Go Live with ReplayKit

Session 601

Ben Harry Software Engineer
Edwin Iskandar Software Engineer

ReplayKit

Record app visuals and audio

Record microphone input

Share recordings

Simple API

ReplayKit

HD quality

- Low performance impact
- Minimal power usage

Privacy safeguards

- User consent prompt
- Recording excludes system UI

Available since iOS 9

ReplayKit

NEW

ReplayKit

NEW

Apple TV support

ReplayKit

NEW

Apple TV support

Live broadcasting

ReplayKit

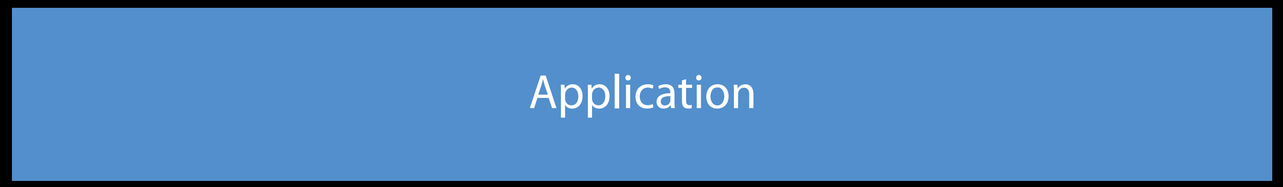
NEW

Apple TV support

Live broadcasting

Expanded commentary options

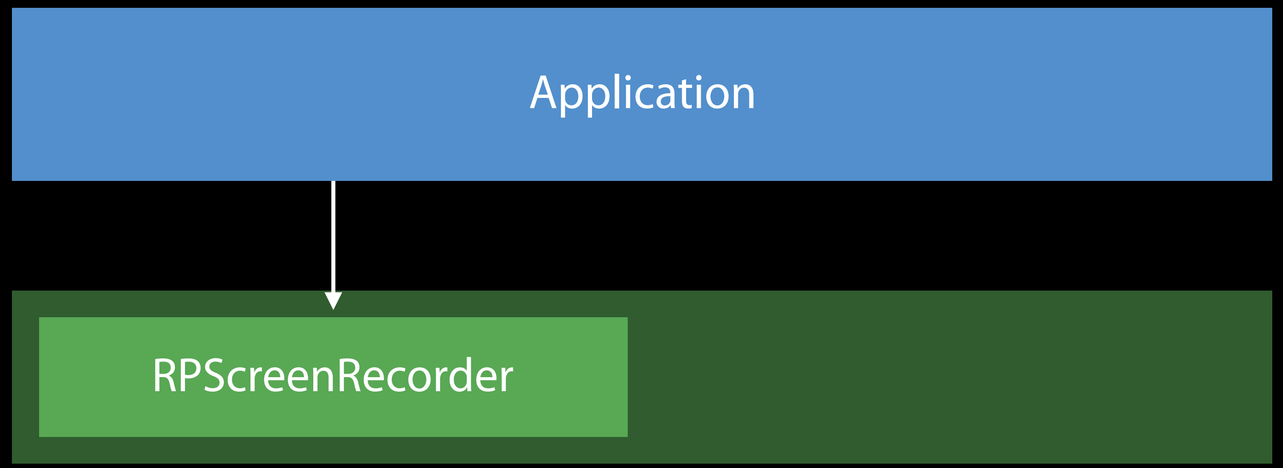
ReplayKit Architecture



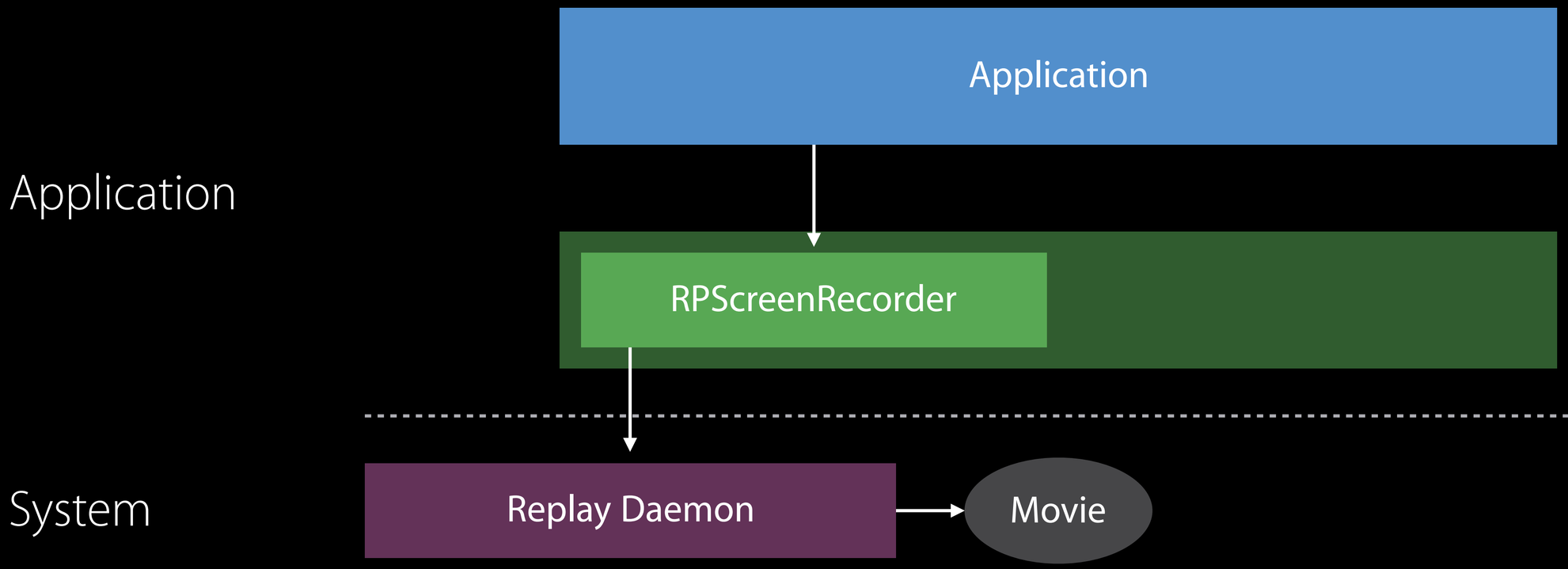
Application

ReplayKit Architecture

Application

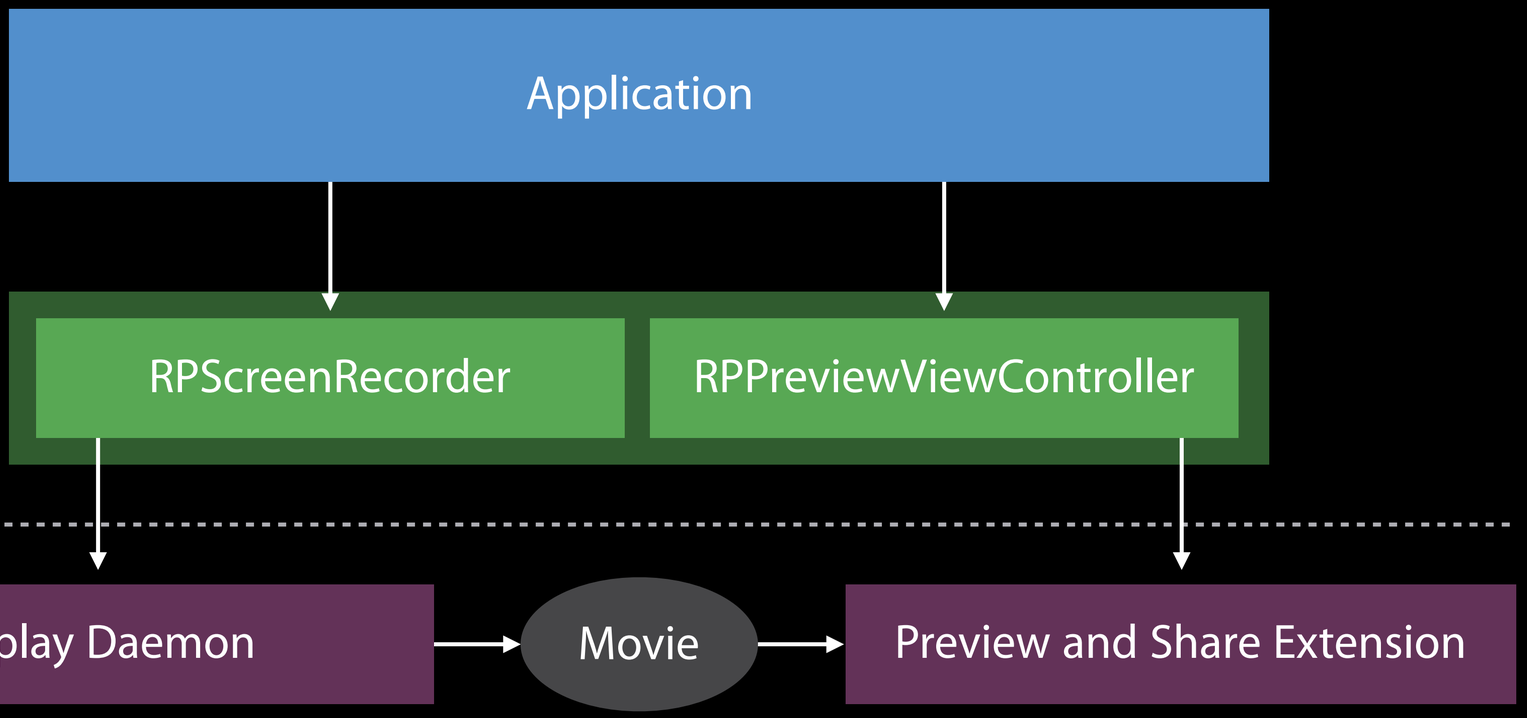


ReplayKit Architecture



ReplayKit Architecture

Application



System

Classes and Protocols

Classes and Protocols

`RPScreenRecorder`

- Start, stop, discard recording
- Check availability to record

`RPScreenRecorderDelegate`

- Availability changes
- Recording stops

Classes and Protocols

`RPScreenRecorder`

- Start, stop, discard recording
- Check availability to record

`RPScreenRecorderDelegate`

- Availability changes
- Recording stops

`RPPreviewViewController`

- Preview the recording
- Edit and trim (iOS)
- Share

`RPPreviewViewControllerDelegate`

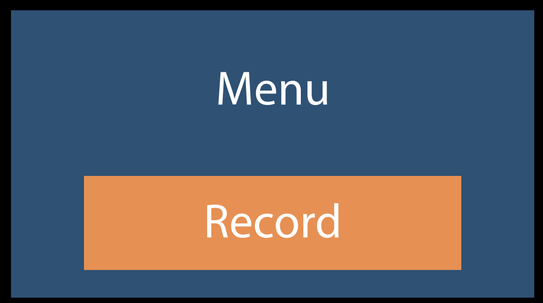
- Finished with preview user interface

Demo

ReplayKit on Apple TV

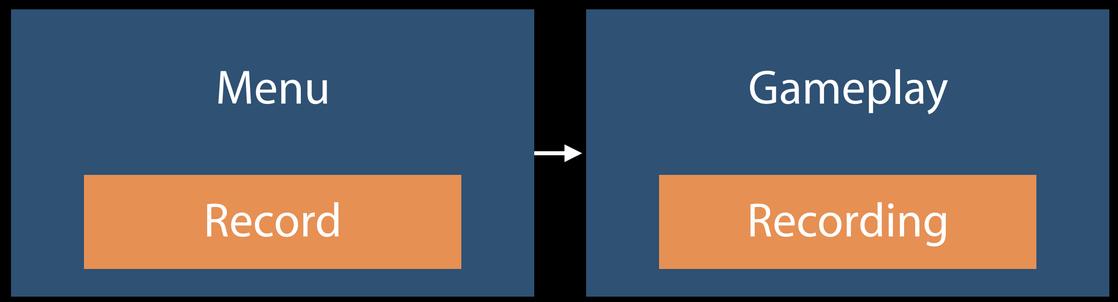
ReplayKit on Apple TV

From beginning to end



ReplayKit on Apple TV

From beginning to end

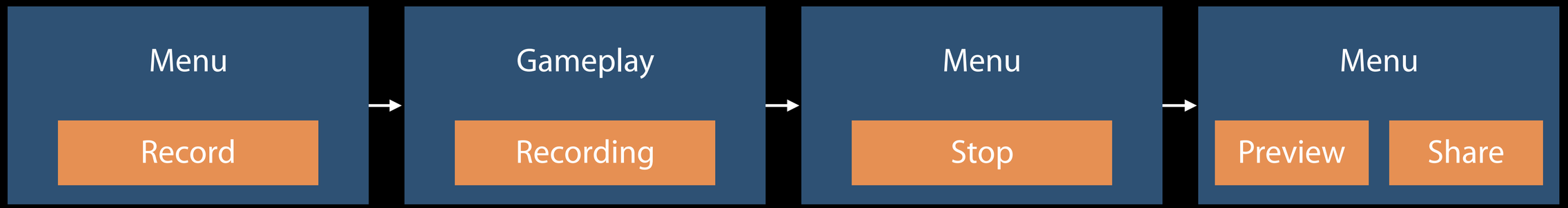


ReplayKit on Apple TV

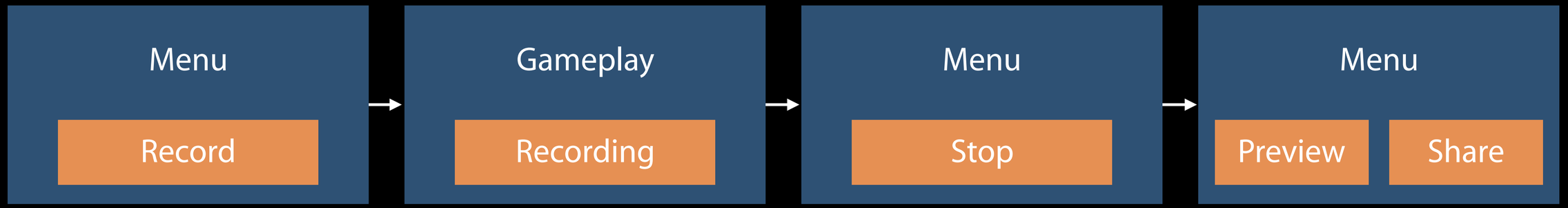
From beginning to end

ReplayKit on Apple TV

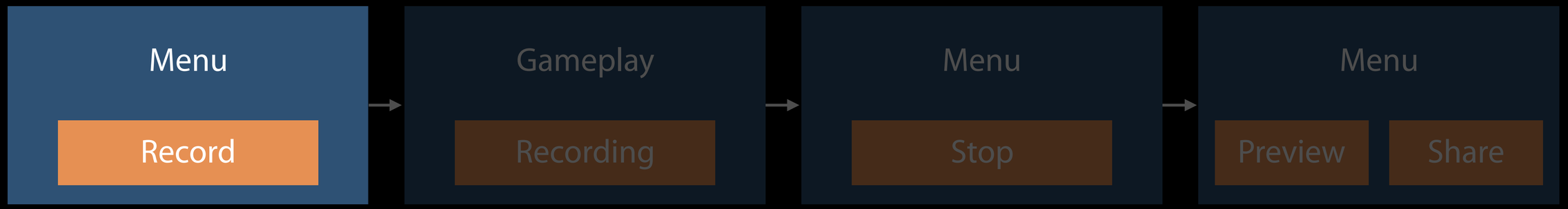
From beginning to end



Start Recording

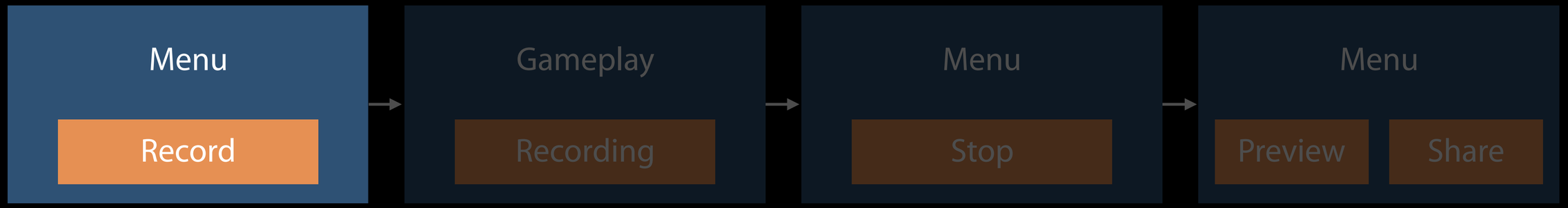


Start Recording



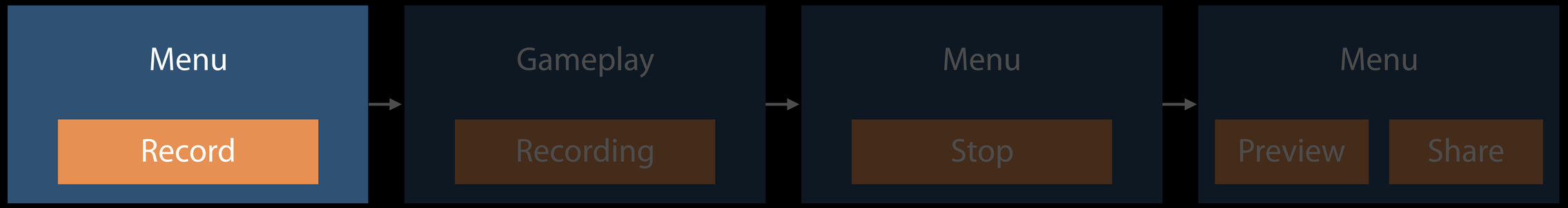

```
func didPressRecordButton() {
    let sharedRecorder = RPScreenRecorder.shared()

    sharedRecorder.startRecording { error in

        if error == nil {
            self.showIndicatorView(text: "Recording")
        }
    }
}
```

