2017-18 Software Engineering Capstone Project

By: Jackson Lawler and Nick Setliff

Survey Results

- 70% of survey respondents reported iOS as their mobile operating system
- 88% thought an AR navigational app would help students find their way around campus
- 55% were likely to very likely to use an AR navigational app
- 95% thought an AR navigational app that provided locations and navigation to current events would be useful
- 80% were likely to very likely to use an AR app that provided locations and navigation to current events
- 100% thought an AR app with fitness features would be beneficial to students
- 92% were likely to use an AR app with fitness features

Vision and Scope Modifications

As a result of the survey data we gathered, we altered our requirements as follows:

- iOS users are the priority/majority
  - Xamarin cross-platform app is still a possibility/technology we're looking into, but as ARKit and ARCore are two separate frameworks, a lot of the code would not be shared.
- Dropped support for AR bulletin boards
  - It didn't make a lot of sense to be forced to go to a specific location to access information.
  - If added in a subsequent release, the information should be available regardless of the user's location.

SceneKit Library

SceneKit combines a high-performance rendering engine with a descriptive API for import, manipulation, and rendering of 3D assets.

- **SCNNode**
  - A structural element of a scene graph, representing a position and transform in a 3D coordinate space, to which you can attach geometry, lights, cameras, or other displayable content.
  - Used to create the arrows that will guide the user to their desired location

ARKit Library

Integrates the device's camera and motion features to produce augmented reality experiences.

- **ARSession**
  - A shared object that manages the device camera and motion processing needed for augmented reality experiences
  - Instance is included with an ARSCNView object
  - Requires a session configuration via ARConfiguration
- **ARWorldTrackingConfiguration**
  - Uses the rear facing camera to precisely track a device's position and orientation
- **ARSCNView**
  - Blends virtual 3D content(SCNNode) with the device's camera view of the world

CoreLocation Library

CoreLocation provides services for determining a device's geographic location, altitude, orientation, or position.

- **CLLocationManager**
  - An object that is used to start and stop the delivery of location-related events to your application.
- **CLLocationCoordinate2D**
  - The latitude and longitude associated with a location

Combining it all

- The steps we're following in our prototype:
- Setup a ARSCNView
- Create a SCNNode of a 3D arrow
- Get the user's coordinates
- Calculate the initial bearing from the user's coordinates to the center of the sculpture garden
- Transform our SCNNode to point towards the bearing
- Add the SCNNode to the ARSCNView

```
var sceneView = ARSCNView()

var locationManager = CLLocationManager()

var loc = locationManager.location.coordinate // User's lat/lon

var sculptureGarden = CLLocationCoordinate2D(latitude: 36.971542, longitude: -82.558492)

var bearing = loc.calculateBearing(to: sculptureGarden) // Bearing in radians

var arrowNode = createArrowNode()

arrowNode.transform = SCNMatrix4Mult(arrowNode.transform, SCNMatrix4MakeRotation(bearing, 0.0,
1.0, 0.0)) // Rotate arrowNode around the y-axis by bearing radians

sceneView.scene.rootNode.addChildNode(arrowNode)

extension CLLocationCoordinate2D {

        func calculateBearing(to coordinate: CLLocationCoordinate2D) -> Double {

                let a = sin(coordinate.longitude.toRadians() - longitude.toRadians()) *
        cos(coordinate.latitude.toRadians())

                let b = cos(latitude.toRadians()) * sin(coordinate.latitude.toRadians()) -
        sin(latitude.toRadians()) * cos(coordinate.latitude.toRadians()) *
        cos(coordinate.longitude.toRadians() - longitude.toRadians())

        return atan2(a, b)

        }

}
```

Issues

We're currently setting ARWorldTrackingConfiguration to gravityAndHeading which modifies the coordinate system so the y-axis is parallel to gravity, its x- and z-axes are oriented to compass heading, and its origin is the initial position of the device.

This works great to orientate our node to a real world location, but makes placement of the node harder to determine.

ARKit has an ARAnchor object that can be used for plane detection and object placement. We're looking into detecting flat surfaces directly in front of the user's device and attaching our arrow node to that anchor, so that the arrow will always be visible in the direction the user is facing.