Lab 2: Animate your Buttons and Receive button input

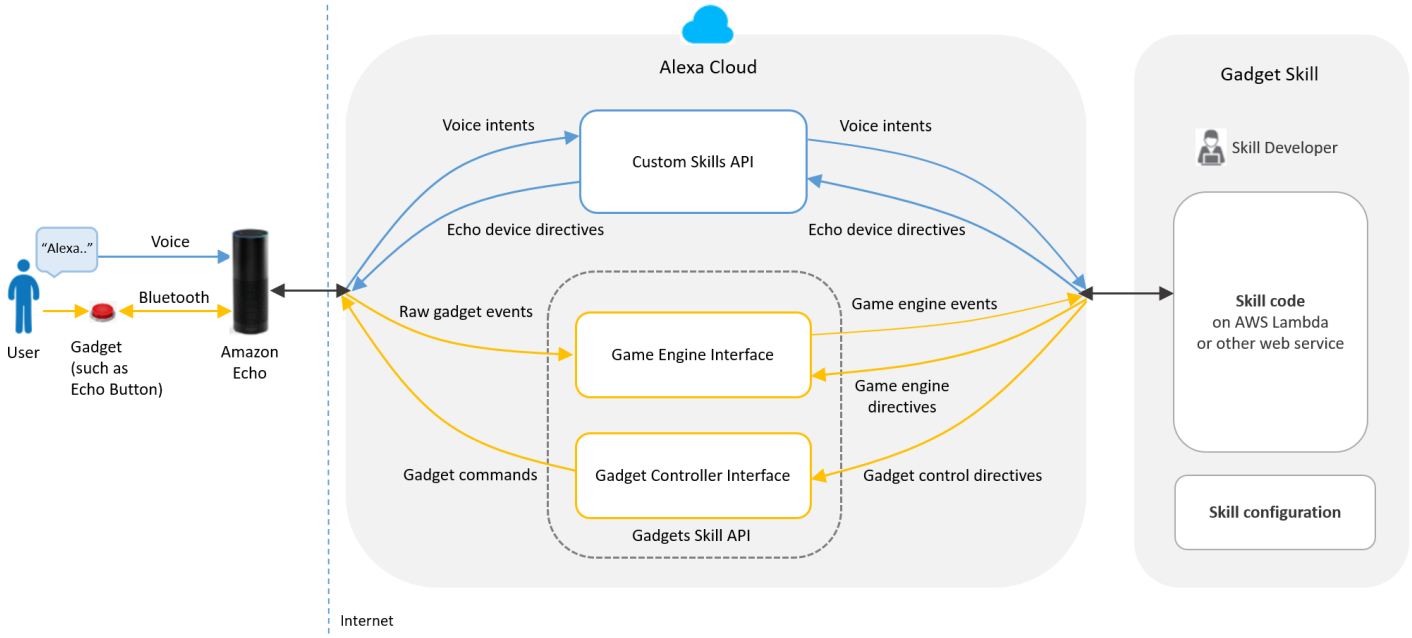
Overview

Using the skill you built in Lab 1, in this lab you will learn how to animate your buttons and receive button events when the buttons are different colors or using different button press patterns using the Alexa Skills Kit SDK for NodeJS.

Hopefully, after the first lab you have completed the following:

1) A working skill that can be invoked using your Echo device
2) An understanding of the account requirements, including the Amazon Developer Portal and AWS.
3) How to create a new skill in the developer portal and link it to a Lambda function in AWS.
4) How to update the Lambda function code by uploading a zip file containing your source code in NodeJS.

Understanding Game interactions using Echo Buttons



Game experiences on Alexa through the Echo Family devices will involve 3 primary modalaties. The first, and most obvious on the platform is voice, where as the game developer you are providing direction and gameplay using Alexa's voice and other sounds that make the experience compelling. Second, is voice input from the user, either being prompted by Alexa as a natural part of the game flow, like telling Alexa which color team you might like to play as, or as a result of an event in the form of a button press, which is the 3rd modality being made available with the Echo Buttons.