Start Recording



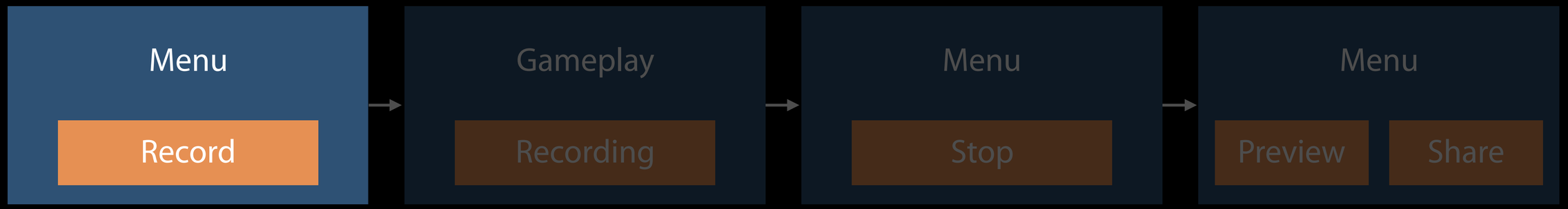

```
func didPressRecordButton() {
    let sharedRecorder = RPScreenRecorder.shared()

    sharedRecorder.startRecording { error in

        if error == nil {
            self.showIndicatorView(text: "Recording")
        }
    }
}
```

Start Recording



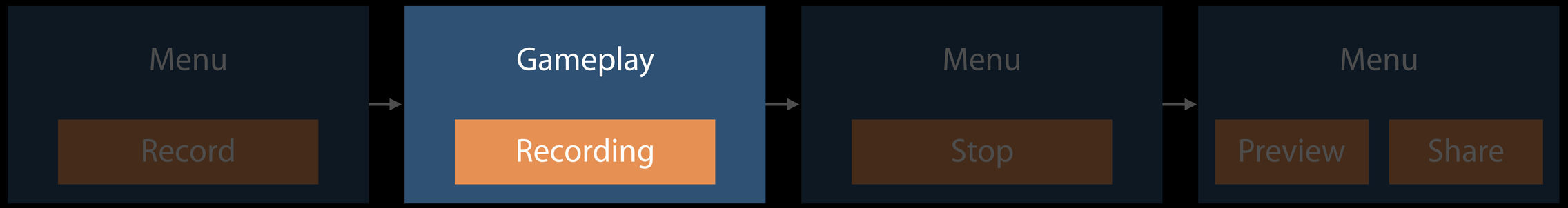

```
func didPressRecordButton() {
    let sharedRecorder = RPScreenRecorder.shared()

    sharedRecorder.startRecording { error in

        if error == nil {
            self.showIndicatorView(text: "Recording")
        }
    }
}
```

Start Recording

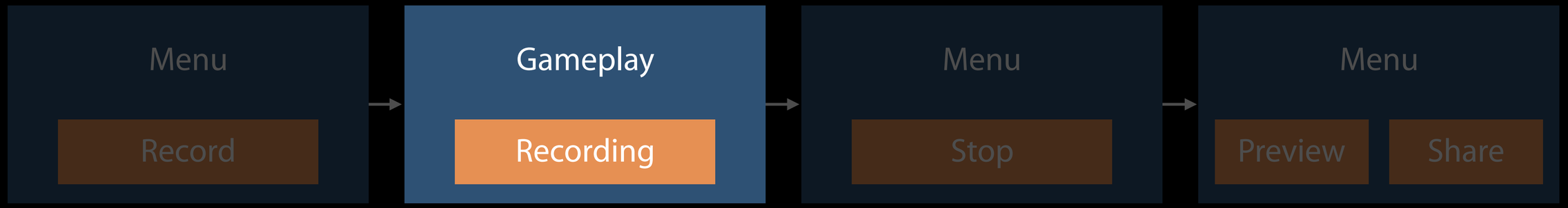



```
func didPressRecordButton() {
    let sharedRecorder = RPScreenRecorder.shared()

    sharedRecorder.startRecording { error in

        if error == nil {
            self.showIndicatorView(text: "Recording")
        }
    }
}
```

Excluding UI

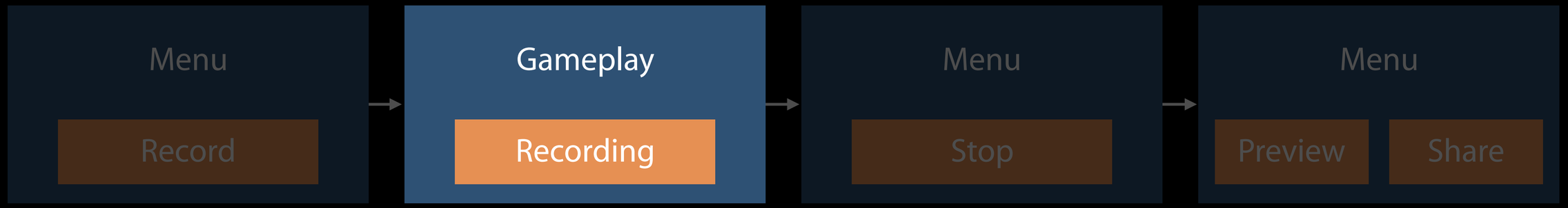


Excluding UI



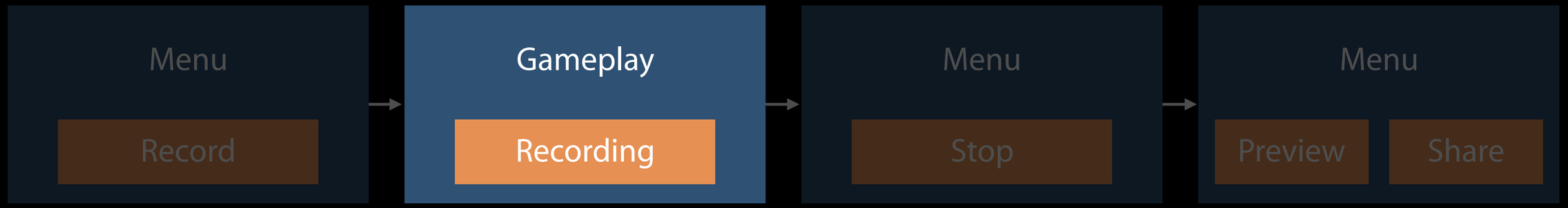

```
func showIndicatorView(text: String) {

    recordingIndicatorWindow = UIWindow(frame: UIScreen.main().bounds)
    recordingIndicatorWindow?.isHidden = false
    recordingIndicatorWindow?.backgroundColor = UIColor.clear()
    recordingIndicatorWindow?.isUserInteractionEnabled = false

    let indicatorView = IndicatorView(text: text)
    recordingIndicatorWindow?.addSubview(indicatorView)
}
```

Excluding UI



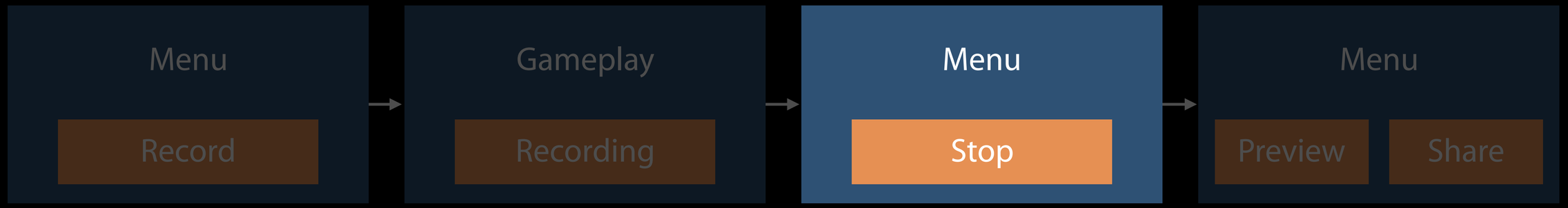

```
func showIndicatorView(text: String) {

    recordingIndicatorWindow = UIWindow(frame: UIScreen.main().bounds)
    recordingIndicatorWindow?.isHidden = false
    recordingIndicatorWindow?.backgroundColor = UIColor.clear()
    recordingIndicatorWindow?.isUserInteractionEnabled = false

    let indicatorView = IndicatorView(text: text)
    recordingIndicatorWindow?.addSubview(indicatorView)
}
```

Excluding UI

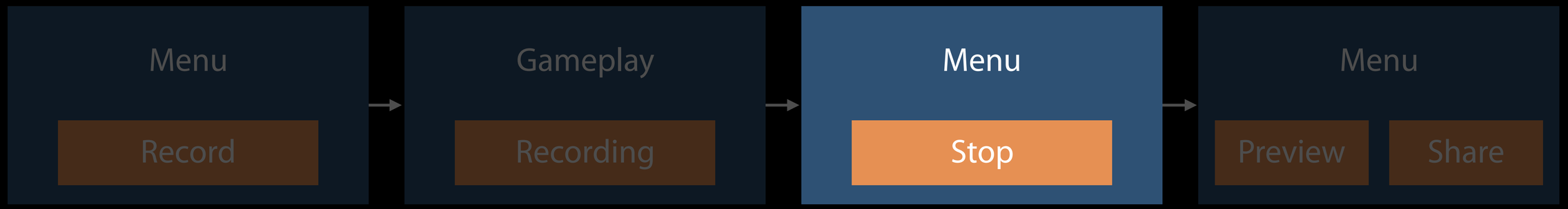



```
func showIndicatorView(text: String) {

    recordingIndicatorWindow = UIWindow(frame: UIScreen.main().bounds)
    recordingIndicatorWindow?.isHidden = false
    recordingIndicatorWindow?.backgroundColor = UIColor.clear()
    recordingIndicatorWindow?.isUserInteractionEnabled = false

    let indicatorView = IndicatorView(text: text)
    recordingIndicatorWindow?.addSubview(indicatorView)
}
```

Stop Recording

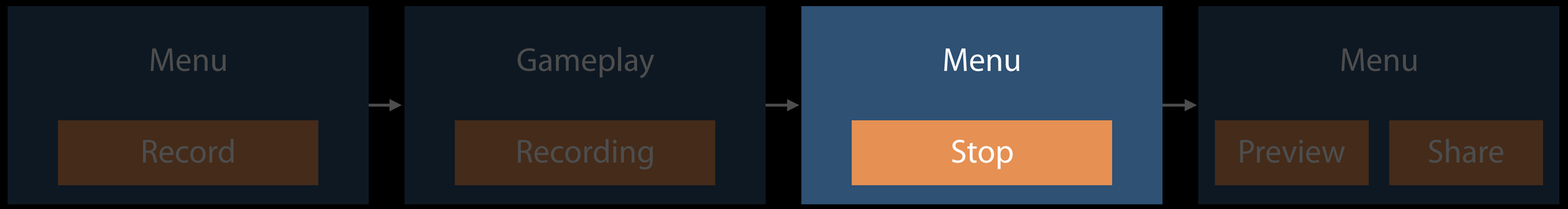


Stop Recording



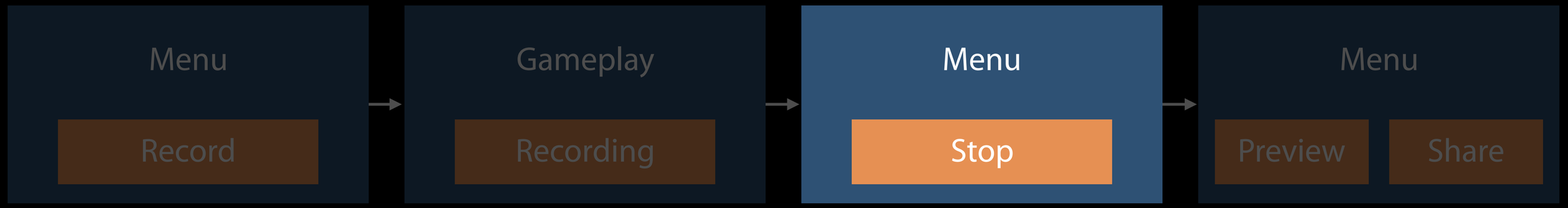

```
func didPressStopButton() {

    sharedRecorder.stopRecording { previewViewController, error in

        self.hideIndicatorView()

        if error == nil {
            self.previewViewController = previewViewController
            self.previewViewController?.previewControllerDelegate = self
        }
    }
}
```

Stop Recording



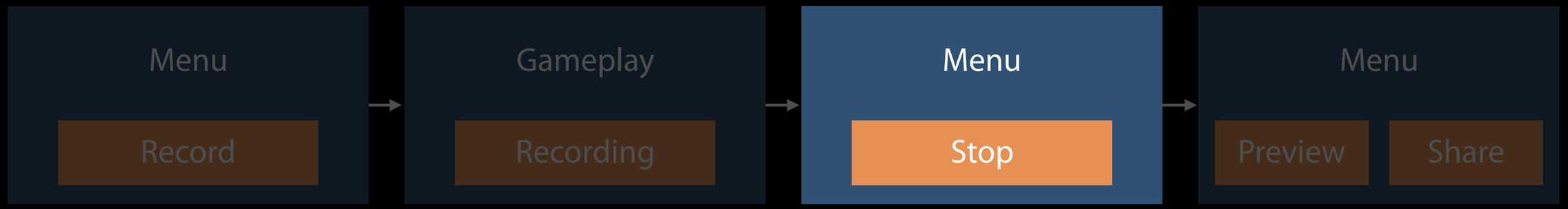

```
func didPressStopButton() {

    sharedRecorder.stopRecording { previewViewController, error in

        self.hideIndicatorView()

        if error == nil {
            self.previewViewController = previewViewController
            self.previewViewController?.previewControllerDelegate = self
        }
    }
}
```

