Core Functionality of State Machine

The core loop of the system contains the following components, which function the following ways.

* Local IoT Show Elements
  + The state machine responds to a system of local IoT devices that communicate via a networking solution either directly with the Edge Proxy or with a local server that serves as the interface between the local system and the Edge Proxy.
  + In the present incarnation, each IoT object has a deviceID, and sends entrance and exit Beacon ID data to the state machine in the form deviceID, beaconID.
  + The IoT objects then receive commands from the state machine. In the current examples, commands include playing sound files, triggering lights, or activating vibration on motors. As explained above, commands can be customized for a particular experience.
  + Each device is connected to a character.
  + In the Raven, show elements were IoT lanterns, so lanterns are referred to in the code for IoT devices.
* State Machine
  + The state machine is an Azure-based service that sends commands to the show elements based on state.
  + The state machine receives beacon information from the show elements in the form deviceID, BeaconID.
  + The state machine uses this information to:
    - Assign a character attribute to the viewer for future scene determination
    - Store information about where viewers are for the purpose of determining scene data and commands
  + The state machine runs the core timer of the show. The machine starts the timer at top of show. Events then trigger at particular times from start as established by scene json data.
    - When a scene triggers the system sends a state to all viewers (based on character or deviceID, depending on scene) in that scene with the sceneName.
  + The scenes the state machine has are divided into two types:
    - Ava Scenes
      * Ava Scenes are the system’s terminology for scenes that are fixed, meaning that they always happen at a fixed moment in the show.
      * Ava Scenes always trigger at a particular time from the start of the show.
      * Each Ava Scene has a particular set of characters who are expected to be in it.
        + If a character is not expected to be in Ava scene, they can still experience it if they enter the beacon field in which it is happening during its duration.
      * Each device in the Ava scene is locked into that scene. They no longer use beacon data unrelated to the scene until the Ava scene ends.
    - Roaming Scenes
      * Roaming scenes are for characters that are not assigned to current Ava scenes.
      * Any character not in Ava scene is considered available.
      * The state machine picks a scene for any character that’s currently marked as available based on:
        + What scenes the character has available to them in character data
        + When the character’s next Ava scene is (characters will not get scenes whose duration would prevent them from seeing an Ava scene)
        + What scenes the character has already seen (characters never repeat scenes)
        + What beacon ID the character enters

Core Scene Logic

There are essentially two types of scenes in the system - scenes that are fixed in time (called **Ava Scenes**) and scenes that can happen at different points in a show (called **Roaming Scenes**). The system handles these two scenes differently. Ava scenes always have priority over roaming scenes.

Ava Scenes

Ava scenes always follow these steps:

* Time [T] passes from the last scene event, indicating the start of a scene.
* The system sends commands to all viewers of that scene. They are no longer available for other scenes.
* Steps of the scene advance by timer or by certain characters (i.e. deviceIDs) entering certain beacons.
* If anyone enters the assigned beacons for an Ava scene, they join the scene when the next step occurs and remain in the scene, unavailable for other scenes, until its completion.
* The scene continues through steps until it completes, at which points all characters in the scene become available again. A new Ava scene can start at this point.

Roaming Scenes

Roaming scenes occur while viewers are available for scenes. Roaming scenes are initiated in the following way:

* Whenever a viewer is flagged as available after a scene, they are given a “breath” of time before their next activity.
* The system then checks the following information about the viewer, based on the constantly updated location (i.e. Beacon ID) the viewer has.
  + Where the viewer is
  + When the viewer’s next Ava scene is (Ava scenes ALWAYS take priority over roaming scenes)
  + What roaming scenes the viewer has already done
* Based on these parameters, the system picks a scene that (in this order):
  + Is a scene the viewer’s character can have
  + The viewer has not done before
  + Fits in the time period the viewer has
  + Works with someone in the space with the viewer
* The system triggers the scene for every viewer that will have the scene, marking them as unavailable.
* The viewers in the scene get commands based on that roaming scene.
* The scene ends when it times out. All viewers of the scene enter a new breath and become available again.
* A roaming scene may immediately trigger another scene.

Data Structure

Scene data is split between three files: lanternToCharacter, scene, and character.

