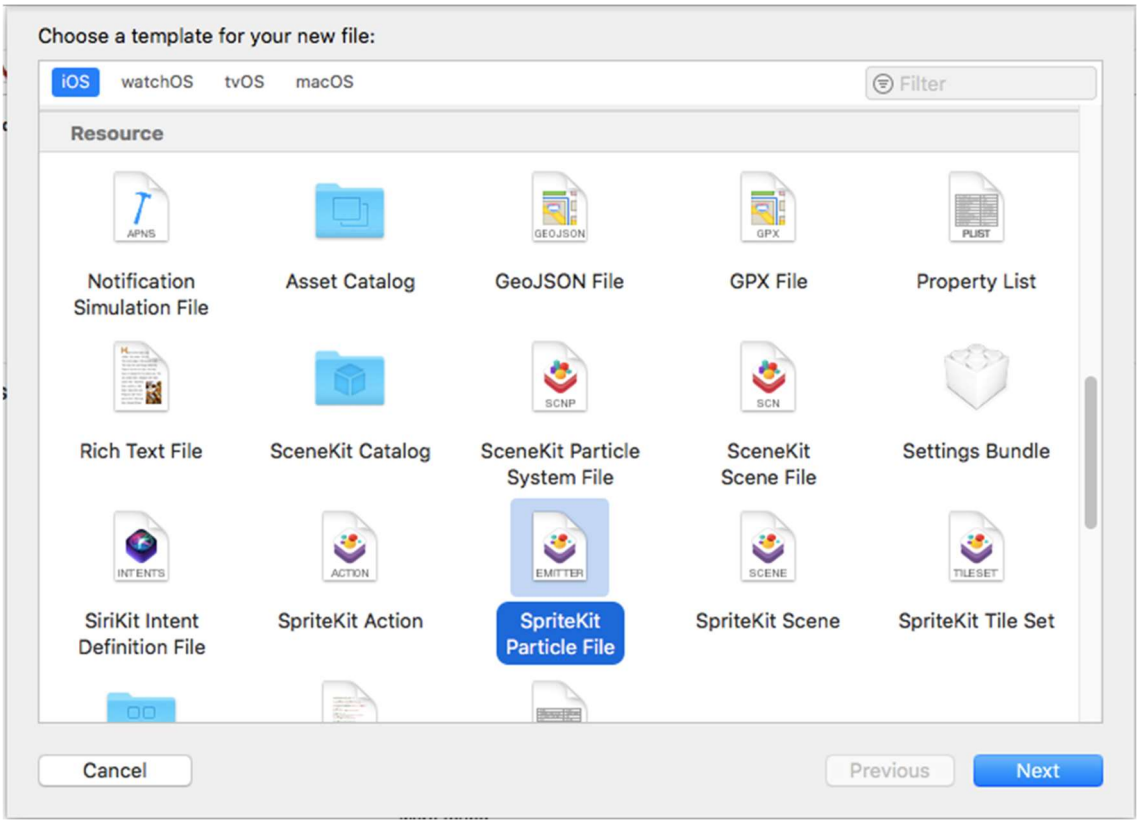
게임을 즐기다 보면 움직임뿐만 아니라 폭발이 일어난다든지, 아이템을 얻었을 때 화려한 폭죽이 터진다든지 하는 등의 다양한 효과를 접할 수 있다. SpriteKit 에도 이러한 효과를 구현할 수 있게 제공되는 파티클 이미터라는 기능이 있다.