Stop Recording



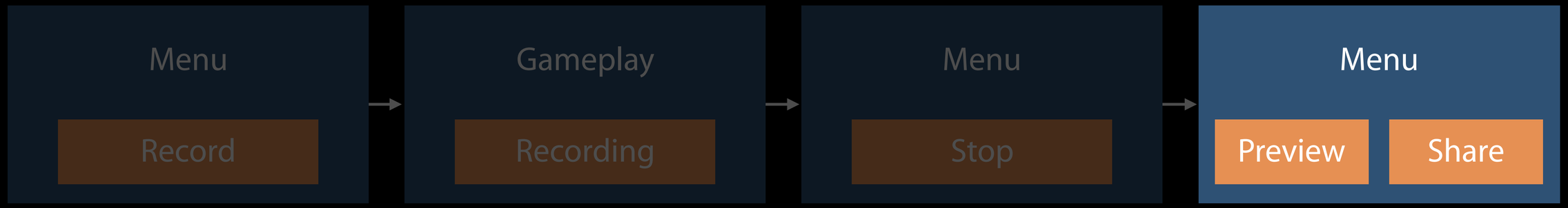

```
func didPressStopButton() {

    sharedRecorder.stopRecording { previewViewController, error in

        self.hideIndicatorView()

        if error == nil {
            self.previewViewController = previewViewController
            self.previewViewController?.previewControllerDelegate = self
        }
    }
}
```

Stop Recording

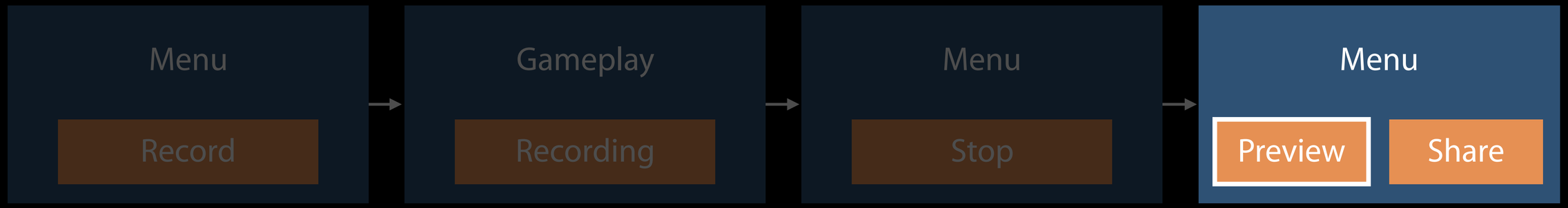



```
func didPressStopButton() {

    sharedRecorder.stopRecording { previewViewController, error in

        self.hideIndicatorView()

        if error == nil {
            self.previewViewController = previewViewController
            self.previewViewController?.previewControllerDelegate = self
        }
    }
}
```

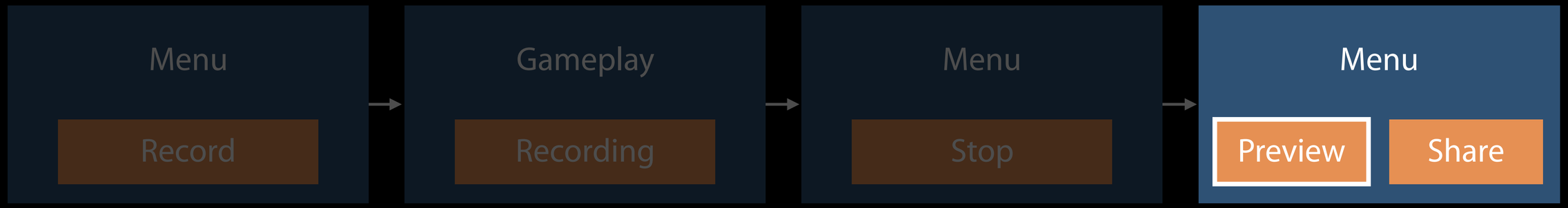
Preview Recording



Preview Recording

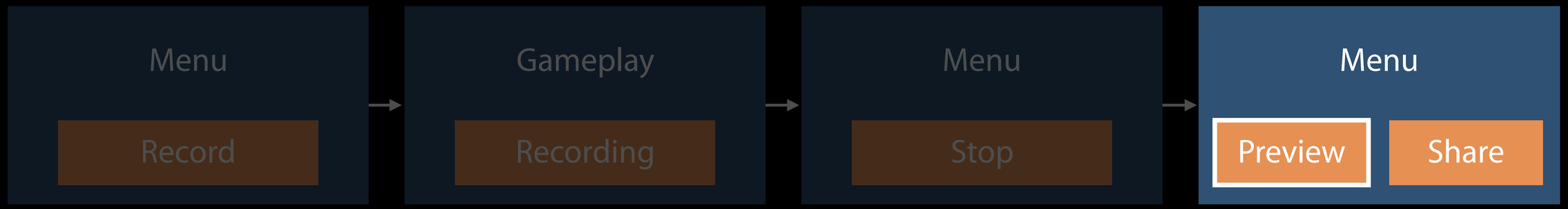


Preview Recording




```
// RPPreviewViewController
public var mode: RPPreviewViewControllerMode
```

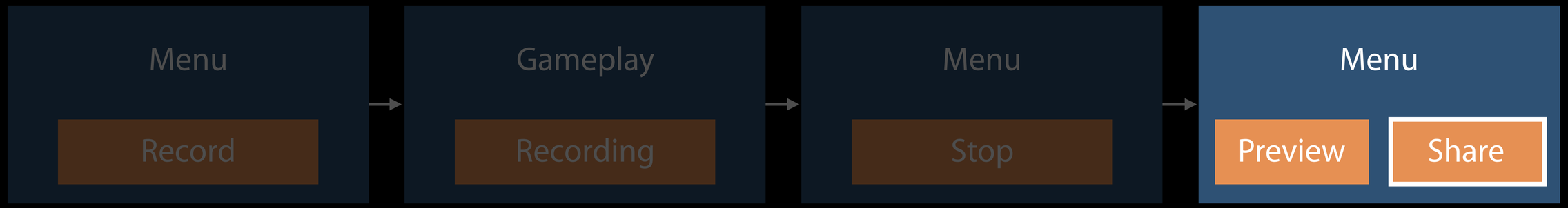
Preview Recording



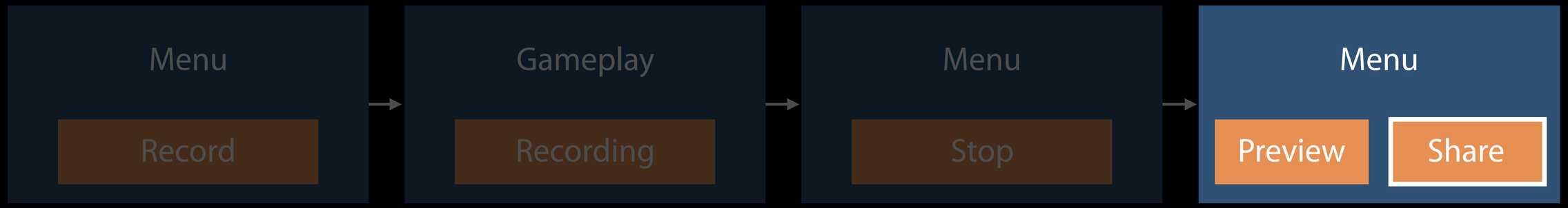

```
// RPPreviewViewController
public var mode: RPPreviewViewControllerMode
```

```
func didPressPreviewButton() {
    if let preview = previewViewController {
        preview.mode = .preview
        self.present(preview, animated: true)
    }
}
```

Share Recording

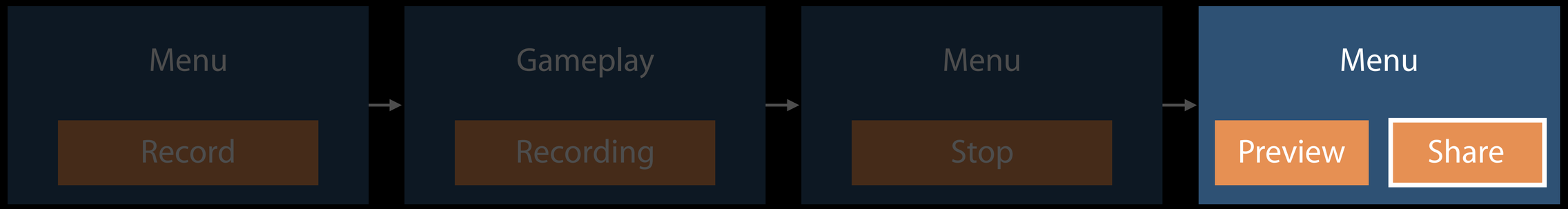


Share Recording




```
// RPPreviewViewController
public var mode: RPPreviewViewControllerMode
```

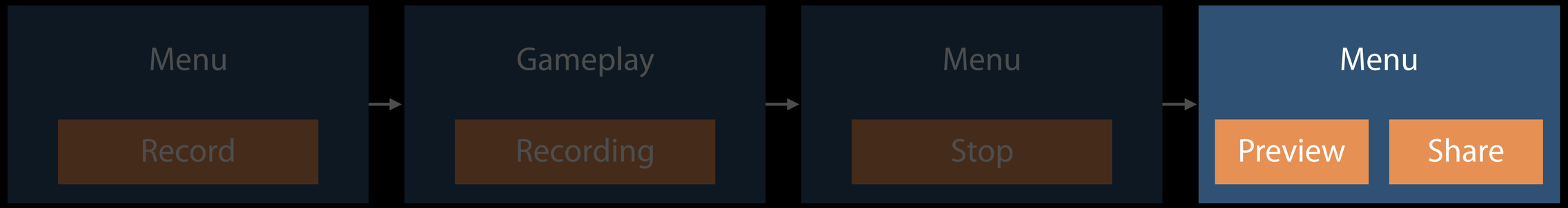
Share Recording



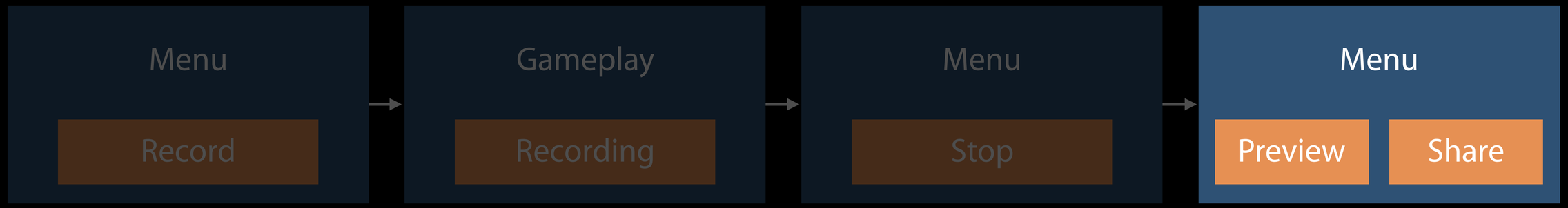

```
// RPPreviewViewController
public var mode: RPPreviewViewControllerMode
```

```
func didPressShareButton() {
    if let preview = previewViewController {
        preview.mode = .share
        self.present(preview, animated: true)
    }
}
```

Dismissing Preview UI



Dismissing Preview UI




```
// RPPreviewViewControllerDelegate
func previewControllerDidFinish(_ previewController: RPPreviewViewController) {
    previewController.dismiss(animated: true)
}
```

Discarding the Recording

Automatically discarded when new recording starts

- One recording allowed at a time, per app

Discard when preview no longer available

- Use discardRecording()