How you create your experience is heavily influenced by understanding how, as the developer, you can control the gameplay, timing and prompting of users through the normal voice-based skill interactions, which we call the interaction model, and through two new interfaces being made available to skill developers that control the Echo Buttons appearance and how Echo button events result in calls to your skill's lambda function. The two new interfaces are the **GadgetController** and **GameEngine** which are available to you as a new element in your skill response called **directives.**

The **GameEngine** is a new service in Alexa that listens for button events based on rules you specify from your skills. The **GameEngine** has two primary **directives** available to skill developers, **GameEngine.StartInputHandler** which is a signal to the service layer, based on a set of **Recognizer** objects you specify, that will trigger an event to your lambda function. **GameEngine.StopInputHandler** is a construct for terminating any input handlers you might have running when you need to stop gameplay to address something else in the skill, like if a user asks for help.

The **GadgetController** is a generic Alexa Gadgets interface used to control devices connected to Alexa endpoints, of which the Echo buttons are the first physical manifestation of an Alexa Gadget. Currently, the one **directive** used by skill developers is the **SetLight** directive which allows you to control the button color and animation at any point in the gameplay.

The goal of this section of the lab is to take a look at those two interfaces in detail and look at some examples of **GameEngine.StartInputHandler** using different **Recognizer** objects such that gameplay can be extended beyond a single button press and **GadgetController.SetLight** to set different animations on the buttons, which can be used for things like visual input for being ready to answer a trivia question, like all players buttons lighting up green to signaling an incorrect response from a user, by lighting up that button red, when a question is answered wrong.

If you are familiar with traditional voice modalities when composing Alexa skills, you understand that up until now you would have to prompt the user for input by having Alexa ask a question and leaving the mic open for input. An important construct for developing Echo Button skills is to understand that both **GameEngine** and **GadgetController** directives can be sent with *any* response from Alexa that was a result of a voice interaction or as a result of a button press and that it is up to the skill developers to create the gameplay flow logic in the code using these directives *along with* normal voice-based interactions.

Objectives

The goals for this lab are to understand how to programmatically control button behavior through colors and animations and understand how to modify the **GameEngine** to suit different gameplay. Lab 2 will be split into 2 sections, setting up the **GameEngine** using the **StartInputHandler** directives and animating buttons using the **GadgetController.SetLight** directives.

In this lab we will:

- Examine GameEngine code from Lab 1 and review GameEngine concepts and the required directives/schema your skill's code needs to include.
- Replace the existing directive that responds to any button press event with a roll call directive that only responds when all buttons are pressed.

- Examine existing button controls from Lab 1 and review button control concepts and the required directives/schema that your skill needs to include
- Modify the Lab 1 code to add your own button animations and successfully test.

Prerequisites

This lab requires:

- Access to a notebook computer with Wi-Fi, running Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat).
- An Internet browser such as Chrome, Firefox, or IE9 (previous versions of Internet Explorer are not supported).
- Completion of Lab 1: Build your First Skill: Hello Buttons

Lab 2 Source Files

The entire lab guide including working source code can be downloaded from here
https://s3.amazonaws.com/alx306/ALX306-lab2.zip

After you unzip the directory you should see the following structure

- lab2/
  - ALX306-Lab2.pdf (this guide)
  - HelloButtons/
    - lambda/custom/
      - node_modules/
      - hello-buttons.js
      - package-lock.json
    - archive.zip
    - models/en-US.json
    - skill.json
    -

NOTE: Uploading changes to your code to AWS Lambda

AWS Lambda requires you to upload your skill's code and any dependencies in the code (in our case the *node_modules* directory) as a zip file and declare your function handler as you did in Lab 1.

In the lab exercises below, you need to make changes to the *HelloButtons/lambda/custom/hello-buttons.js* file and upload them to AWS Lambad using the AWS Console.

To create a working zip file, do **not** zip the folder structure, instead zip the content such that the *hello-buttons.js* code is at the root of your zip. To replace the existing *archive.zip* file with your new code, from the *lamba/custom* directory

```
zip -r ../../archive.zip node_modules/ hello-buttons.js
```