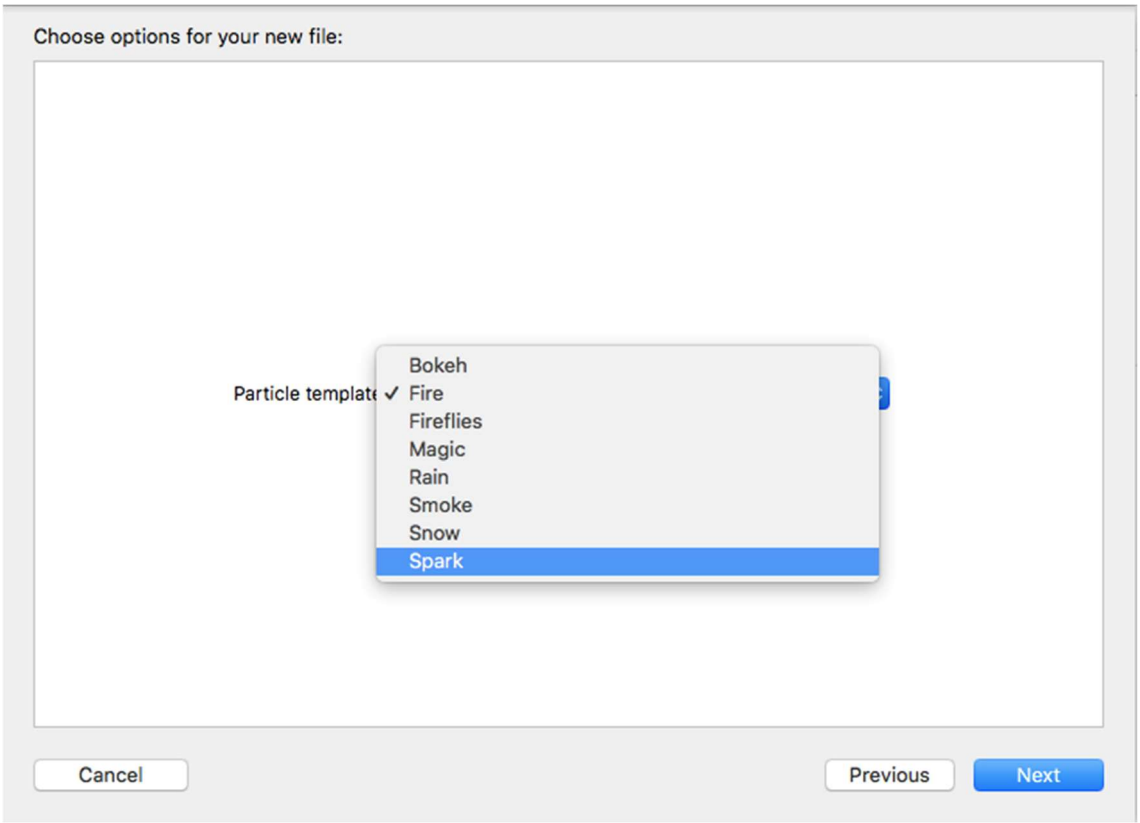
파티클 이미터는 특수 효과를 제공하는 기능으로 이 기능 역시 노드 형태로 사용하며 SKEmitter Node 클래스를 이용해서 다루게 된다.