ReplayKit on Apple TV

Record your app video and audio content

- Microphone reserved by system

Preview and share the recording

Same simple API as iOS

New in tvOS 10

Live Broadcast

Edwin Iskandar Software Engineer

Live Broadcast

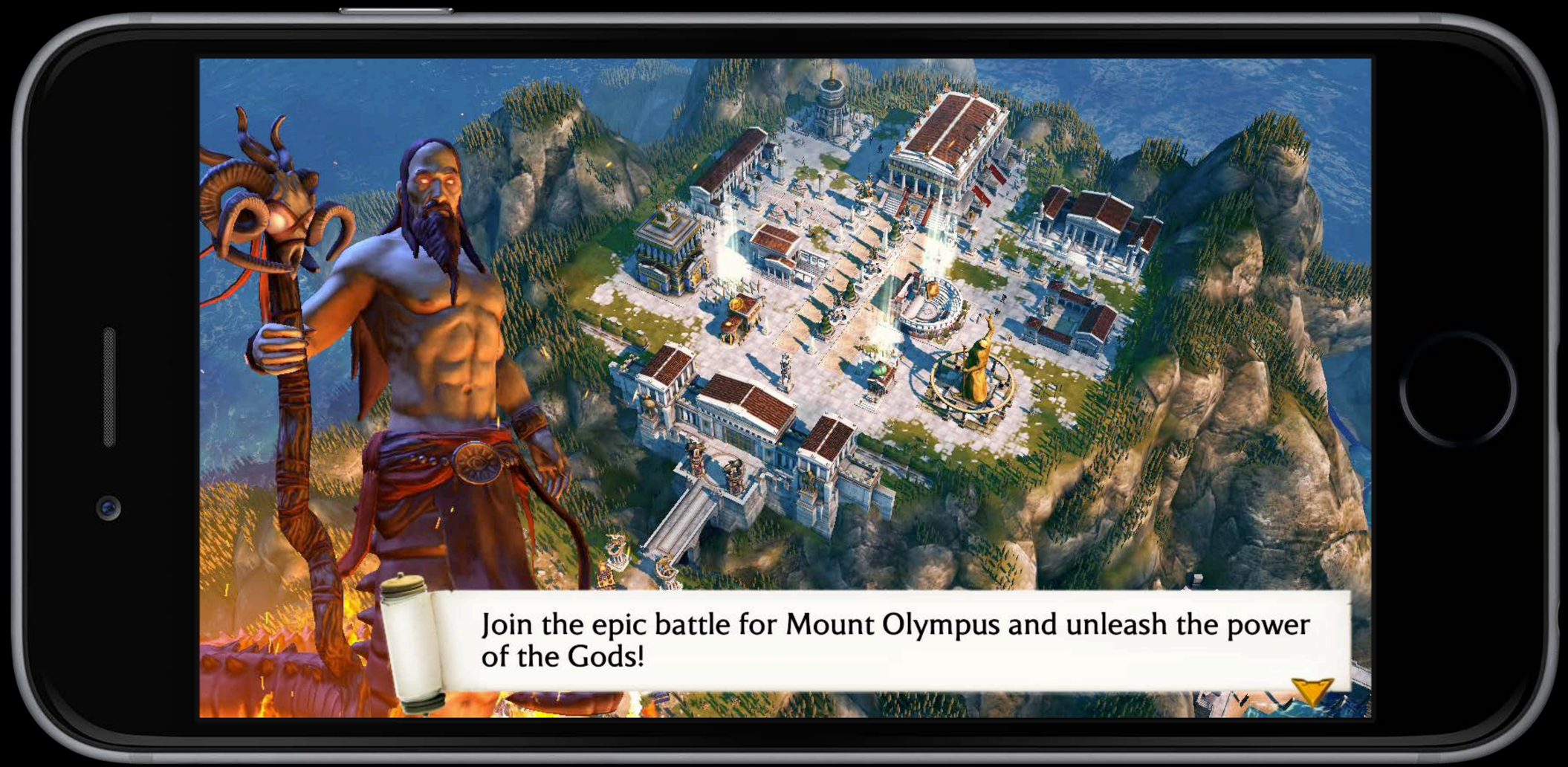
NEW

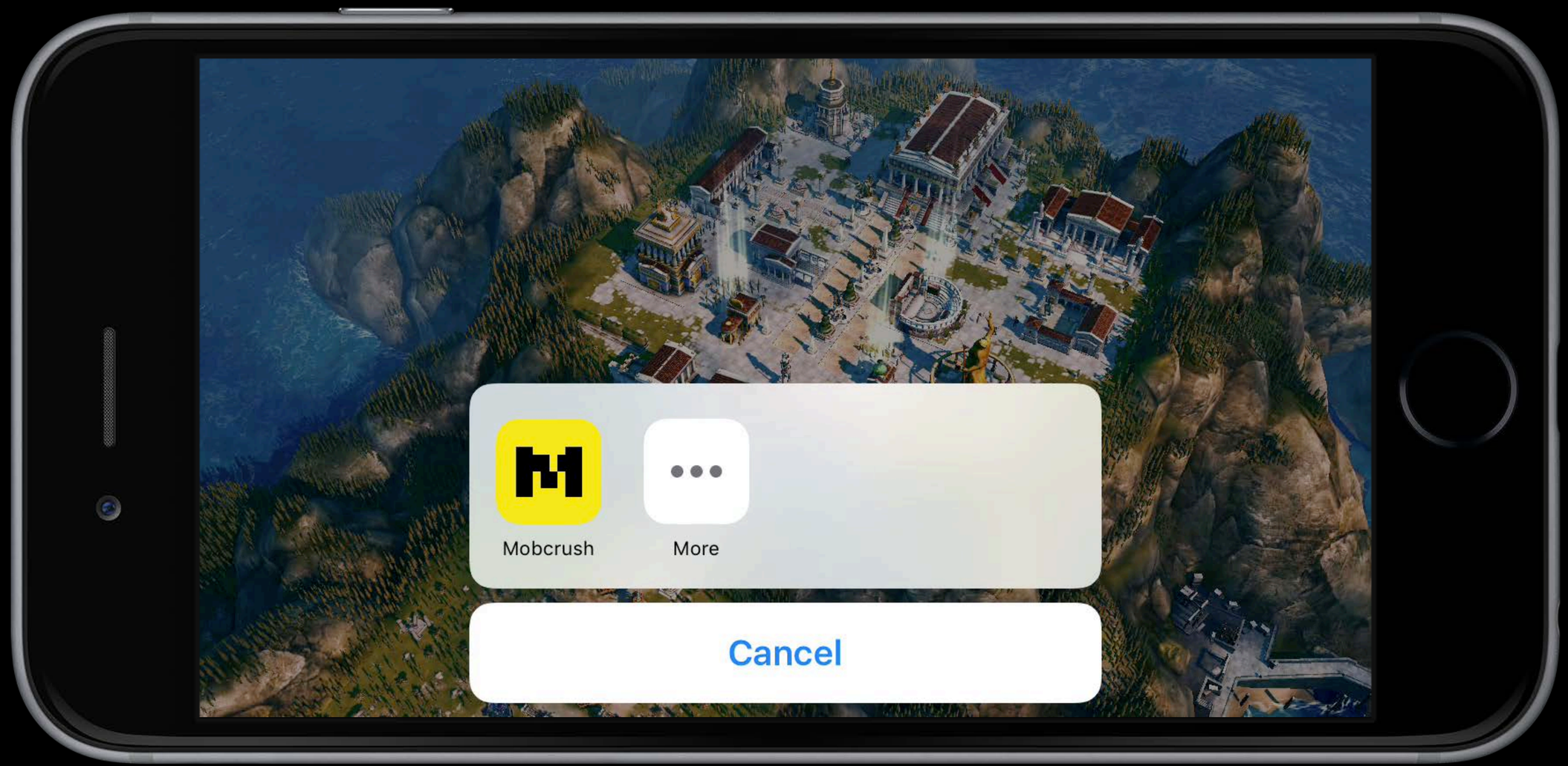
Broadcast live to 3rd party broadcast services

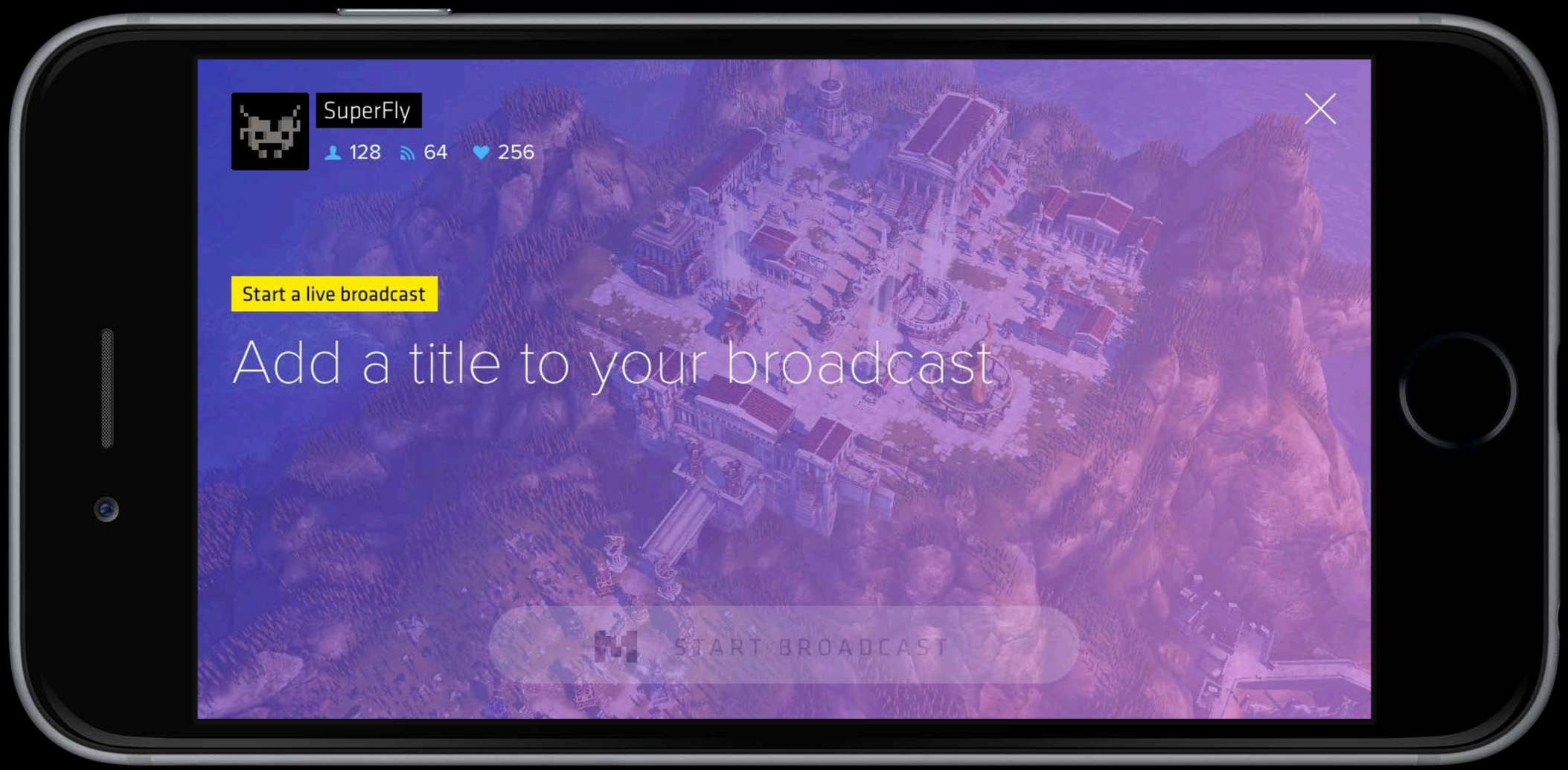
Directly from iOS / tvOS device

Provide commentary with mic and camera (iOS)

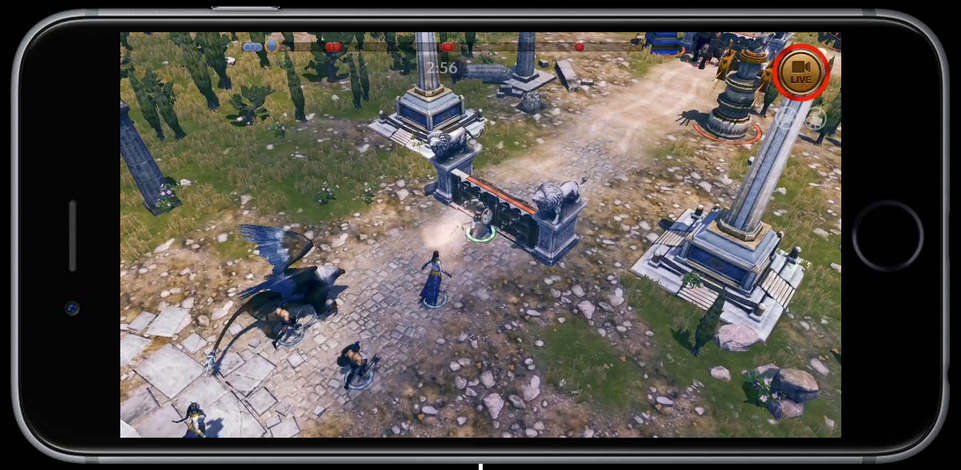
Content is secure and only accessible to the broadcast service

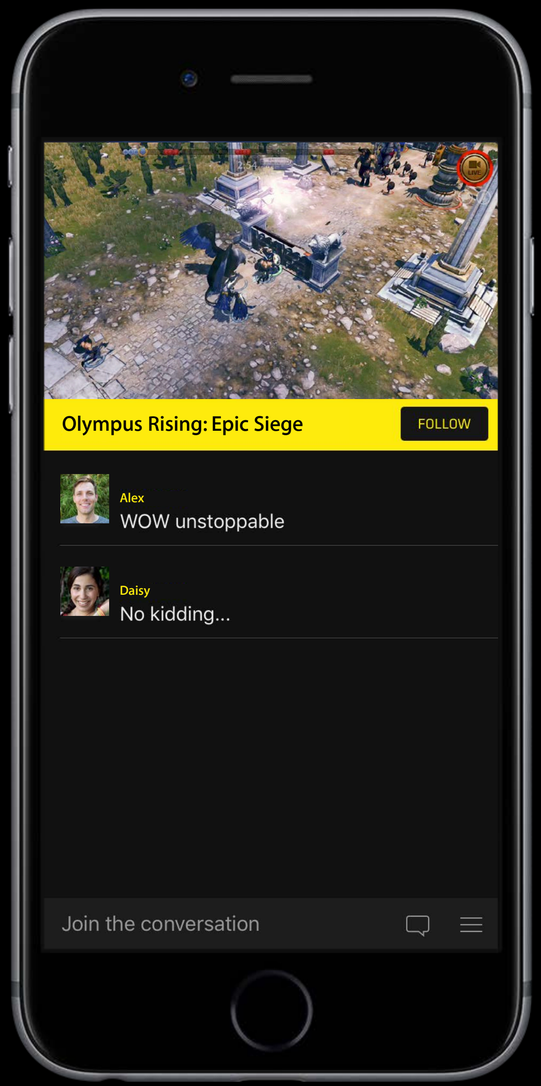

Join the epic battle for Mount Olympus and unleash the power of the Gods!

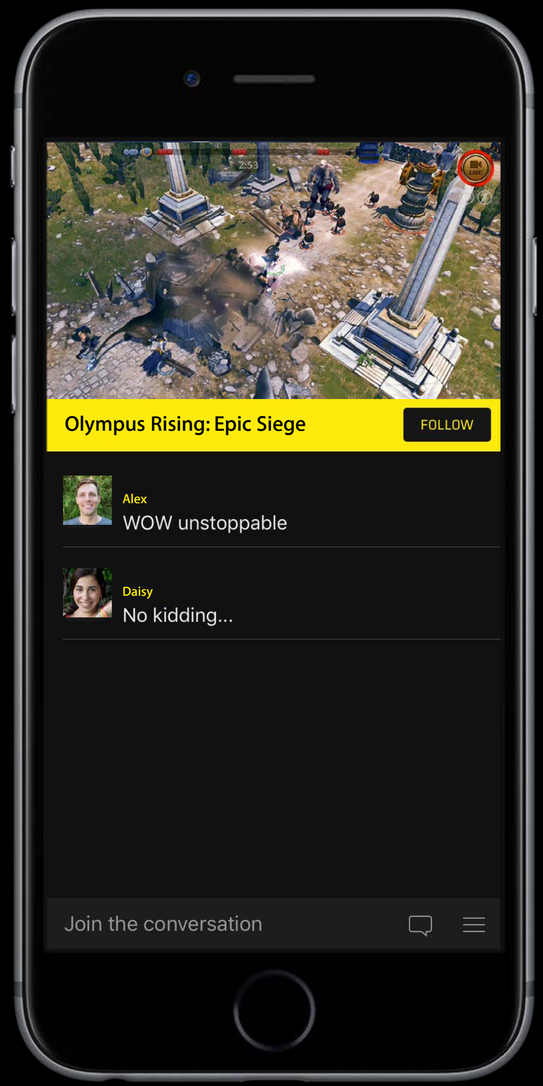

Mobcrush
More
Cancel


SuperFly
128
64
256
Start a live broadcast
Add a title to your broadcast
START BROADCAST

