TODAY

- Music
- Accelerometer
- Undo
SOUNDS

- Lots of sounds, might be:
  - beeps / bleeps
  - background music
- Play using AVAudioPlayer
  - each instance can play one thing
  - stops as soon as deallocated (goes out of scope)
- mediated via singleton instance of AVAudioSession

AVAUDIOSESSION
try? AVAudioSession.sharedInstance().setCategory(AVAudioSession.Category.playback)
try? AVAudioSession.sharedInstance().setActive(true)

var beepPlayer = AVAudioPlayer()
var backgroundMusicPlayer = AVAudioPlayer()

func playBeep(number: Int) {
    let soundFile = (number > 2048) ? "beep4096" : "beep\(number)"
    guard let url = Bundle.main.url(forResource: soundFile, withExtension: "mp3") else { return }
    try? beepPlayer = AVAudioPlayer(contentsOf: url)
    beepPlayer.play()
}

func playBackground(forResource: String, withExtension: String = "mp3") {
    guard let url = Bundle.main.url(forResource: forResource, withExtension: withExtension) else { return }
    try? backgroundMusicPlayer = AVAudioPlayer(contentsOf: url)
    backgroundMusicPlayer.setVolume(0.1, fadeDuration: 0)
    backgroundMusicPlayer.play()
}
MOTION

- iPhone devices:
  - proximity sensor (how close to face)
  - ambient light
  - moisture
  - compass
  - barometer
  - touch ID
  - face ID
  - **accelerometer**
    - linear acceleration (w/ or w/o gravity)
  - **gyroscope**
    - rotation rate around spatial axis

MOTION AXIS
CMDeviceMotion objects

- CMDeviceMotion:
  - The device's orientation (or attitude) in three-dimensional space relative to a reference frame
  - The current magnetic field vector
  - The rotation rate (gyroscope)
  - The user-generated acceleration vector (without gravity)
  - The current gravity vector (accelerometer)

Core Motion

- Set update interval
- Either poll or provide closure

```swift
self.motionManager.accelerometerUpdateInterval = self.MOTION_UPDATE

self.motionManager.startAccelerometerUpdates(to: OperationQueue.main,
                                          withHandler: { (data, error) in
                                          ...
                                          })
```

- Caveats:
  - Updates need to be explicitly turned
  - Filter to get events rather than just a stream
THE UNDO MANAGER

- Very simple:
  - each app has access to an `undoManager`
  - `.registerUndo(withTarget: self) { .... }`
  - `.undo()`
  - `.redo()`

- undo and redo stacks
  - `.registerUndo()` adds to undo stack
  - when calling `.undo()`, will need to register an undo of the undo via `.registerUndo()` (again)
  - gets put on redo stack because during an undo
Code

- Code to transition the state
  - calling itself recursively calls another registerUndo, an undo of an undo, which is put on the redo stack.

```swift
func changeState(state: [Int]) {
    let old = self.game.getState()
    undoManager?.registerUndo(withTarget: self, handler: {
        (targetSelf) in
        self.changeState(state: old)
    })
    self.game.putState(instate: state)
    self.redoOutlet.isEnabled = undoManager!.canRedo
    self.undoOutlet.isEnabled = undoManager!.canUndo
    updateViewFromModel()
}
```

Demo