LanternToCharacter

This simple JSON file allows lanterns (i.e. IoT devices) to be assigned to characters. A character is a label used by the state machine to determine individual commands sent to that target device.

This is useful to show running because there are times when technical issues will require an IoT device to be swapped out for another one during a show. Character aliases allow you to maintain a user’s status in the show when you switch their IoT element.

Scene JSON

Scenes are organized into series of steps. The core logic of a scene is that it has participants, commands given to those participants, triggers that cause the next step to happen, and a target step to go to when the step is complete.

Scenes can be either Ava or Roaming. The scene JSON has the following components:

* Description: comment field for notes
* DurationSec: the maximum amount of time that a scene will last. The scene automatically advances with duration sec is up
* Type: can be AVA or Roaming
* Participants(AVA only): the characters who are automatically in the Ava scene
* RequiredAva: DEPRECIATED, leave NULL
* Steps: steps are the components of scenes. Scenes move from step to step.
  + Description: comment field for notes on the step
  + Commands: what is sent to ALL PARTICIPANT CHARACTERS in the scene at the start of this step
    - Cue/sound/vibrate/light: these are sample key-value pairs that demonstrate what commands to IoT elements might look like
    - SpecialText: DEPRECIATED
    - DurationSec: how long the state machine waits to check for the trigger to move to the next scene once the command is sent
    - PaddingSec: additional time that the state machine wants before checking to see if advancement should happen
      * We use DurationSec and PaddingSec together because that way we can set DurationSec to the length of a media object and know that PaddingSec is silent time after that fact
  + RequiredID: array of characters that will trigger the next step by entering a beacon field
    - This is NULL if character actions do not trigger the next step
    - ALL means all characters must reach the RequiredBeacons to trigger the next step.
  + RequiredBeacons: array of beacons that can be triggered to enter the next step; all characters in RequiredID must enter the final of ANY of beacons here
  + TimeTriggerSec: the time that passes from the start of the step to enter the next step; NULL means the scene will never end based on time
    - If there are both RequiredIDs/RequiredBeacons and TimeTriggerSec, the scene will advance based on whichever condition is met first
    - Our advice is to never leave TimeTriggerSec null, simply set it high enough that it will only trigger if the beacon-ID combination is clearly not made.
  + OnTriggerNext: the next step that the show moves to
    - If this is the end of a scene, this value should be “NextScene:FirstStep”
    - If null, the show stops after this step.

Character JSON

Character JSON is organized by each character in the show. Each character can have individualized commands for each scene. The character JSON is where each characters individual commands are scripted.

The character JSON has the following components:

* Scenes: this is the list of scenes from the scene JSON. Each character should mirror all of the scenes in the scene Json
  + Description: comment field for the character in the scene
  + Type: AVA or Roaming
    - If AVA, the character is locked into the AVA scene structure
    - If Roaming, the character is available for any roaming scenes in the characters json
  + Steps: this is a reflection of the steps in the scene in the scene JSON
    - Only used when the character is in Ava scenes; otherwise NULL
    - Steps can be omitted if no individual character action is happening in them.
    - DurationSec: DEPRECIATED
    - Commands: commands that ONLY THAT CHARACTER receives
      * light/sound: example commands to the character IoT device
      * DurationSec: DEPRECIATED - this is based on the scene timing
      * PaddingSec: DEPRECIATED - this is based on the scene timing
  + Triggers: these are beacon triggered events in roaming scenes
    - Only used when the character is roaming; otherwise NULL
    - Beacons: the beaconID that triggers this roaming scene
      * Commands: what is sent to THAT SINGLE CHARACTER from the beacon trigger
        + Sound/light: these are sample key-value pairs that demonstrate what commands to IoT elements might look like
        + SpecialText: DEPRECIATED
        + DurationSec: how long the state machine waits to check for the trigger to move to the next scene once the command is sent
        + PaddingSec: additional time that the state machine wants before checking to see if advancement should happen

We use DurationSec and PaddingSec together because that way we can set DurationSec to the length of a media object and know that PaddingSec is silent time after that fact

* Description: DEPRECIATED; leave null