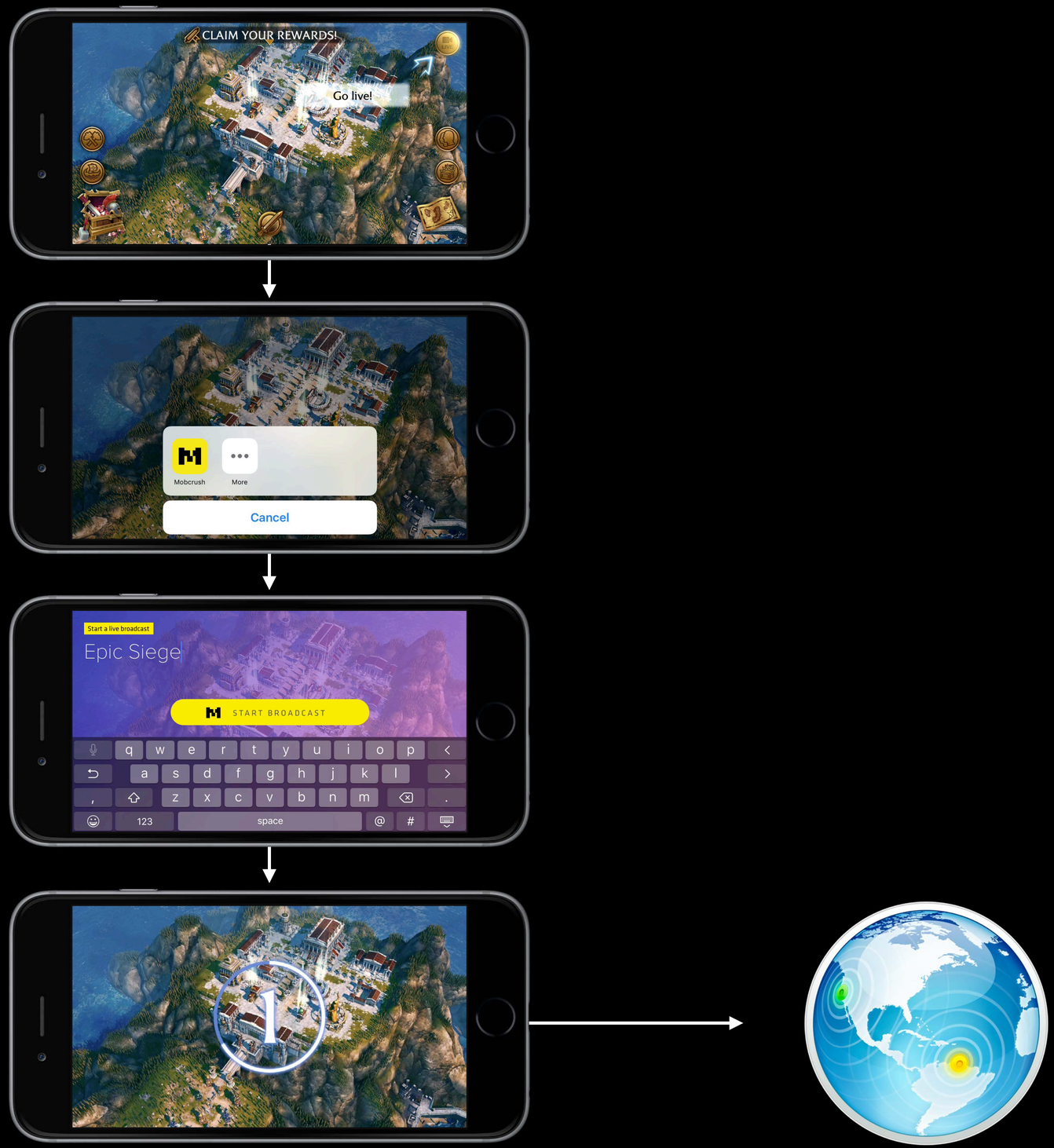

2:56
LIVE


2:56
LIVE

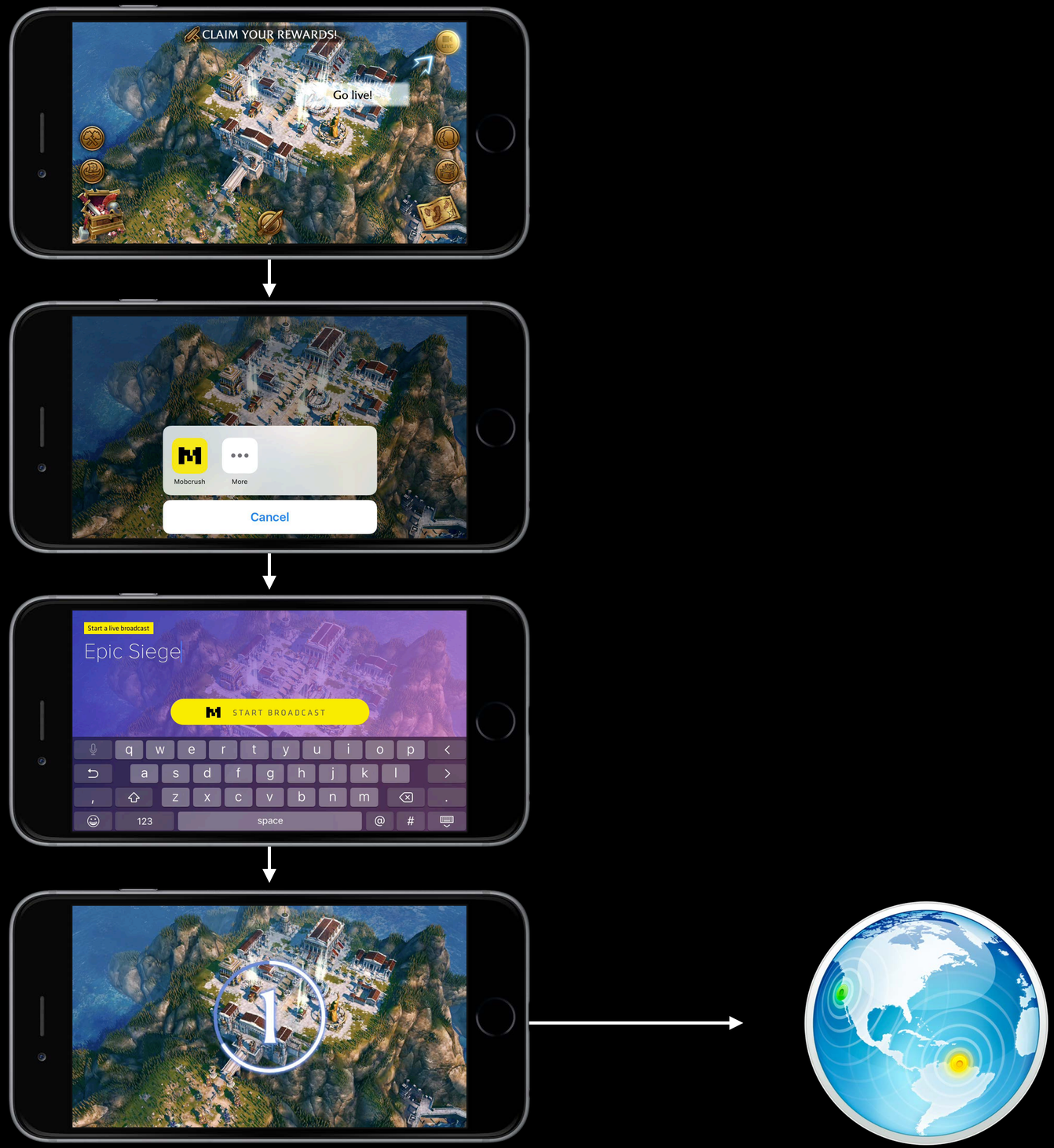

Olympus Rising: Epic Siege
FOLLOW
Alex
WOW unstoppable
Daisy
No kidding...
Join the conversation


Olympus Rising: Epic Siege
FOLLOW
Alex
WOW unstoppable
Daisy
No kidding...
Join the conversation

Game Implementation

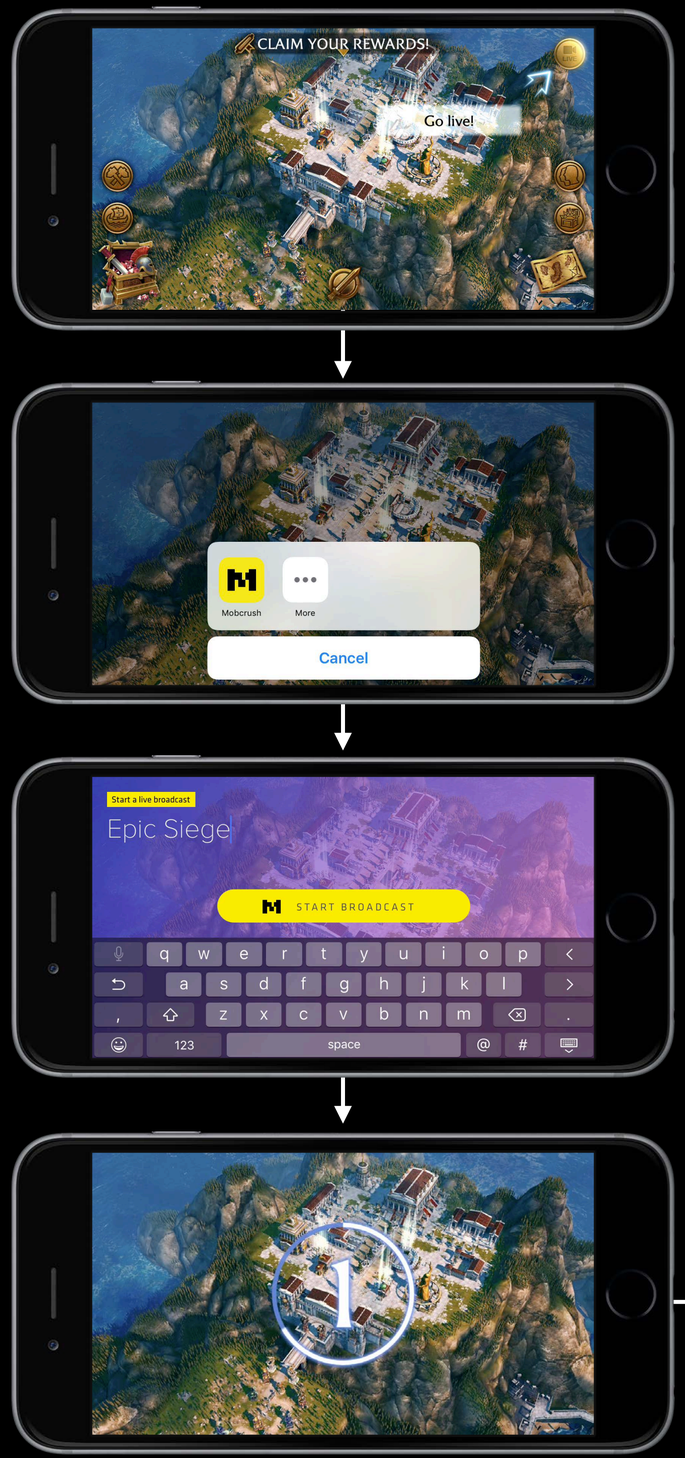

CLAIM YOUR REWARDS!
Go live!
Mobcrush
More
Cancel
Start a live broadcast
Epic Siege
START BROADCAST
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
123
space
@
#
1