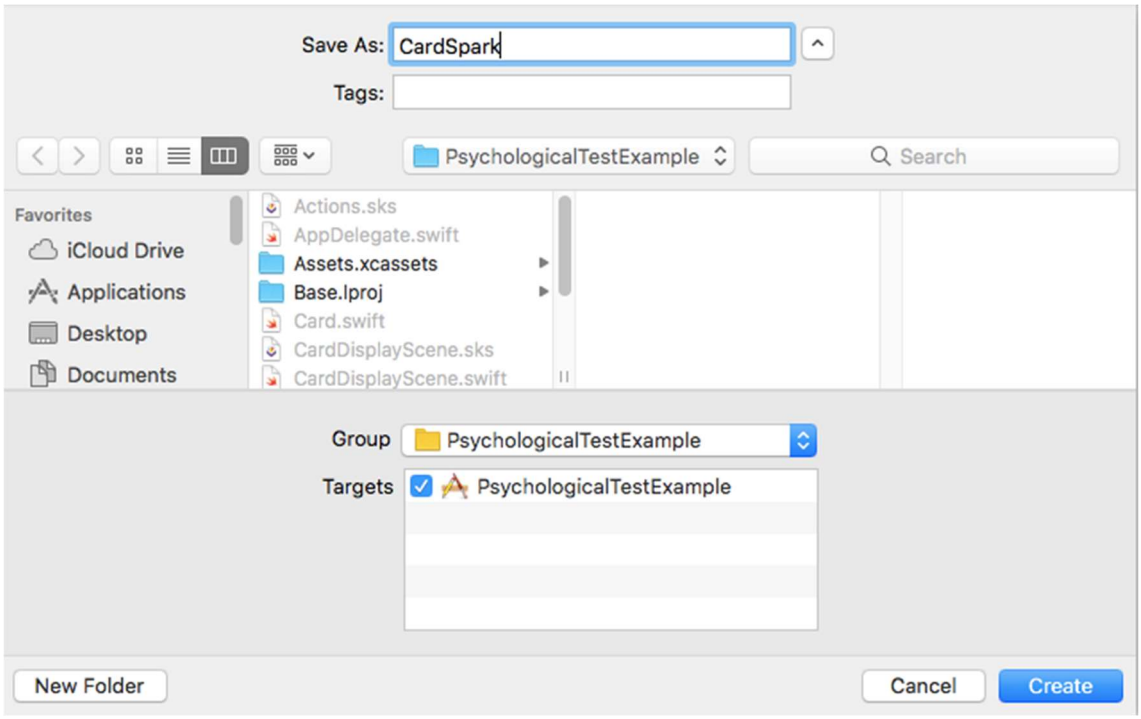
파티클 이미터는 파티클 이미터 편집기를 이용해 만들고 편집해서 사용할 수 있다. 이 편집기 실습을 위해 새로운 프로젝트를 만들어 보겠다.

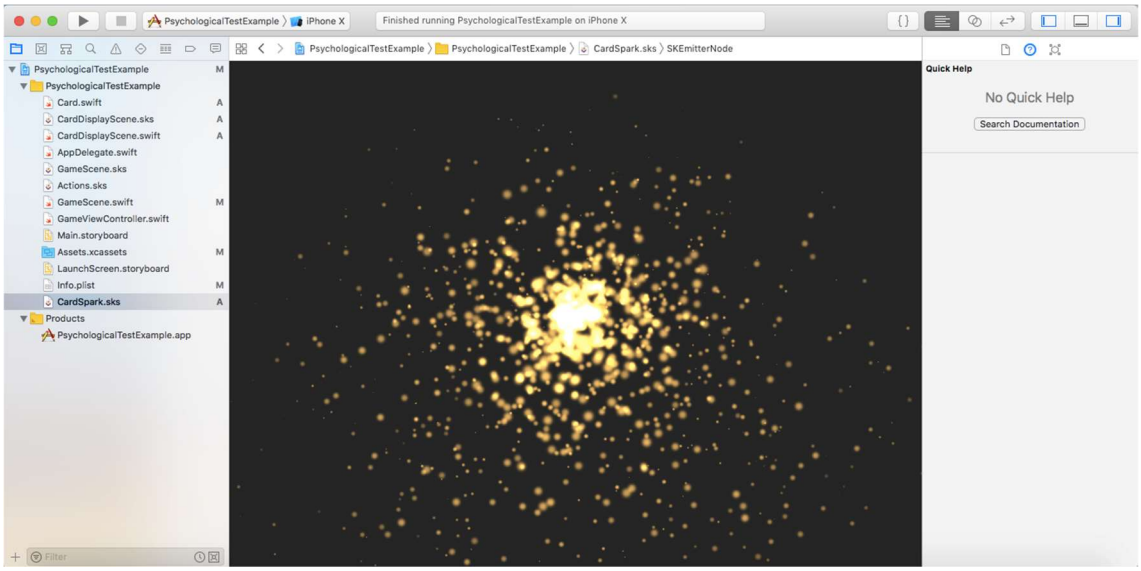
프로젝트에 새로운 파일을 추가한다. Project navigator 에서 [프로젝트 이름]폴더를 마우스 오른쪽 버튼으로 클릭해 바로 가기 메뉴에서 [New File] 을 선택하고 Choose a template for new file 창에서 [iOS] -> [Resource] -> [SpriteKit Particle File]을 선택한 후 <Next>버튼을 클릭한다.




