GameToSpeech 2020.1 prototype report

David Libeau

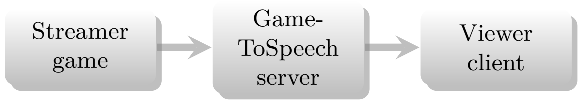
October 22, 2020

1 Introduction

A minimal prototype has been created in order to demonstrate a possible way of exporting data out of a game for generating audio description. *GameToSpeech* is the brand name of the technological side of the PhD project of David Libeau. This PhD aims to study a method for generating real-time audio description for live streamed gaming [1]. This report describes the key components of the first prototype of *GameToSpeech*.

2 Project architecture

GameToSpeech is built like a basic client–server model. The streamer and the viewer could be considered as clients and the server is *GameToSpeech*. The server role is to dispatch the streamer's data to the viewers clients. The viewers cannot send back information to the streamer through the *GameToSpeech* server. In a way, we can also consider the *GameToSpeech* server as a router.



The *GameToSpeech* server is coded in JavaScript thanks to NodeJS. It transmit data to the viewer client through WebSocket. It deploy an API where the streamer game can send data through HTTP.

3 Sending data out of a Unity game

In this prototype, the streamer game is simulate thanks to a Unity project build upon the free 3D Game Kit asset [2]. In this Unity project, a C script has been added providing a mean to export data out of the game.

3.1 The Unity script

The main *GameToSpeech* script (named *GameToSpeechManager* in Unity) is made of a simple *sendToServer* function.

This method is accepting two parameters. The first one is the category, needed in order to classify the data, and the second is the value, the data content itself in JSON format. The *sendToServer* function is sending this data to the *GameToSpeech* server by doing a HTTP GET on the address *https://api.gametospeech.com/* with a route build with the first parent as the game ID and the second parent as the channel. The channel value is identifying the streamer so it needs to be unique to the streamer. The game ID is, at the opposite, identify the game played by the streamer. In this prototype the game ID was *unity* and the channel *david*.

You can find below the main *GameToSpeech* Unity script's code.

```
using System.Collections;
using System.Collections.Generic;
using System.Text;
using UnityEngine;
using UnityEngine.Networking;

public class GameToSpeechManager : MonoBehaviour
{

    public string gameId;
    public string channel;

    public void sendToServer(string category, string data)
    {
        StartCoroutine(serverSend(category, data));
    }

    IEnumerator serverSend(string category, string data){
        string json = "{\"category\":\""+category+"\",\"value\":\""+data+"\"}";
        Debug.Log(json);
        UnityWebRequest www = UnityWebRequest.Get("http://api.gametospeech.com/"+
            gameId+"/"+channel+ "/?data="+json);
        yield return www.SendWebRequest();

        if(www.isNetworkError || www.isHttpError) {
            Debug.Log(www.error);
        }
        else {
            Debug.Log("OK");
        }
    }
}
```

3.2 Using the script

The *GameToSpeechManager* itself is useless if any other script is calling the *sendToServer* function.

First, in the game a *GameToSpeech* object has to be created with the *GameToSpeechManager* script attached. Next, in other game's scripts, the *sendToServer* function will be callable with a simple line.

Here is an example where the game is sending the value of the variable *text* as a dialog to the server.

```
GameObject.Find("GameToSpeech").GetComponent<GameToSpeechManager>()
   .sendToServer("dialog", text);
```

4 The *GameToSpeech* server

As previously mentioned, the server is coded in Javascript thanks to NodeJS.

The script has two main routes. The game's API and the page for the end-user. The API endpoint is build with the game ID and channel name. For this prototype, it is only working with *unity* as game ID. When the script is receiving data, it tries to decode as it is JSON. Then it sends the data on the socket with the namespace of the channel name. Thanks to that only the users connected to that namespace will receive the data. The script can also initialize the socket if it detects that it was not been created before.

The client user will have to connect to the */listen* route with the channel name as second parent in the path. When he/she will load this url, an HTML page will show up with minimal content, but with a Javascript script run on the browser. This script will try to connect to the socket with the appropriate namespace in order to receive data of the right streamer. When data is received, the browser will use an internal text-to-speech software to transform the data value into speech.

Here is a sample of the NodeJS script run on the server. This script is using ExpressJS and SocketIO modules.

```
//the init of express and socket.io is not shown here

//init channel
var channels = {};
function initChannel(channelName){
    channels[channelName] = {};
    channels[channelName]['socket'] = io.of('/'+channelName);
    channels[channelName]['socket'].on('connection', socket => {
        console.log('Socket connected on channel '+channelName);
    });
}
```

```
//API (receiving data from the game)
app.use('/:gameId/:channel',
    function (req, res, next) {
        if (req.params.gameId == 'unity') {
            console.log(req.params);
            var data = JSON.parse(req.query.data);
            console.log(data);

            if(!channels[req.params.channel]){
                initChannel(req.params.channel);
            }
            channels[req.params.channel]['socket'].emit('message', data);

            res.send('OK');
        } else {
            next();
        }
    });

//render and serv the client HTML page
app.get('/listen/:channel', (req, res) => {
    res.render('listen', {
        channel: req.params.channel
    });
});
```

The code below is a sample of the Javascript script run on the client browser.

```
//the HTML page is not shown here

var socket = io('/<%= channel %>');
socket.on('connect', () => {
    console.log('connect');
    document.getElementById('status').innerHTML = 'You are connected to channel <%= channel %>.';
});
socket.on('disconnect', () => {
    console.log('disconnect');
    document.getElementById('status').innerHTML = 'You have been disconected.';
});
socket.on('message', data => {
    console.log(data);
    switch (data.category) {
        case 'dialog':
            speak(data.value, 'Samantha');
            break;
        case 'health':
            speak(data.value + " health points", 'Alex');
            break;
    }
});
```

5 Conclusion

This first working prototype is very simple. It only transmit Unity game's data to an client web browser which automatically transform it into speech. This is a basic audio description proof of concept which need improvements and further studies.

You can find the complete source code at :

- https://framagit.org/DavidLibeau/gametospeech-server
- https://framagit.org/DavidLibeau/gametospeech-unity

If you cannot install and run the whole project, several demo videos are available upon request.

References

[1] David Libeau. *Generating real-time audio description for live streamed gaming.* 2020. URL: `https://www.researchgate.net/publication/342051779_Generating_real-time_audio_description_for_live_streamed_gaming`.

[2] July 2020. URL: `https://assetstore.unity.com/packages/templates/tutorials/3d-game-kit-115747`.