Initiate Broadcast



Initiate Broadcast

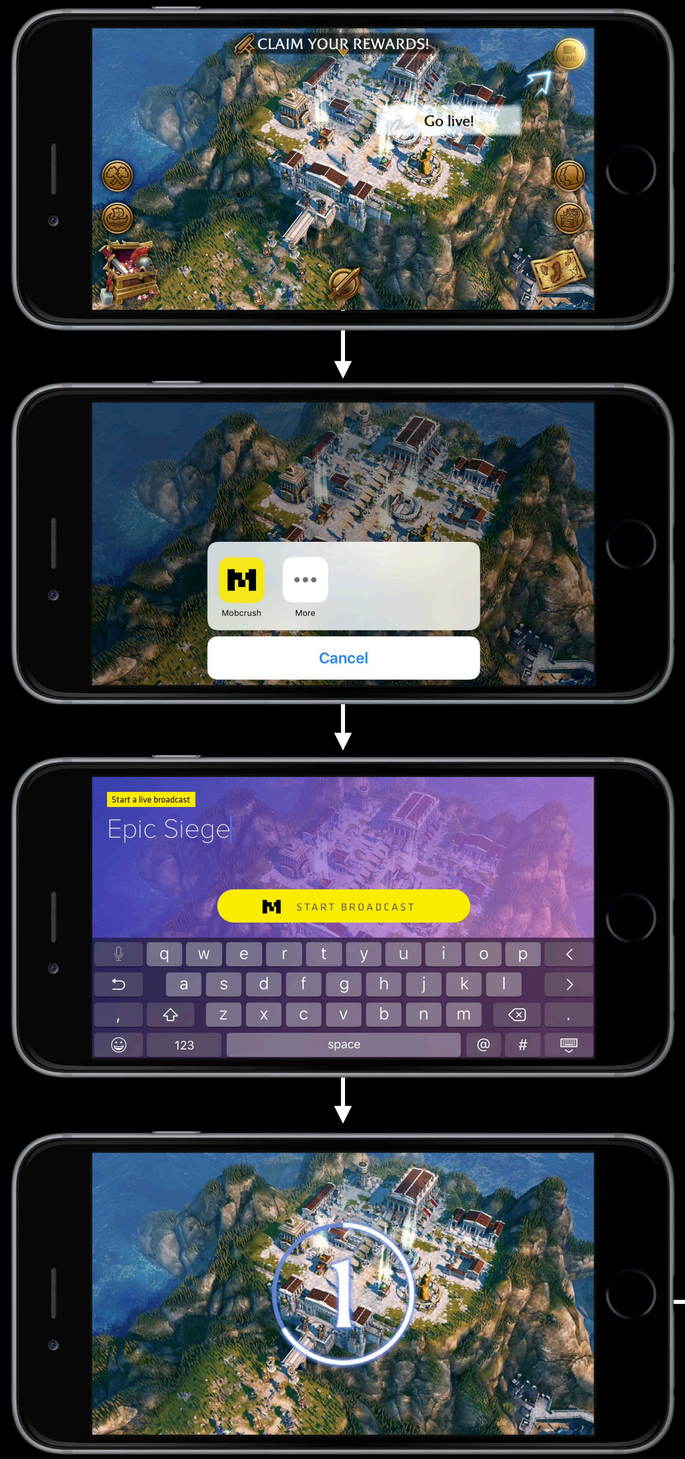
Select a Broadcast Service



Initiate Broadcast

Select a Broadcast Service

Set Up a Broadcast



Initiate Broadcast

Select a Broadcast Service

Set Up a Broadcast

Start and Stop a Broadcast

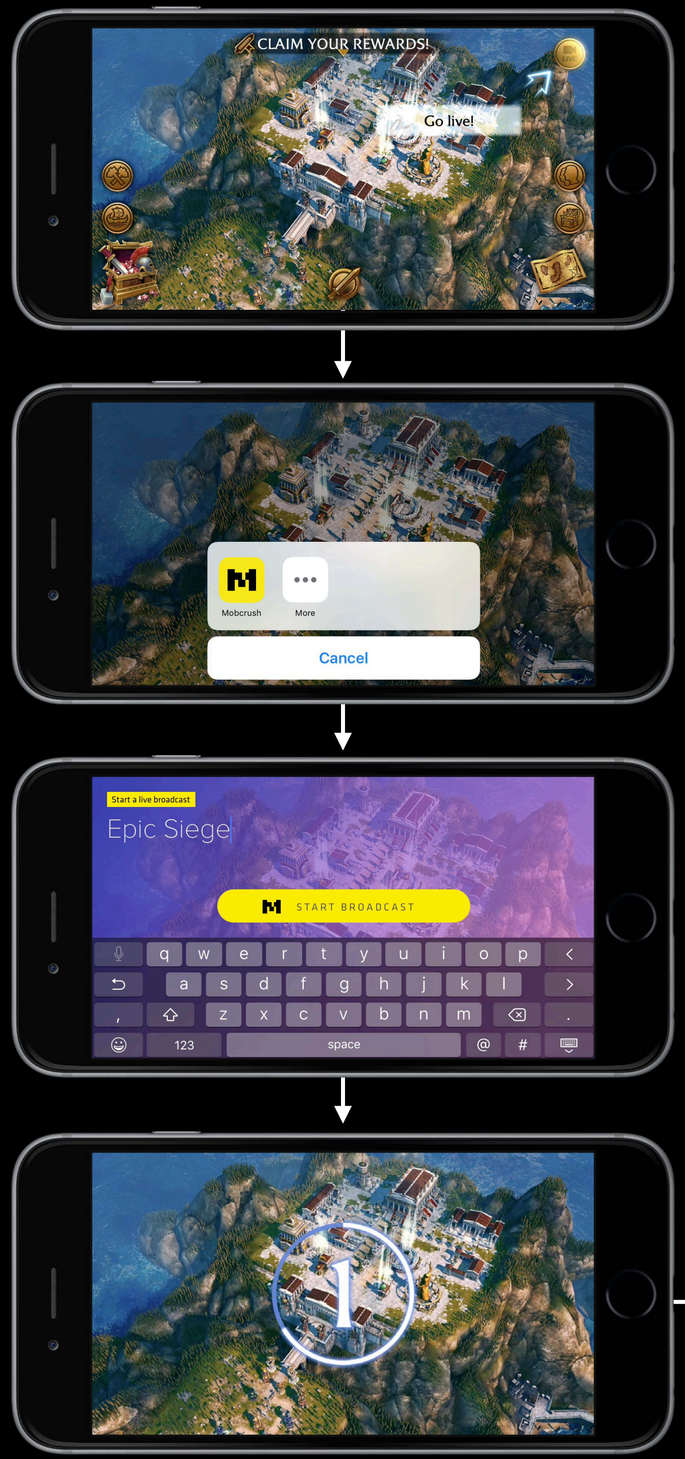
Initiate Broadcast

Select a Broadcast Service

Set Up a Broadcast

Start and Stop a Broadcast

Indicate Broadcast



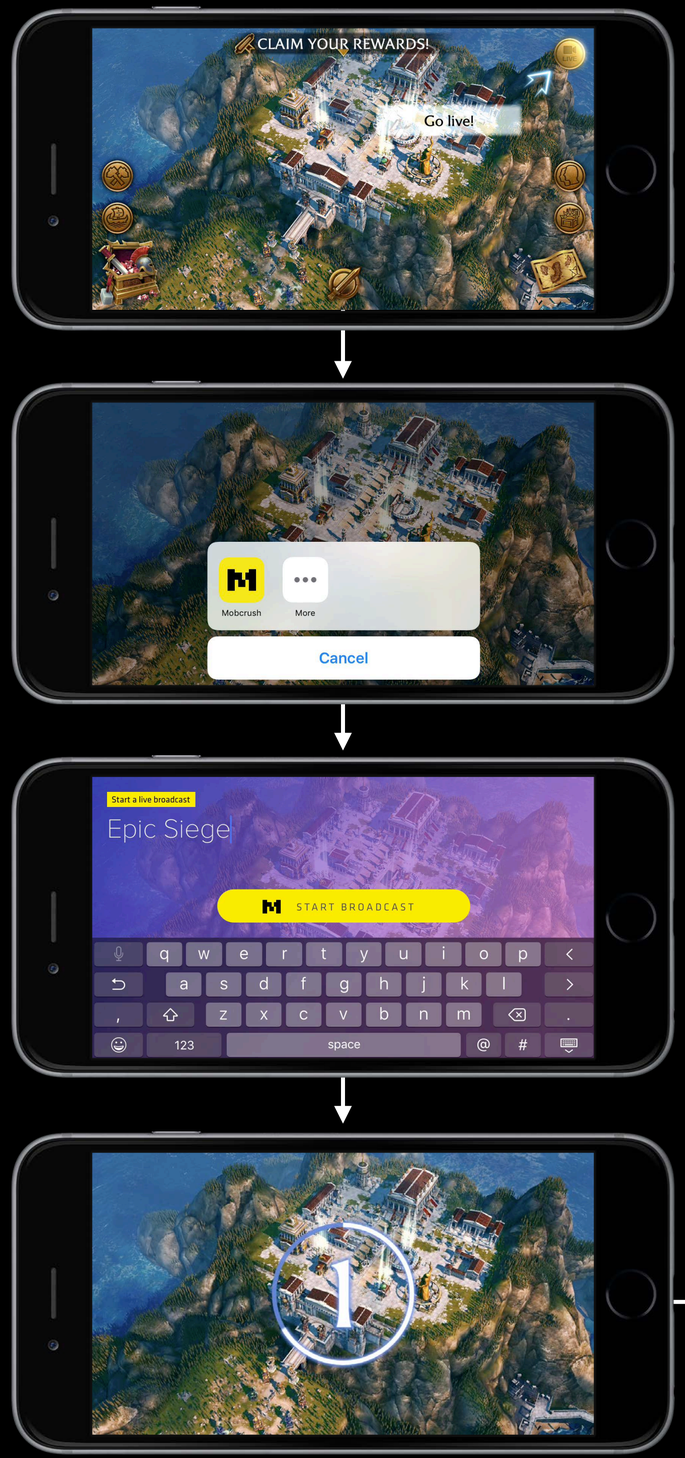
Initiate Broadcast

Select a Broadcast Service

Set Up a Broadcast

Start and Stop a Broadcast

Indicate Broadcast



Upload

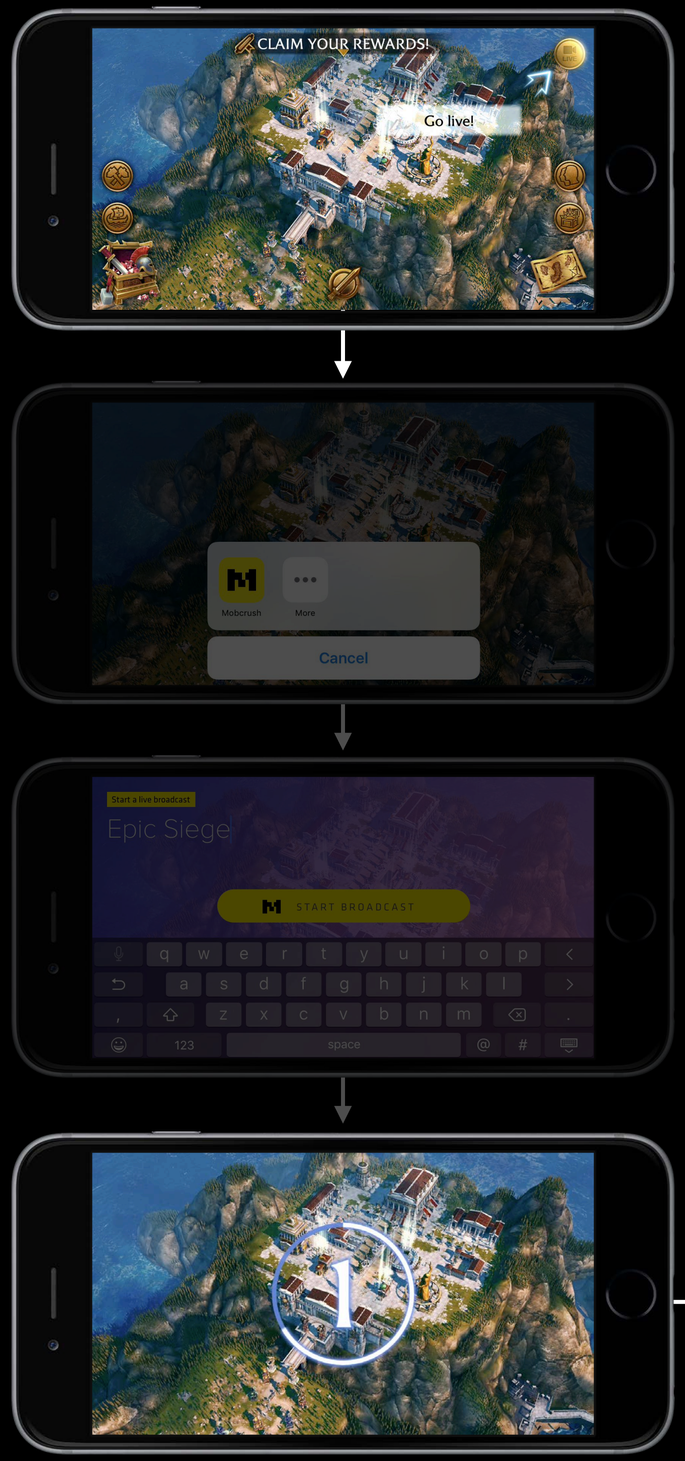
Initiate Broadcast

Select a Broadcast Service

Set Up a Broadcast

Start and Stop a Broadcast

Indicate Broadcast



Upload

Initiating a Broadcast


```
func didPressBroadcastButton() {

     RPBroadcastActivityViewController.load { broadcastAVC, error in

          if let broadcastAVC = broadcastAVC {
               broadcastAVC.delegate = self
               self.present(broadcastAVC, animated: true)
          }
     }
}
```

Initiating a Broadcast

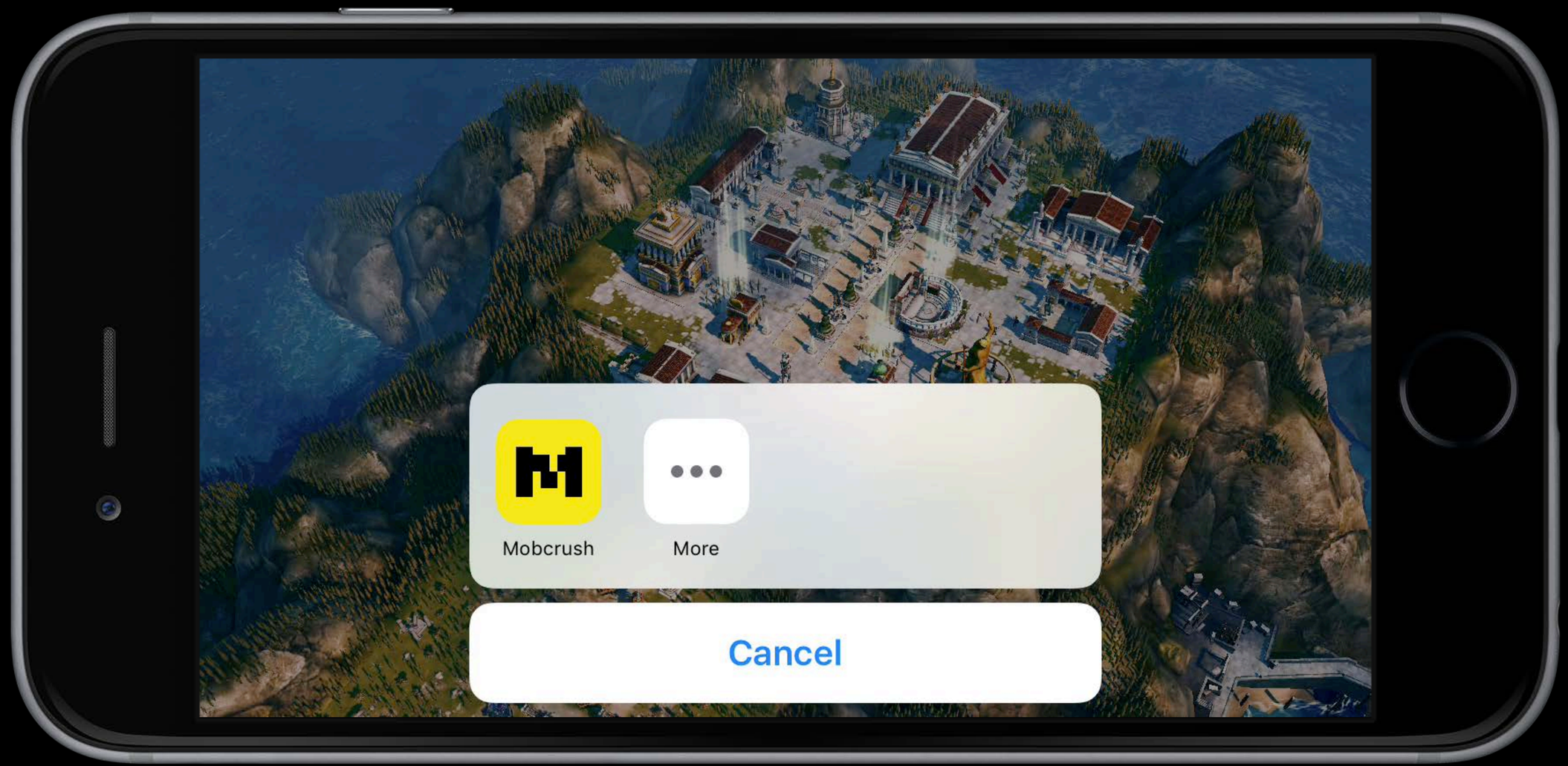

```
func didPressBroadcastButton() {

    RPBroadcastActivityViewController.load { broadcastAVC, error in

        if let broadcastAVC = broadcastAVC {
            broadcastAVC.delegate = self
            self.present(broadcastAVC, animated: true)
        }
    }
}
```

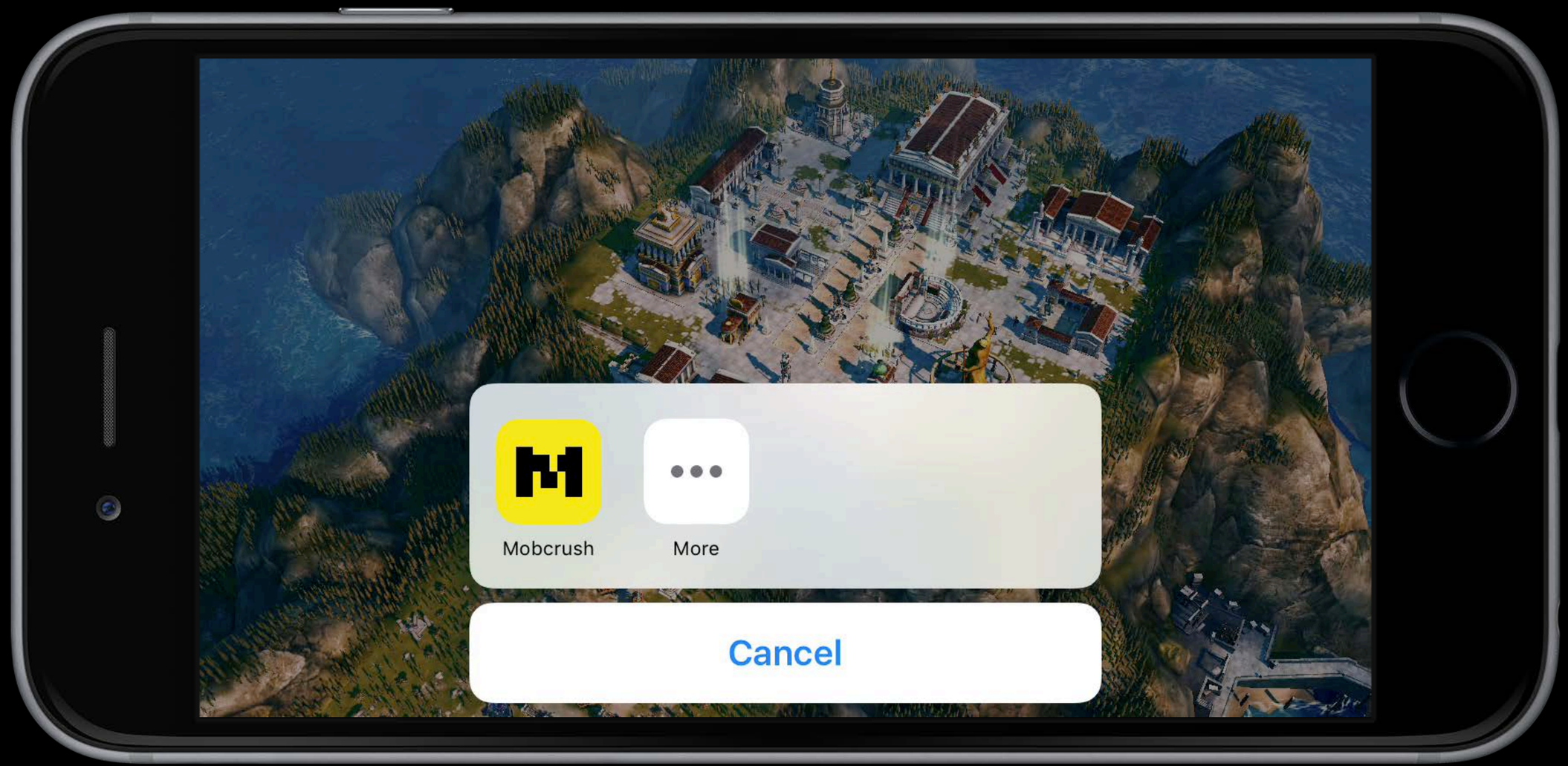
Initiating a Broadcast


```
func didPressBroadcastButton() {

     RPBroadcastActivityViewController.load { broadcastAVC, error in

          if let broadcastAVC = broadcastAVC {
               broadcastAVC.delegate = self
               self.present(broadcastAVC, animated: true)
          }
     }
}
```

Initiating a Broadcast


```
func didPressBroadcastButton() {

     RPBroadcastActivityViewController.load { broadcastAVC, error in

          if let broadcastAVC = broadcastAVC {
               broadcastAVC.delegate = self
               self.present(broadcastAVC, animated: true)
          }
     }
}
```


Mobcrush
More
Cancel


Mobcrush
More
Cancel

Starting a Broadcast


```
func broadcastActivityViewController(
                          _ broadcastAVC: RPBroadcastActivityViewController,
        didFinishWith broadcastController: RPBroadcastController?,
                                    error: NSError?) {

        broadcastAVC.dismiss(animated: true) {

            self.startCountDownTimer {
                broadcastController?.startBroadcast { error in
                    // broadcast started!
                }
            }
```

Starting a Broadcast


```
func broadcastActivityViewController(
                          _ broadcastAVC: RPBroadcastActivityViewController,
        didFinishWith broadcastController: RPBroadcastController?,
                                    error: NSError?) {

        broadcastAVC.dismiss(animated: true) {

            self.startCountDownTimer {
                broadcastController?.startBroadcast { error in
                    // broadcast started!
                }
            }
```

Starting a Broadcast


```
func broadcastActivityViewController(
                            _ broadcastAVC: RPBroadcastActivityViewController,
        didFinishWith broadcastController: RPBroadcastController?,
                                    error: NSError?) {

        broadcastAVC.dismiss(animated: true) {

            self.startCountDownTimer {
                broadcastController?.startBroadcast { error in
                    // broadcast started!
                }
            }
```