Choose a template for your new file:
iOS
watchOS
tvOS
macOS
Filter
Resource
Notification Simulation File
Asset Catalog
GeoJSON File
GPX File
Property List
Rich Text File
SceneKit Catalog
SceneKit Particle System File
SceneKit Scene File
Settings Bundle
SiriKit Intent Definition File
SpriteKit Action
SpriteKit Particle File
SpriteKit Scene
SpriteKit Tile Set
Cancel
Previous
Next


Choose options for your new file:
Particle template
Bokeh
Fire
Fireflies
Magic
Rain
Smoke
Snow
Spark
Cancel
Previous
Next





파티클 효과는 Project navigator 에 있는 CardSpark.sks 파일에서 조정할 수 있다. 또 CardSpark.sks 파일 아래를 보면 spark.png 라는 새로운 파일이 하나 추가된 것이 보이는데, 이는 스파크 파티클 이미지이다.

파티클 이미터 편집기에서는 간단한 조작으로 효과를 변경할 수 있다. 파티클 파일은 CardSpark.sks 파일을 선택하고 오른쪽 유틸리티 영역에 있는 Atrribute inspector 를 살펴본다.

Card.swift 파일을 선택해 touchesEnded 메서드에서 unfold 액션을 실행하는 self.run(unfold) 코드 아래에 추가한다.

```
1   run(fold, completion: {
2                   self.texture = self.back
3                   self.run(unfold)
4
5                   if let sparkParticlePath: String = Bundle.main.path(forResource:
    "CardSpark", ofType: "sks") {
6                       let sparkParticleNode =
    NSKeyedUnarchiver.unarchiveObject(withFile: sparkParticlePath) as!
    SKEmitterNode
7                       sparkParticleNode.position = CGPoint(x: 0,y: 0)
8
9                       self.addChild(sparkParticleNode)
10                  }
11          })
```

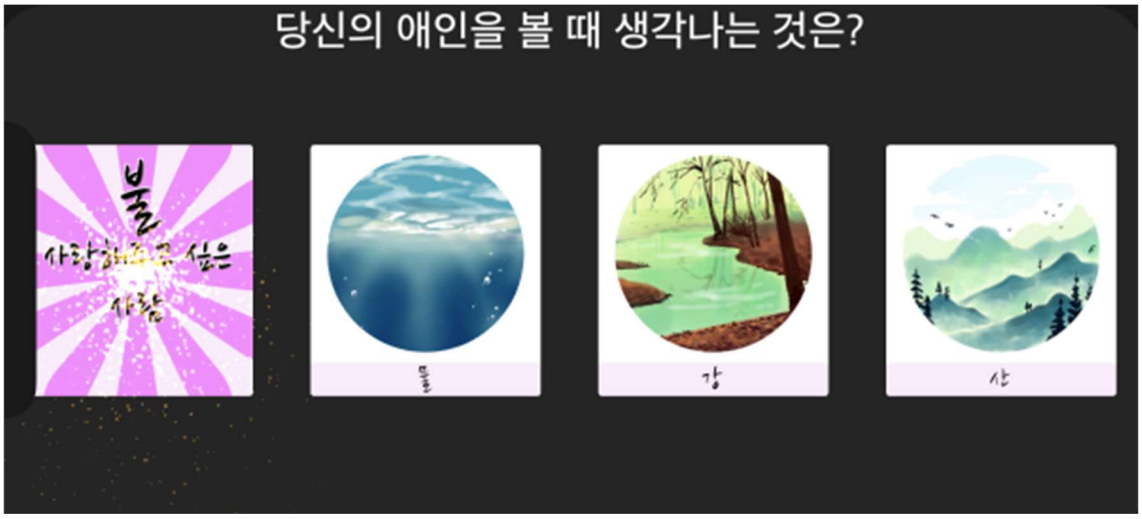
5~10 행 추가

5 행은 Bundle 클래스부터 CardSpark.sks 파일의 해당 리서스 경로를 받아온다.