Task 1: Examine GameEngine code from Lab 1 and changing the input handler

Overview

In Lab 1 we were able to create an Echo Buttons skill that reacted to Echo Buttons and changed their color when pressed. In order to do this correctly, the skill had to return a **GameEngine.StartInputHandler** directive from the initial **LaunchRequest**, which is the intent that gets invoked when a user launches a skill by saying 'open hello buttons' or 'start hello buttons'. Lets take a look at that code here

```
this.response._addDirective({
      "type": "GameEngine.StartInputHandler",
      "timeout": 30000,
      "recognizers": {
        "button_down_recognizer": {
          type: "match",
          fuzzy: false,
          anchor: "end",
          "pattern": [{
            "action": "down"
          }]
        },
        "button_up_recognizer": {
          type: "match",
          fuzzy: false,
          anchor: "end",
          "pattern": [{
            "action": "up"
          }]
        }
      },
      "events": {
        "button_down_event": {
          "meets": ["button_down_recognizer"],
          "reports": "matches",
          "shouldEndInputHandler": false
        },
        "button_up_event": {
          "meets": ["button_up_recognizer"],
          "reports": "matches",
          "shouldEndInputHandler": false
        },
        "timeout": {
          "meets": ["timed out"],
          "reports": "history",
          "shouldEndInputHandler": true
        }
      }
    });
```

GameEngine.StartInputHandler directive properties

Lets take a look at the 4 root properties of a **GameEngine.StartInputHandler** directive

type	Is always “GameEngine.StartInputHandler”
timeout	A timeout value for the input handler, maximum of 300,000 milliseconds (5 mins)
recognizers	A list of named recognizer objects
events	A list of events to fire when recognizers

You can also review the StartInputHandler schema in our developer documentation preview
https://developer.integ.amazon.com/docs/gadget-skills/receive-echo-button-events.html#start

Summarizing what the directive does

In the sample code above, we can summarize the definition as

1) Fire distinct events, one named ‘button_down_event’ when **any** button is pressed down and another named ‘button_up_event’ when **any** button is released.
2) The firing of these events should not end the current input handler due to **shouldEndInputHandler: false** for both events
3) The input handler shoud timeout after 30 seconds and fire an event called ‘timeout’

What problems does this directive have?

This handler definition will respond to any button press, whether up or down, from any button connected to your Alexa endpoint for 30 seconds. This might be a perfect handler to define for a *first-to-buzz-in* segment of the game, it doesn’t do a great job of identifying *specific* buttons.

To do that, before you have any knowledge of how many buttons are connected to the Alexa endpoint, we will introduce a concept of performing a ‘roll call’ to have players declare their intent to play the game by pressing their buttons.

Introducing Proxies

Proxies (https://developer.integ.amazon.com/docs/gadget-skills/define-echo-button-events.html#proxies) are a concept for the **GameEngine** that allow you to assign arbitrary handles to buttons as their corresponding event

Take a look at the sample in your source directory under lab2.zip/samples/rollCall.json

```
{
  "type": "GameEngine.StartInputHandler",
  "timeout": 10000,
  "comment": "discover exactly two anonymous buttons, or fail",
  "proxies": [ "one", "two"],
  "recognizers": {
    "both_pressed": {
      "type": "match",
      "fuzzy": true,
      "anchor": "start",
      "pattern": [
        {
          "gadgetIds": [ "one" ],
          "action": "down"
        },
        {
          "gadgetIds": [ "two" ],
          "action": "down"
        }
      ]
    }
  },
  "events": {
    "all_in": {
      "meets": [ "both_pressed" ],
      "reports": "matches",
      "shouldEndInputHandler": true
    },
    "timeout": {
      "meets": [ "timed out" ],
      "reports": "history",
      "shouldEndInputHandler": true
    }
  }
}
```

Notice how this example adds a new property to the directive called **Proxies.**
The concept for proxies is that it assigns anonymous handles to the buttons, in the order they were pressed according to the **Pattern** defined.

Summarizing this directive:

1) Fire an event named *all_in* to the lambda function when two button have been pressed within 10 seconds
2) If no two buttons are pressed, fire an event named *timeout*

Lets put this in action by adding it to your code

Task 1.1: Replace the GameEngine.StartInputHandler directive in the LaunchRequest with a Roll Call directive

1) Navigate to HelloButtonsSkill/lambda/custom and edit **hello-buttons.js**
2) Find the **LaunchRequest** handler code starting on line 14
3) Comment out the previous directive and replace with the directive from lab2.zip/samples/rollCall.json
4) This is most easily accomplished by commenting out the function call starting on line 25 **this.response._addDirective(…** and extending the block comment to line 63.
5) To add the directive in lab2.zip/samples/rollCall.json, add a new function call using the same syntax **this.response._addDirective(<block of json from rollCall.json>);**