Starting a Broadcast


```
func broadcastActivityViewController(
                          _ broadcastAVC: RPBroadcastActivityViewController,
        didFinishWith broadcastController: RPBroadcastController?,
                                    error: NSError?) {

        broadcastAVC.dismiss(animated: true) {

            self.startCountDownTimer {
                broadcastController?.startBroadcast { error in
                    // broadcast started!
                }
            }
```

Starting a Broadcast


```
func broadcastActivityViewController(
                            _ broadcastAVC: RPBroadcastActivityViewController,
        didFinishWith broadcastController: RPBroadcastController?,
                                    error: NSError?) {

        broadcastAVC.dismiss(animated: true) {

            self.startCountDownTimer {
                broadcastController?.startBroadcast { error in
                    // broadcast started!
                }
            }
```

Indicating a Broadcast

Animate to indicate activity

Merge with controls if space constrained

Required during broadcast

```
broadcastController.isBroadcasting
```

Indicating a Broadcast

Animate to indicate activity

Merge with controls if space constrained

Required during broadcast

```
broadcastController.isBroadcasting
```

Indicating a Broadcast

Animate to indicate activity

Merge with controls if space constrained

Required during broadcast

```
broadcastController.isBroadcasting
```

Indicating a Broadcast

Animate to indicate activity

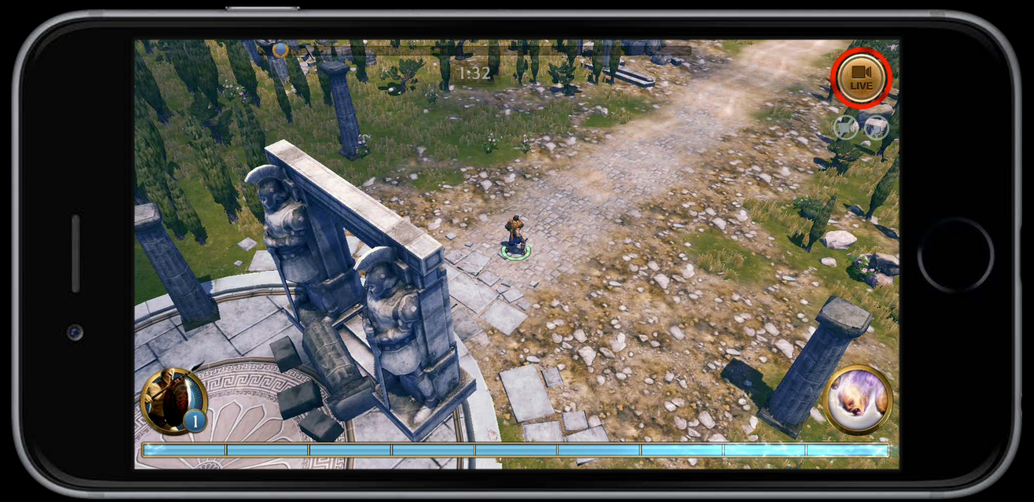
Merge with controls if space constrained

Required during broadcast


```
broadcastController.isBroadcasting
```

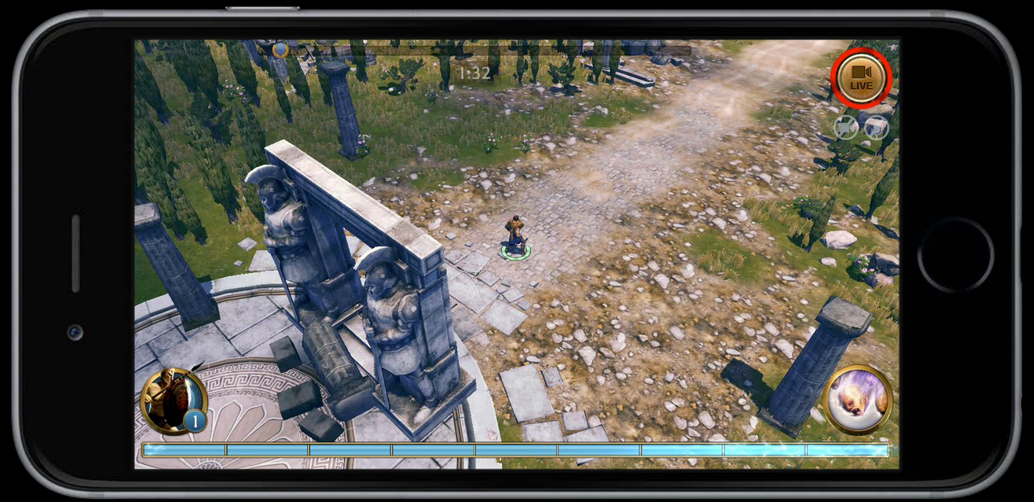
```
func updateBroadcastButton() {

     if self.broadcastController?.isBroadcasting == true {
         self.startAnimateIndicator()
     } else {
         self.stopAnimatingIndicator()
     }
 }
```

Finish Broadcast



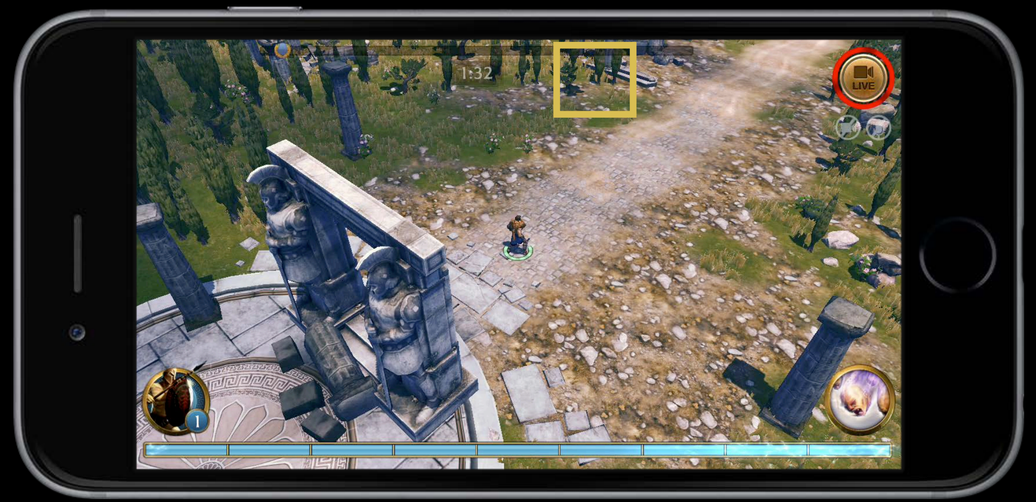

```
func didPressBroadcastButton() {

    self.broadcastController?.finishBroadcast { error in

          if error == nil {
               // broadcast finished!
               self.updateBroadcastUI()
          }
    }
}
```

Finish Broadcast



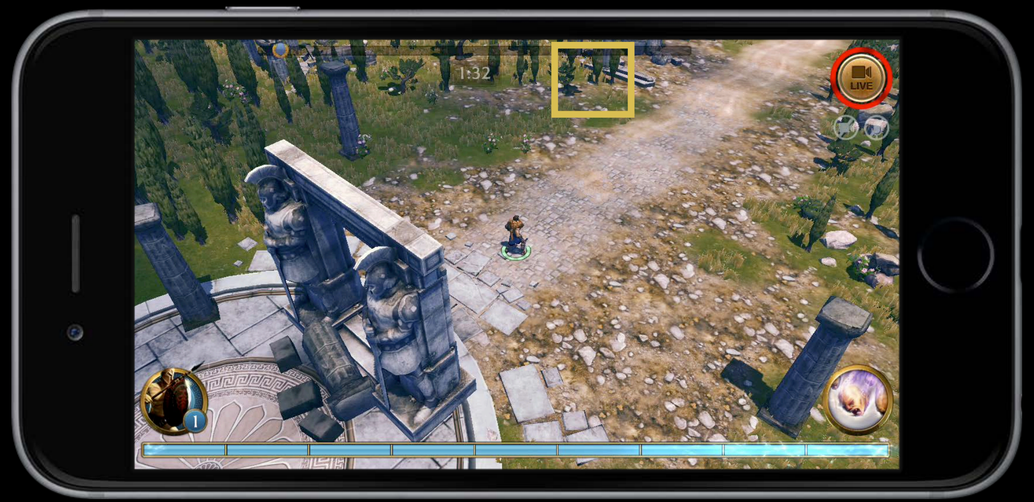

```
func didPressBroadcastButton() {

    self.broadcastController?.finishBroadcast { error in

          if error == nil {
               // broadcast finished!
               self.updateBroadcastUI()
          }
    }
}
```

Finish Broadcast




```
func didPressBroadcastButton() {

    self.broadcastController?.finishBroadcast { error in

          if error == nil {
               // broadcast finished!
               self.updateBroadcastUI()
          }
    }
}
```

Finish Broadcast




```
func didPressBroadcastButton() {
    self.broadcastController?.finishBroadcast { error in
        if error == nil {
            // broadcast finished!
            self.updateBroadcastUI()
        }
    }
}
```

```
// Error Handling

func broadcastActivityViewController(
      _ broadcastActivityViewController: RPBroadcastActivityViewController,
      didFinishWith broadcastController: RPBroadcastController?,
                                  error: NSError?) {

    self.broadcastController = broadcastController

    // set a delegate to be notified of errors
    self.broadcastController?.delegate = self
}
```

```
// Error Handling

func broadcastActivityViewController(
      _ broadcastActivityViewController: RPBroadcastActivityViewController,
      didFinishWith broadcastController: RPBroadcastController?,
                                  error: NSError?) {

    self.broadcastController = broadcastController

    // set a delegate to be notified of errors
    self.broadcastController?.delegate = self
}
```

```
// Error Handling

func broadcastController(
                _ broadcastController: RPBroadcastController,
            didFinishWithError error: NSError?) {

    if error != nil {

        // error occurred during broadcast
        self.showErrorMessage(message: error!.localizedDescription)

        // update UI to indicate the broadcast is stopped
        self.updateBroadcastUI()
    }
}
```

```
// Error Handling

func broadcastController(
                _ broadcastController: RPBroadcastController,
            didFinishWithError error: NSError?) {

    if error != nil {

        // error occurred during broadcast
        self.showErrorMessage(message: error!.localizedDescription)

        // update UI to indicate the broadcast is stopped
        self.updateBroadcastUI()
    }
}
```

```swift
// Application Backgrounding

    func applicationWillResignActive() {
        // ReplayKit will automatically pause the broadcast
    }

    func applicationDidBecomeActive() {
        if self.broadcastController?.isBroadcasting == true {
            self.promptUserToResumeBroadcast { userWantsToResume in
                if (userWantsToResume == true) {
                    // user wants to resume
                    self.broadcastController?.resumeBroadcast()
                    self.updateBroadcastUI()
                } else {
                    // user does not want to resume
                    self.broadcastController?.finishBroadcast { error in
                        self.updateBroadcastUI()
                    }
```

```
// Application Backgrounding

    func applicationWillResignActive() {
        // ReplayKit will automatically pause the broadcast
    }

    func applicationDidBecomeActive() {
        if self.broadcastController?.isBroadcasting == true {
            self.promptUserToResumeBroadcast { userWantsToResume in
                if (userWantsToResume == true) {
                    // user wants to resume
                    self.broadcastController?.resumeBroadcast()
                    self.updateBroadcastUI()
                } else {
                    // user does not want to resume
                    self.broadcastController?.finishBroadcast { error in
                        self.updateBroadcastUI()
                    }
```

```swift
// Application Backgrounding

    func applicationWillResignActive() {
        // ReplayKit will automatically pause the broadcast
    }

    func applicationDidBecomeActive() {
        if self.broadcastController?.isBroadcasting == true {
            self.promptUserToResumeBroadcast { userWantsToResume in
                if (userWantsToResume == true) {
                    // user wants to resume
                    self.broadcastController?.resumeBroadcast()
                    self.updateBroadcastUI()
                } else {
                    // user does not want to resume
                    self.broadcastController?.finishBroadcast { error in
                        self.updateBroadcastUI()
                    }
```

```swift
// Application Backgrounding

    func applicationWillResignActive() {
        // ReplayKit will automatically pause the broadcast
    }

    func applicationDidBecomeActive() {
        if self.broadcastController?.isBroadcasting == true {
            self.promptUserToResumeBroadcast { userWantsToResume in
                if (userWantsToResume == true) {
                    // user wants to resume
                    self.broadcastController?.resumeBroadcast()
                    self.updateBroadcastUI()
                } else {
                    // user does not want to resume
                    self.broadcastController?.finishBroadcast { error in
                        self.updateBroadcastUI()
                    }
```