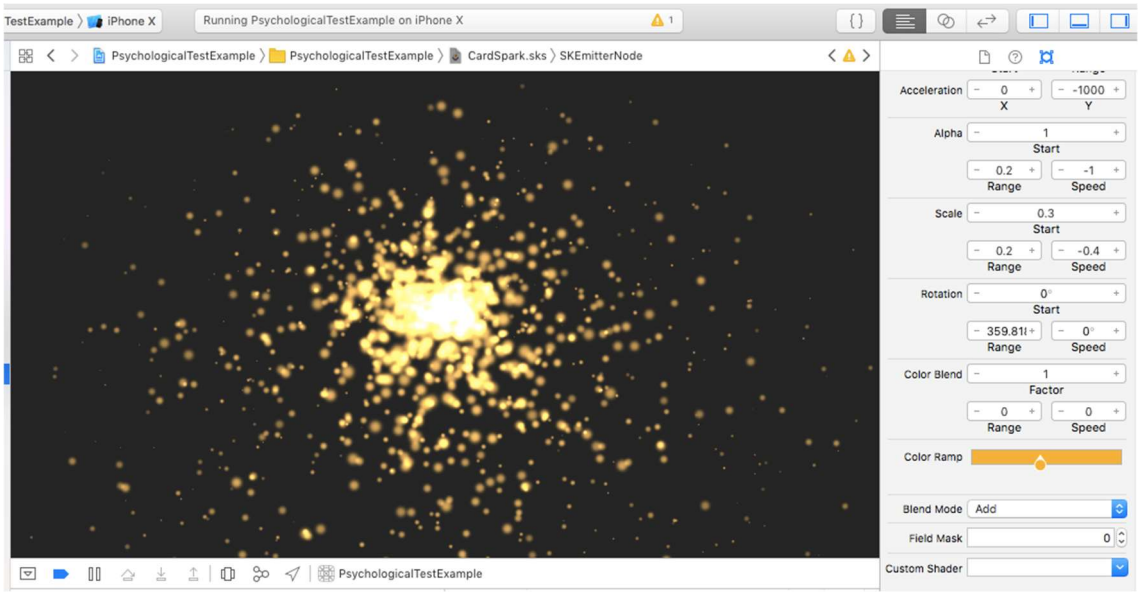
6 행은 NSKeyedUnarchiver 클래스를 이용해서 해당 객체로 만들어준다. 우리가 이용할 객체는 CardSpark.sks 로 부터 얻어온 SKEmitterNode 클래스의 인스턴스이므로 타입 캐스팅까지 해주어야한다.

8~9 행은 파티클 이미터 노드의 위치를 (x:0,y:0)으로 지정하고 카드 자신에게 노드를 추가한다.

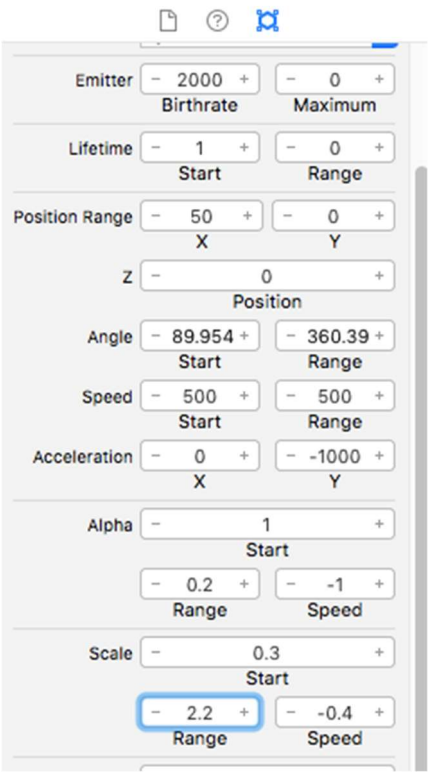
실행 결과를 확인한다.



