the queue, it would be ideal to cancel the existing ones. This is why `collectionView(_:cancelPrefetchingForItemsAt:)` exists. In the case described, it would get called with the indexPaths that are no longer required, allowing you to cancel operations and free resources.

Add the following in your `UICollectionViewDataSourcePrefetching` extension:

```
func collectionView(_ collectionView:
  UICollectionView, cancelPrefetchingForItemsAt indexPaths:
  [IndexPath]) {
  for indexPath in indexPaths {
    if let dataLoader = loadingOperations[indexPath] {
      dataLoader.cancel()
      loadingOperations.removeValue(forKey: indexPath)
    }
  }
}
```

If a loading operation exists for a passed `indexPath`, this code cancels it and then deletes it from `loadingOperations`.

Build and run, then scroll around a bit. You won't notice any difference in behavior, because at most unneeded operations will be canceled.

Even with a light weight application with limited content like EmojiRater, the benefits of prefetching are evident. For apps loading large collections that require a lot of time to load, this feature will have an enormous impact!

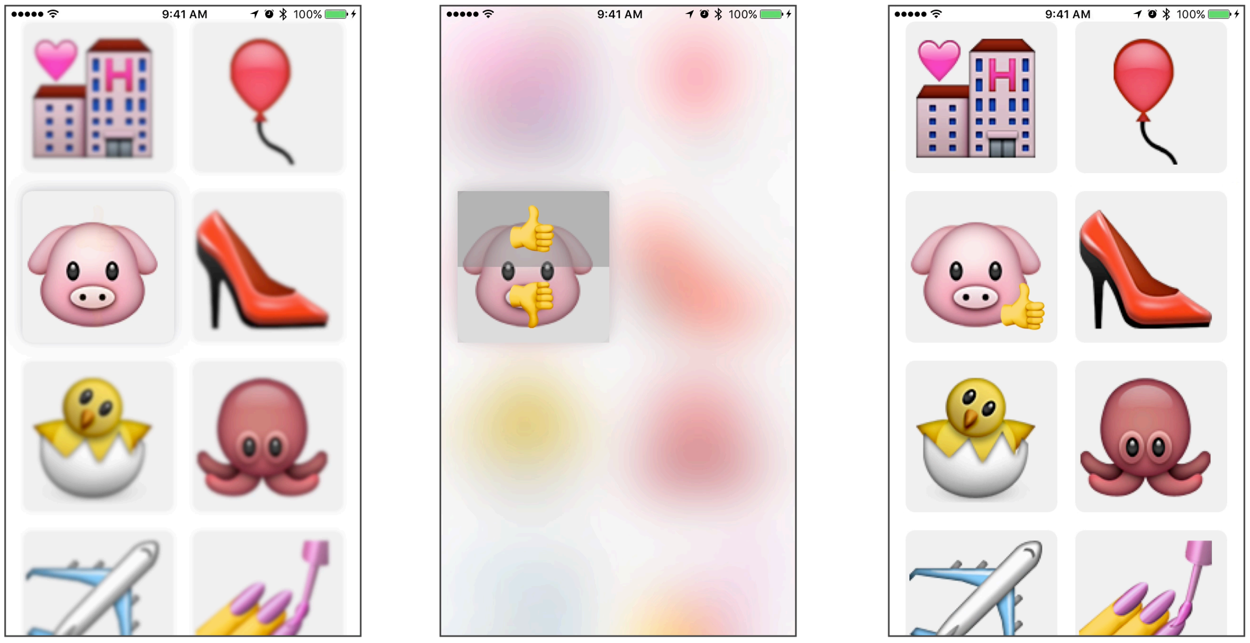
UIPreviewInteraction [Theory]

iOS 9 introduced 3D Touch along with an interaction you know and love - Peek and Pop. The Peek (known as Preview to the API) provides a preview of a destination controller, while the Pop (Commit) navigates to the controller. While you can control the content and quick actions displayed with Peek and Pop, you can't customize the look of the transition or how users interact with it.