Classes and Protocols

Game API

Classes and Protocols

Game API

`RPBroadcastActivityViewController`

- Present installed broadcast services

`RPBroadcastActivityViewControllerDelegate`

- Notified when broadcast setup is complete

Classes and Protocols

Game API

`RPBroadcastActivityViewController`

- Present installed broadcast services

`RPBroadcastActivityViewControllerDelegate`

- Notified when broadcast setup is complete

`RPBroadcastController`

- Start and finish broadcast
- Check if broadcast is in-progress

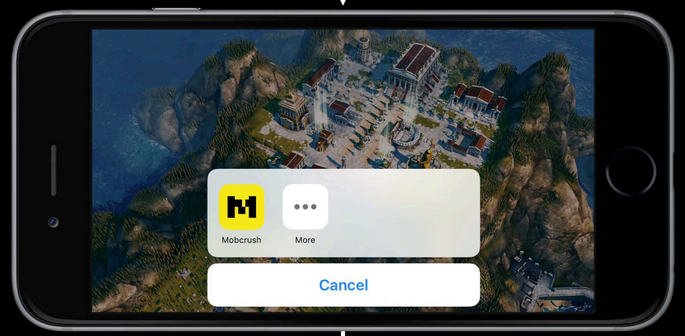
`RPBroadcastControllerDelegate`

- Handle errors during broadcast

Broadcast Services

Initiate Broadcast

Select a Broadcast Service



Set Up a Broadcast

Start and Stop a Broadcast

Indicate Broadcast

Upload

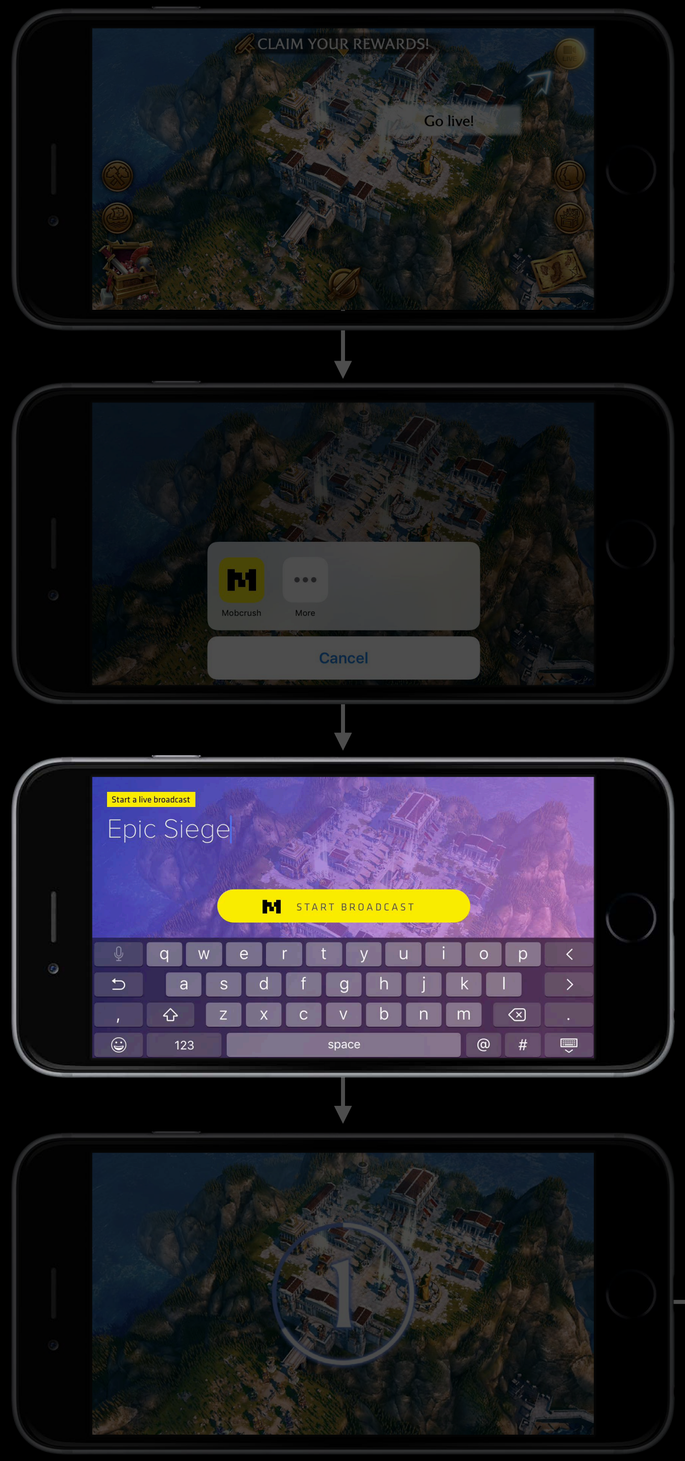
Initiate Broadcast

Select a Broadcast Service

Set Up a Broadcast

Start and Stop a Broadcast

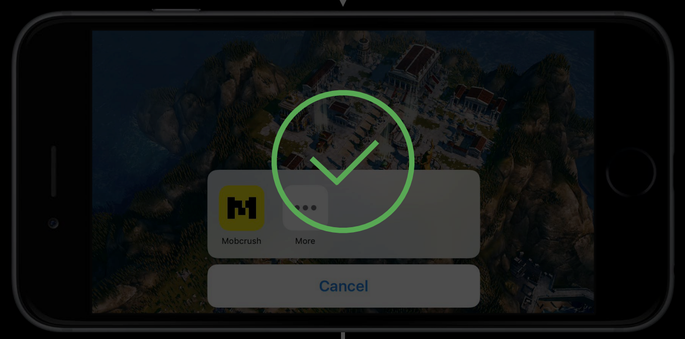
Indicate Broadcast



Upload

Initiate Broadcast

Select a Broadcast Service



Set Up a Broadcast

Start and Stop a Broadcast

Indicate Broadcast

Upload

Broadcast Services

Broadcast UI Extension

- Set up broadcast

Broadcast Upload Extension

- Process and upload video and audio data

Set Up a Broadcast

Upload

Broadcast Extensions

Embedded in your application

Execute alongside other application processes

Can share data between parent application

Limited in resources compared to applications

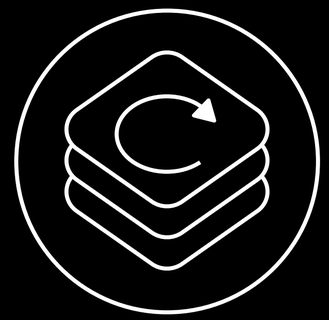
Xcode Templates

New Target templates available in Xcode

Add Target -> iOS/tvOS -> Application Extension

Pre-configured with NSExtension properties in info.plist

Broadcast UI Extension



Broadcast Upload

Broadcast UI Extension

Authenticate the user and provide sign-up

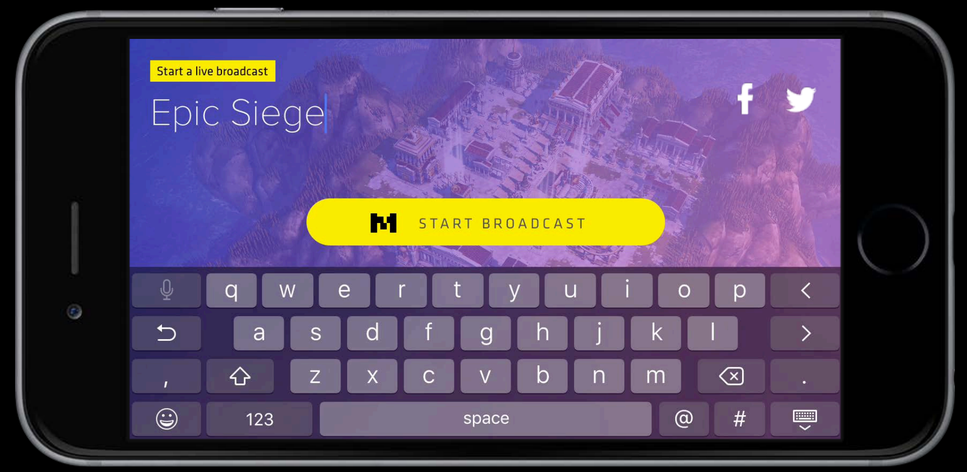
Accept terms and conditions

Set up the broadcast

Optionally share via social media

Notify setup is complete

Set Up a Broadcast



Broadcast Upload Extension

Receive and process video and audio data

Upload to server

Implementation to be defined by broadcast services

Work together with us

Initiate Broadcast

Select a Broadcast Service

Set Up a Broadcast

Start and Stop a Broadcast

Indicate Broadcast

Upload

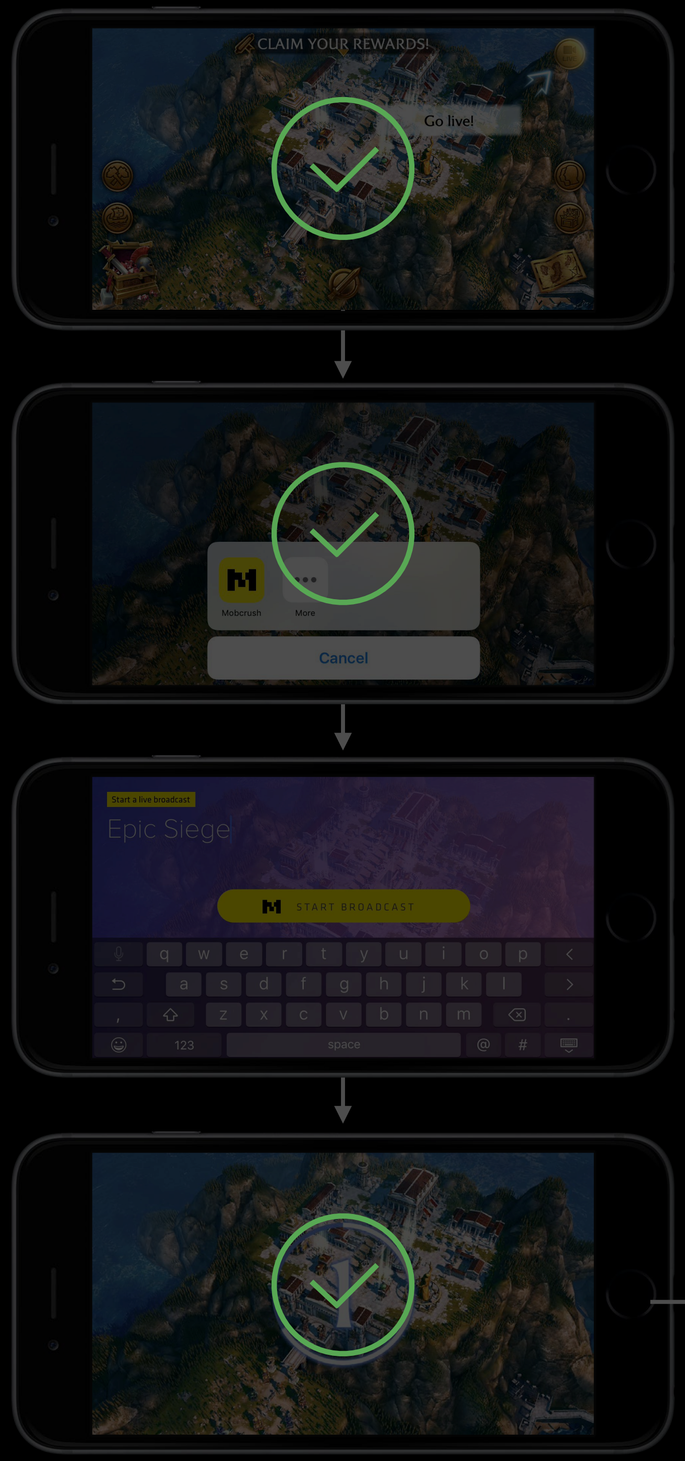
Initiate Broadcast

Select a Broadcast Service

Set Up a Broadcast

Start and Stop a Broadcast

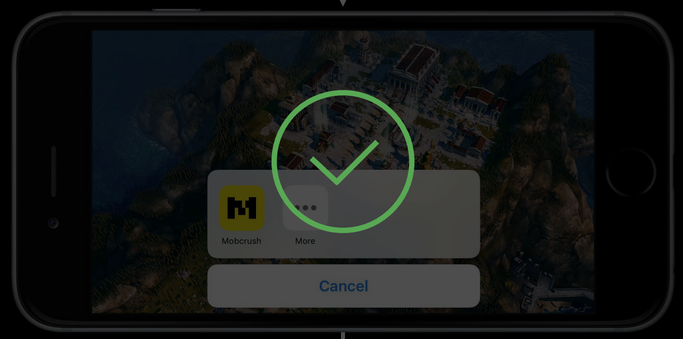
Indicate Broadcast



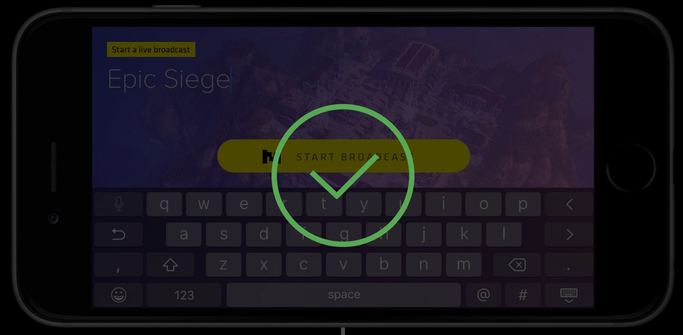
Upload

Initiate Broadcast

Select a Broadcast Service



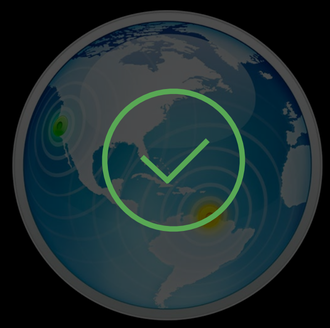
Set Up a Broadcast



Start and Stop a Broadcast

Indicate Broadcast

Upload



Responsibilities

Initiate Broadcast

Start and Stop a Broadcast

Indicate Broadcast

Select a Broadcast Service

Set Up a Broadcast

Upload

Responsibilities

Game	ReplayKit	Broadcast Service
Initiate Broadcast	Select a Broadcast Service	Set Up a Broadcast
Start and Stop a Broadcast		Upload
Indicate Broadcast		

Live Broadcasting

Expanded Commentary Options

FaceTime camera support

Flexible microphone recording

Available in iOS 10

FaceTime Camera Support

FaceTime Camera Support

FaceTime Camera Support

```
RPScreenRecorder.shared().isCameraEnabled
```

Camera preview view available in RPScreenRecorder

Subclass of UIView

Position to not obstruct gameplay

Optionally allow the user to move it

FaceTime Camera Support

```
RPScreenRecorder.shared().isCameraEnabled
```

Camera preview view available in RPScreenRecorder

Subclass of UIView

Position to not obstruct gameplay

Optionally allow the user to move it

```
RPScreenRecorder.shared().isCameraEnabled = true
```

FaceTime Camera Support

```
RPScreenRecorder.shared().isCameraEnabled
```

Camera preview view available in RPScreenRecorder

Subclass of UIView

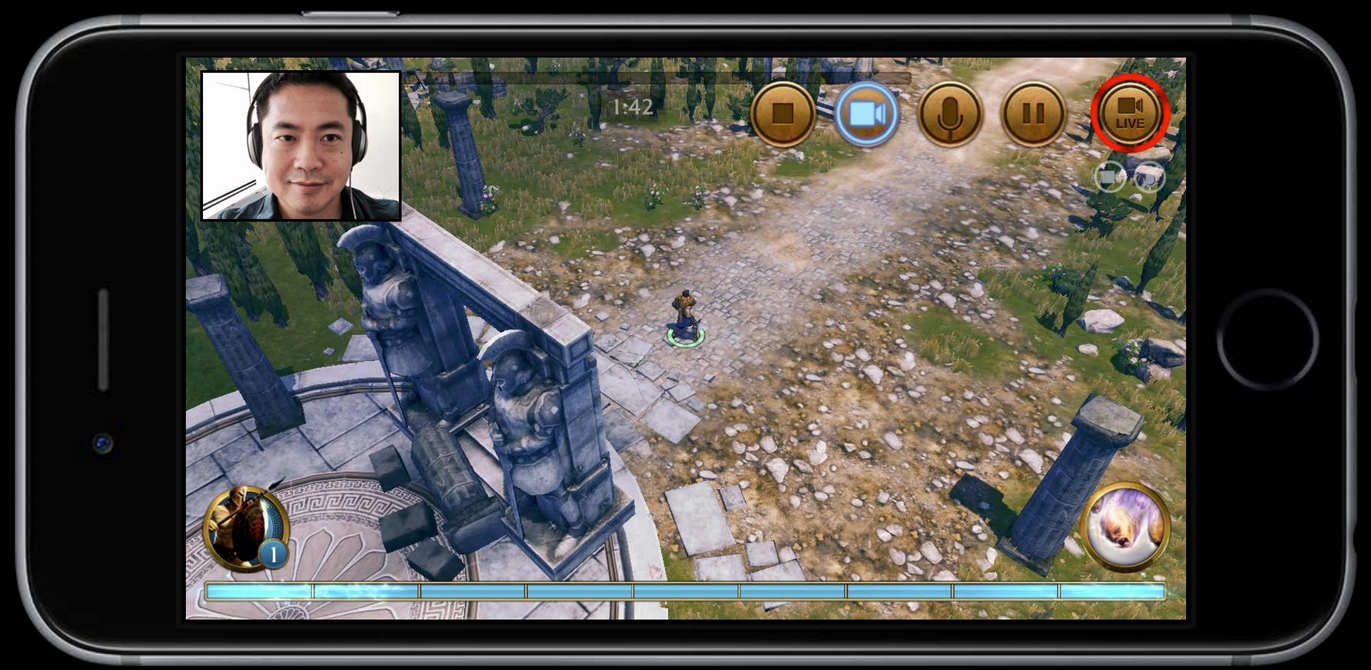
Position to not obstruct gameplay

Optionally allow the user to move it

```
RPScreenRecorder.shared().isCameraEnabled = true
```

```
if let cameraPreview = RPScreenRecorder.shared().cameraPreviewView {
    cameraPreview.frame = CGRect(…)
    self.view.addSubview(cameraPreview)
}
```

Microphone Support



```
// Microphone Recording


    func enableMic {
        RPScreenRecorder.shared().isMicrophoneEnabled = true
    }

    func disableMic {
        RPScreenRecorder.shared().isMicrophoneEnabled = false
    }
```

Summary

Apple TV support

Live broadcasting

Expanded commentary options

More Information

https://developer.apple.com/wwdc16/601

Related Sessions

What's New in GameplayKit	Pacific Heights	Thursday 9:00AM
What's New in SpriteKit	Presidio	Thursday 5:00PM
What's New in Game Center	Mission	Friday 10:00AM

Labs

ReplayKit Lab	Graphics Lab A	Tuesday 12:00PM
ReplayKit Lab	Graphics Lab B	Wednesday 9:00AM

WWDC16