파티클 이미터가 제대로 추가되는 것을 확인 했으니 유틸리티 영역의 Atrribute inspector 로 이동해서 적절한 효과를 만들어 본다.
파티클 이미터 편집기로 이동해서 적당한 형태로 퍼지는 파티클 효과를 만든다.



앱을 실행해 보면 지정한 효과가 적용된 것을 확인할 수 있다. 하지만 파티클이 아주 명확하게 보이지 않아 아쉬운 느낌이 든다.



파티클이 좀 더 명확하게 보이게 하기 위해 파티클의 크기를 키워본다,
파티클 이미지 편집기에서 [Scale]의 [Range]수치를 '2.2'로 변경한다.

앱을 실행해 보면 파티클이 좀 더 확실히 보이는 것을 확인할 수 있다. 그래도 아직은 파티클이 카드에 가려져 있어 화려한 효과가 확실하게 보이질 않는다. 파티클이 카드 앞에서 터져나오는 모습이 연출될 수 있도록 수정해보자.

Card.swift 파일의 touchesEnded 메서드를 수정한다. Run(fold, completion)메서드에서 카드에 노드를 추가하기 직전에 zPosition 을 '10'으로 변경한다,

```
1   run(fold, completion: {
2                   self.texture = self.back
3                   self.run(unfold)
4
5                   if let sparkParticlePath: String = Bundle.main.path(forResource:
    "CardSpark", ofType: "sks") {
6                       let sparkParticleNode =
    NSKeyedUnarchiver.unarchiveObject(withFile: sparkParticlePath) as!
    SKEmitterNode
7                       sparkParticleNode.position = CGPoint(x: 0,y: 0)
8                       sparkParticleNode.zPosition = 10
9                       self.addChild(sparkParticleNode)
10                  }
11              })
```

8 행을 추가한다.

zPosition 은 웹 프로그램에서 사용하는 z-index 나 포토샵에서 볼수 있는 레이어 순서와 같다.
zPosition 값을 높게 설정할수록 위쪽으로 보이게 된다.

모든 코드의 적용을 마치면 파티클 효과가 카드 앞쪽에서 나타나기 때문에 파티클 효과를 명확하게 확인할 수 있다.

20-5. SNS 에 포스팅하기
iOS 에서 앱에 소셜 네트워크 공유기능을 추가하는 방법은 여러가지가 있다. 그 중 많이 사용하는 방법은 아래와 같다.
1) UIActionController 클래스를 이용해 사용자에게 어떤 동작을 취할 것인지 선택하는 방법
2) Social Framwork 를 이용해 직접적으로 소셜 미디어 API 와 통신하여 공유하는 방법
3) Accounts Framework 를 이용해 현재 기기에 연결될 계정을 통해 공유하는 방법

* UIActionController 클래스
UIActionController 클래스는 iOS 앱에서 자주 볼 수 있는 Activity 기능을 사용하게 해주는 클래스이다. 보통 <공유> 버튼을 터치하면 뷸수 있는 기능이다.

Card.swift 파일을 선택한 후 Card 클래스에 있는 touchesEnded 메서드를 찾아 self.addChild(sprike ParticleNode)코드 아래에 내용을 추가한다.

```
1   let alert = UIAlertController(title: "알림", message: "결과를
    공유하시겠습니까?", preferredStyle: .alert)
2   let actionOK = UIAlertAction(title: "예", style:
    UIAlertAction.Style.default, handler: { (action) -> Void in alert.dismiss(animated:
    true, completion: nil)})
    let actionCancel = UIAlertAction(title: "아니오", style:
3   UIAlertAction.Style.default, handler: { (action) -> Void in alert.dismiss(animated:
    true, completion: nil)})
        alert.addAction(actionOK)
4       alert.addAction(actionCancel)
5
6       self.scene?.view?.window?.rootViewController?.present(alert,
7   animated: true, completion: nil)
8
```