Task 1.2: Change the spoken response from Alexa

1) Navigate to HelloButtons/lambda/custom and edit **hello-buttons.js**
2) Find the **LaunchRequest** handler code starting on line 14
3) Look for the line that begins with
   a. *this.response.speak("Welcome to the Hello Buttons skill*
4) Change this to an appropriate message for asking players to *roll call.* Something to the effect of '*…to play this round, please press your button…'*

```
// New response for lab 2
this.response.speak('Hi, who is playing? Please press your buttons one at a time');
```

5) Zip and upload your code to AWS Lambda and successfully test the change.
   a. From the HelloButtonsSkill/lambda/custom directory run
   `zip -r ../../archive.zip hello-buttons.js node_modules`

Task 1.3: Add a new item to the switch statement to handle the all_in event

1) Navigate to HelloButtonsSkill/lambda/custom and edit **hello-buttons.js**
2) Find the **GameEngine.InputHandlerEvent** handler code, and the switch statement that can respond to the named events

```
},
'GameEngine.InputHandlerEvent': function() {
  console.log('Received game event', JSON.stringify(this.event, null, 2));
  let gameEngineEvents = this.event.request.events || [];
  for (let i = 0; i < gameEngineEvents.length; i++) {
    let buttonId;
    // in this request type, we'll see one or more incoming events
    // corresponding to the StartInputHandler we sent above
    switch (gameEngineEvents[i].name) {
        // Adding 'all_in' handler for Lab2
```

3) In the switch statement, add code to match on an event named 'all_in' and change the response from Alexa to let the players know they have successfully roll called and are ready to play
   a. e.g. *this.response.speak("Thanks player X, you are ready to play")*

```
        // corresponding to the StartInputHandler we sent above
        switch (gameEngineEvents[i].name) {
            // Adding 'all_in' handler for Lab2
            case 'all_in':
```

4) Log the gadgetID's for each button in the cloudwatch logs
    a. Do this by inspecting the Cloudwatch log for your lambda function and looking at the entire event that was passed to your lambda when the 'all_in' event fired to see how to parse for the right data. You are looking for the gadgetID's for each of the distinct buttons.
    b. Example event shown below


```
"request": {
    "type": "GameEngine.InputHandlerEvent",
    "requestId": "amzn1.echo-api.request.67fcd1c6-4ada-4476-9ee1-2a5f3524b7b6",
    "timestamp": "2017-11-25T23:20:01Z",
    "locale": "en-US",
    "originatingRequestId": "amzn1.echo-api.request.d25b66ed-ec69-4b67-ad89-221317b5c67c",
    "events": [
        {
            "name": "all_in",
            "inputEvents": [
                {
                    "action": "down",
                    "gadgetId": "amzn1.ask.gadget.05RPH7PJG9C61DHI4QR0RLOQOHKOA6VAAT0A2I8M1R
                    "color": "6B1D00",
                    "feature": "press",
                    "timestamp": "2017-11-25T23:20:00.508Z"
                },
                {
                    "gadgetId": "amzn1.ask.gadget.05RPH7PJG9C61DHI4QR0RLOQOHKOA6VAAT0A2I8M1R
                    "timestamp": "2017-11-25T23:20:01.422Z",
                    "color": "C10B00",
                    "feature": "press",
                    "action": "down"
                }
            ]
        }
    ]
```


    c. e.g. Button 1: console.log("Button 1 Found: " + event.request.events.inputEvents[0].gadgetId);
    d. e.g. Button 2: console.log("Button 2 Found: " + event.request.events.inputEvents[1].gadgetId);
5) Zip and upload your code to AWS Lambda and successfully test the change.
    a. From the HelloButtonsSkill/lambda/custom directory run

```
zip -r ../../archive.zip hello-buttons.js node_modules
```

Task 2: Examine GadgetController code from Lab 1 and changing the button colors in response to roll call events

In the hello-buttons.js code for the original example, you can find lines in the skill code like this

```
*/
this.response._addDirective(buildButtonIdleAnimationDirective([buttonId], breathAnimationRed));
this.response._addDirective(buildButtonDownAnimationDirective([buttonId]));
this.response._addDirective(buildButtonUpAnimationDirective([buttonId]));
```

which add *directives* to your skill response in the form of **GameEngine.StartInputHandler** directives or **GadgetController.SetLight** directives. In the above example, the sample code contains conveinience methods for generating some of these boilerplate structures. One important note before moving on, note how you can add more than one animation, in sequence, to a single response to create more compelling visual cues and triggers for players.