In iOS 10, the all new **UIPreviewInteraction** API allows you to create custom preview interactions similar in concept to Peek and Pop. Generically, a preview interaction consists of up to two interface states accessed by steadily increasing pressure of a 3D Touch. As each state completes, unique haptic feedback is provided to signal this to the user.

These interactions are not limited to navigation, as Peek and Pop is. The Preview state can draw or fade in any interface you like. When a user Commits a touch on that interface, you can can run whatever code you like in the callback.

Preview is the first state, and is where you'd animate your content and interface into view. The Commit state occurs next, and is where you'd allow interaction with the presented content. Below is an example of the two states followed by the outcome you'll implement with EmojiRater.



Preview State Commit State Post Commit

The preview state slowly fades out the background and focuses on the selected cell, finally placing thumb controls over the emoji. In the commit state, moving your finger will toggle selection between the two things. Once you press deeper to commit the vote, the interaction fades away and the vote is reflected.

To implement, you need to configure a `UIPreviewInteractionDelegate` which receives messages as progress and state transitions occur. Let's take a look at the protocol methods to get a bit more clarity on how it works:

- **previewInteractionShouldBegin(_:)** is called when 3D Touch kicks off a preview. This is where you'd start preparations for presenting the Preview.
- **previewInteraction(_:didUpdatePreviewTransition:ended:)** is called as the preview state progresses. It receives a value from 0.0 to 1.0 representing the user's progress through the state based on Apple's algorithm that determines intent largely from pressure. The `ended` boolean switches to `true` when the value reaches 1.0 and the preview state completes.
- **previewInteractionDidCancel(_:)** is called when the preview is canceled - either by the user removing their finger, or from an interruption like a phone call. When this message is received, you need to gracefully dismiss the preview.
- **previewInteraction(_:didUpdateCommitTransition:ended:)** is called as the commit state progresses. It works identically to its preview counterpart in terms of the parameters it receives. When it ends, this is your cue to take action on whatever the user interacted with.

The starter project for this section is located in the folder titled **previewInteraction-starter**. Open **EmojiRater.xcodeproj** and get ready to be strangely judgmental about emojis! :]

Implement UIPreviewInteractionDelegate [Instruction]

TODO: may break this up into subsections

Open **EmojiCollectionViewController.swift** and add the following extension at the end of the file:

```
extension EmojiCollectionViewController: UIPreviewInteractionDelegate {
  func previewInteraction(_ previewInteraction:
    UIPreviewInteraction, didUpdatePreviewTransition
    transitionProgress: CGFloat, ended: Bool) {
    print("Preview: \(transitionProgress), ended: \(ended)")
  }

  func previewInteractionDidCancel(_ previewInteraction:
    UIPreviewInteraction) {
    print("Canceled")
  }
}
```

`EmojiCollectionViewController` now adopts the `UIPreviewInteractionDelegate` protocol and implements its two required methods. For now, you're simply logging the calls so you can get a better understanding of how these methods are used.

Build and run. 3D touch one of the cells and watch the console as you vary your touch pressure. It should look something like this:

```
Preview: 0.0, ended: false
Preview: 0.0970873786407767, ended: false
Preview: 0.184466019417476, ended: false
Preview: 0.271844660194175, ended: false
Preview: 0.330097087378641, ended: false
Preview: 0.378640776699029, ended: false
Preview: 0.466019417475728, ended: false
Preview: 0.543689320388349, ended: false
Preview: 0.631067961165048, ended: false
Preview: 0.747572815533981, ended: false
Preview: 1.0, ended: true
Canceled
```

The `transitionProgress` starts at 0.0, and increases to 1.0 as you increase pressure. `ended` remains false here, because Once you remove your finger, you'll see *Canceled* indicating `previewInteractionDidCancel(_:)` was called.

- Implement delegate methods with prints for status testing
- add preview mode overlay with rating emoji over cells
- commit implementation - obtain emoji rating and update datastore, update cell to show new rating