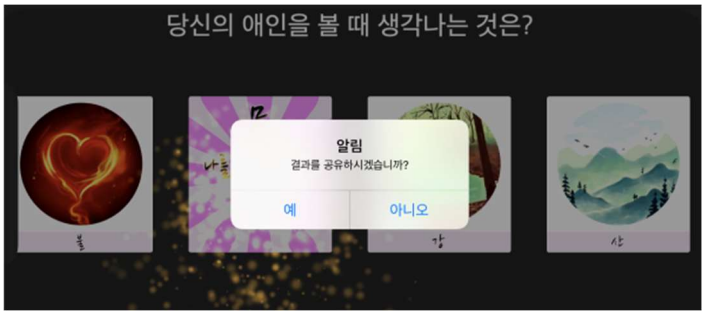
1 행은 UIAlertController 클래스를 이용해 결과를 공유할 것인지의 여부를 묻는 창을 열 것이다. 기본적으로 UIAlertController 클래스는 title 파라미터와 message 파라미터에 제목과 메시지를 지정하는 형태이다.

2 행은 "예"를 선택했을 때 액션을 추가해서 선택에 따라 어떠한 가능을 수행하도록 한다. 선택기능을 수행하기 위해서는 액션을 추가하고 버튼을 터치했을 때 수행되는 핸들러 메세지를 추가해주어야 한다.

3 행은 "아니오"를 선택했을 때 액션을 추가해서 선택에 따라 어떠한 기능을 수행하도록 한다,

4~5 행은 해당 액션을 실행하게 한다.

8 행은 알림 창을 열기 위해서는 현재 뷰 컨트롤러의 present 메서드를 사용해야 하는데 코드가 추가된 위치가 카드이므로 카드로부터 뷰 컨트롤러를 찾는 과정이 복잡하다. 토드 처럼 scene->view -> window -> rootViewController 의 순서로 찾는 과정을 거친다. 다행히 찾아낼 수 있지만 이러한 방법이 불가능한 상황이라면 카드 객체를 생성할 때 뷰 컨트롤러를 미리 지정해두는 것도 하나의 방법이다.



그런데 카드를 선택하고 바로 '결과를 공유하시겠습니까?'라는 메세지 창이 나타나면 좀 당황스러울것이다.
파티클 효과를 다 감상하기도 전에 알림창이 화면을 가려버릴 수 있기 때문이다. 이를 수정해 보겠다.

```
1  class Card: SKSpriteNode {
2        var front: SKTexture
3        var back: SKTexture
4        var touched = false
5
6  ...
7  }
```

touched 라는 변수는 현재 카드가 뒤집혀 있는지의 여부를 저장히기 위한 변수이다. 이를 통해 한 번 뒤집힌 카드에는 뒤집히는 효과가 다시 적용되지 않도록 하고 다시 한번 터치했을 때 공유하기 알림창을 나타낼 것이다.
touchesEnded 메서드를 수정한다.

```
1  override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
2            let dropDown = SKAction.scale(to: CardDisplayScene.cardScale, duration:
3  0.2)
4            run(dropDown)
5
6            removeAction(forKey: "rotate")
7            let rotate = SKAction.rotate(toAngle: 0, duration: 0)
8            run(rotate)
9
10           self.texture = self.back
11
12           if !self.touched {
13                 let fold = SKAction.scaleX(to: 0.0, duration: 0.4)
14                 let unfold = SKAction.scaleX(to: CardDisplayScene.cardScale,
15 duration: 0.4)
16
17                 run(fold, completion: {
18                       self.texture = self.back
```

```
19                      self.run(unfold)
20
21                      if let sparkParticlePath: String = Bundle.main.path(forResource:
22  "CardSpark", ofType: "sks") {
23                          let sparkParticleNode =
24  NSKeyedUnarchiver.unarchiveObject(withFile: sparkParticlePath) as!
25  SKEmitterNode
26                          sparkParticleNode.position = CGPoint(x: 0,y: 0)
27                          sparkParticleNode.zPosition = 10
28                          self.addChild(sparkParticleNode)
29                      }
30                      self.touched = true
31                  })
32              } else {
33                  let alert = UIAlertController(title: "알림", message: "결과를
34  공유하시겠습니까?", preferredStyle: .alert)
35
36                  let actionOK = UIAlertAction(title: "예", style:
37  UIAlertAction.Style.default, handler: { (action) -> Void in alert.dismiss(animated:
38  true, completion: nil)})
39                  let actionCancel = UIAlertAction(title: "아니오", style:
40  UIAlertAction.Style.default, handler: { (action) -> Void in alert.dismiss(animated:
41  true, completion: nil)})
42
43                  alert.addAction(actionOK)
44                  alert.addAction(actionCancel)
45
46                      self.scene?.view?.window?.rootViewController?.present(alert,
47  animated: true, completion: nil)            }
48  }
49
50
51
52
```