Taking a look at the what actually gets generated, we see the following function for *buildButtonIdleAnimationDirective* which returns a JSON object representing the **GadgetController.SetLight** directive to be added to your skill's response

```
// build idle animation directive
const buildButtonIdleAnimationDirective = function(targetGadgets, animation) {
  return {
    "type": "GadgetController.SetLight",
    "version": 1,
    "targetGadgets": targetGadgets,
    "parameters": {
      "animations": [{
        "repeat": 100,
        "targetLights": ["1"],
        "sequence": animation
      }],
      "triggerEvent": "none",
      "triggerEventTimeMs": 0
    }
  }
};
```

Taking a look at the properties

type	Always 'GadgetController.SetLight'
version	Always set to 1
targetGadgets	A list of gadget ID's to play this animation on
parameters	Contains list of **animations** objects, a **triggerEvent** which is one of 'buttonUp', 'buttonDown' or 'none', and a **triggetEventTimeMs** which is how long to wait after the **triggerEvent** to play the **animation**

animations object properties

repeat	How many times to play the animation
targetLights	Always set to 1, hardcoded for now to indicate the LED set on Echo Buttons you can manipulate from a skill
sequence	A descriptive property set for the animation

sequence object properties

durationMs	How many millseconds the animation should last
color	Hex representation of the color
intensity	0-255 level for button brighness
blend	Boolean value to fade the effect or not

Task 2.1: Create a new button animation function to animate the buttons when the 'all_in' event is fired from Task 1.3

1) Navigate to *HelloButtonsSkill/lambda/custom/hello-buttons.js*
2) Find the switch statement you added in **Task 1.3** that responded to the '**all_in**' event you created in **Task 1.2**
3) Add code to iterate through the button events

```
// iterate through the button events
for (let j = 0; j < gameEngineEvents[i].inputEvents.length; j++){
```

4) Generate a new animation directive by calling the *buildBreathAnimation()* function

```
var buttonCode = colors[buttonColor];
// build a breath animation
var breathAnimation = buildBreathAnimation('000000', '0000FF', 30, 1200);
```

Parameters*: buildBreathAnimation(startColor, endColor, repeat, timePerAnimation)*

5) Generate an animation object calling *buildButtonAnimationDirective([], animation)*

```
// add the animation to the response
this.response._addDirective(buildButtonIdleAnimationDirective([buttonId], breathAnimation));
```

Parameters: *buildButtonIdleAnimationDirective(array of button Id's, animation object)*

6) Wrap the function call above in a call to *addDirective*

```
// add the animation to the response
this.response._addDirective(buildButtonIdleAnimationDirective([buttonId], breathAnimation));
```

Parameters: *this.response._addDirective(generated json from buildButtonIdleAnimationDirective)*

7) Get creative with your animation, try making it flash, fade, change colors (hint! requires multiple animations objects in response)
8) Zip and upload your code to AWS Lambda and successfully test the change.
    - From the HelloButtonsSkill/lambda/custom directory run
      `zip -r ../../archive.zip hello-buttons.js node_modules`

Lab 2 Complete

Congratulations! You have completed Lab 2 of the Echo Buttons workshop and AWS RE:Invent!

In this lab, you learned:

- What role the **GameEngine.StartInputHandler** directive plays in setting up the Game Engine to listen for button events.
- How to define named **recognizer** objects as components to that directive's definition
- How to define named **events** that result in request events to your lambda when the **meets** criteria is met

- How to add a new handler to you skill's lambda function to handle the new **event** type.
- The concept of a roll call handler to register players for a game.
- How to store button data for reuse in **GadgetController.SetLight** directives
- How to add your own **GadgetController.SetLight** directive(s) to the skills response.
- How to manipulate the **GadgetController.SetLight** directive to create different button animations using colors, repeats, and possibly using multiple animations in sequence.