13 행은 touched 변수가 false 가 아니면(현재 카드가 뒤집혀 있는 상태가 아니라면) 카드를 뒤집고 파티클 노드를 추가해 효과를 나타난다.
34 행은 카드가 뒤집혀 있는 상태라면 뒤집히는 효과는 적용하지 않고 바로 공유하기 알림창을 나타내도록 한다.

카드를 하나 선택하면 뒤집히고 나서 파티클 이미터 효과까지 제대로 나타난다. 하지만 한 번 더 터치하면 뒤집히는 효과가 나타나지 않는다

이제 적당한 타이밍에 알림창이 나타나게 만들었으니 '예' 버튼을 터치하면 Activity 창이 나타나도록 UIActivityViewController 클래스를 이용해보자.

먼저 Activity 창이 나타나도록 하는 shareHandler 메서드의 코드를 추가한다, Card.swift 클래스 안에

```
1  func shareHandler(_ action: UIAlertAction!, alert: UIAlertController) {
2          alert.dismiss(animated: true, completion: nil)
3          let shareTxt = "당신은 나에게 이런 의미입니다."
4          let shareImg = UIImage(cgImage: (self.texture?.cgImage)!())
5          let shareItems = [shareTxt, shareImg] as [Any]
6          let activityController = UIActivityViewController(activityItems: shareItems,
7  applicationActivities: nil)
8                  self.scene?.view?.window?.rootViewController?
9  .present(activityController, animated: true, completion: nil)
10 }
11
12
13
14
```

3 행은 알림창을 사라지게 하는 코드는 이전과 동일하다.

4~5 행은 공유할 내용과 이미지를 지정한다. 이번에 우리가 전달할 아이템은 텍스트와 이미지이다, 텍스트는 적절한 문장을 입력해주고 이미지는 현재 카드의 텍스처 이미지를 찾아서 전달할 것이므로 self.texture?.cgImage 를 이용한다. 현재 카드의 텍스처에서 CGImage 객체를 얻어올 수 있다.

8 행은 이렇게 준비된 내용을 배열 형태로 묶어서 파라미터로 전달하면 UIActivityViewController 의 인스턴스가 생성된다.

9 행은 UIActivityViewController 클래스의 인스턴스를 만들 때는 공유할 내용을 아이템으로 전달해야 한다, 기본적으로는 어떤 오브젝트라도 아이템으로 전달할 수 있다.

12 행은 UIActivityViewController 클래스의 인스턴스를 만들어서 present 메서드를 통해 화면에 보여주는 과정은 어느 뷰 컨트롤러를 다를 때와 다르지 않다.

마지막으로 touchesEnded 메서드를 수정한다.

actionOK 상수 부분의 handler 파라미터를 변경한다.

```
   else {
```

```
                let alert = UIAlertController(title: "알림", message: "결과를
공유하시겠습니까?", preferredStyle: .alert)
                let actionOK = UIAlertAction(title: "예", style:
UIAlertActionStyle.default, handler: { (action) -> Void in self.shareHandler(action,
alert: alert)})
                let actionCancel = UIAlertAction(title: "아니오", style:
UIAlertActionStyle.default, handler: { (action) -> Void in alert.dismiss(animated: true,
completion: nil)})

                alert.addAction(actionOK)
                alert.addAction(actionCancel)

                self.scene?.view?.window?.rootViewController?.present(alert,
animated: true, completion: nil)
}
```

.