Using Audio for Player Interaction in HTML5 Games

Jefferson Torres de Freitas* Ernesto Trajano de Lima† Artur de Oliveira da Rocha Franco‡
José Gilvan Rodrigues Maia§

Federal University of Ceará, Virtual University Institute, Brazil

ABSTRACT

Advances on computing hardware and software throughout the last decades drastically molded the prospect of the entertainment industry. Within this context, a prominent development was the emergence of interactive computer-driven products. The introduction of specialized game controllers established a paradigm in which the player explores her hand skills for fast and flexible interaction with games. Sophisticated interaction techniques supporting body gestures and voice commands were introduced later. However, most successful technologies in this field are typically bound to specific hardware. With the advent of HTML5 and recent JavaScript performance enhancements, most modern web browsers already provide refined programmable components for audio processing that support sound analysis and synthesis in real-time. This paper presents an investigation on how audio capture and processing can be used as an input device for user interaction in web games. An API supporting programmable game interaction driven by audio for HTML5 was designed and implemented using JavaScript. An actual prototype game was developed using this API for evaluation purposes.

Keywords: Audio, Web games, Interaction.

1 INTRODUCTION

Entertainment became a central business in the last decades. Actually, numerous technological advances brought forth new interaction possibilities that emerged as a new kind of products that changed the entertainment industry radically: digital games. As gaming computers and consoles became more and more popular, user input apparatus like joypads allowed people to use both hands for relatively comfortable and fast interaction. Moreover, these devices were proven effective for adapting to different games' specific needs by using clever programming techniques.

Other interaction devices were developed using new hardware and efficient data analysis algorithms for real-time, reliable recognition of landmark blobs that are prominent from the background in infrared images due to specially crafted materials and controlled lighting. Recent developments accounted for an even more sophisticated and more "natural" interaction by identifying the human body and its parts, making it possible for the user to interact by expressing a myriad of inputs coming from her eye gaze, hand gestures and voice commands, for example. For example, in first, karaokes were using simple techniques of audio processing [18], at the turn of century twenty-first century like Kinect, Wiimote and Oculus Rift which have a important applications like digital heritage [21].

*e-mail: jeffersontorres.freitas@gmail.com
†e-mail: ernesto@virtual.ufc.br
‡e-mail: arturoliveira@virtual.ufc.br
§e-mail: gilvan@virtual.ufc.br

However, such interactive forms are bound to specific and usually expensive hardware. Moreover, most computer vision and image sensing techniques resort to infrared illumination which may damage the user's retina due to prolonged exposure or when the device operates outside accessible and maximum permissible limits due to malfunctioning.

Despite those drawbacks, using a subset of such interactive technologies is becoming feasible in recent computers as available processing power increases and domain-specific languages evolve.

Modern web browsers already have improved, standardised audio engines that allow for real time sound analysis and synthesis. In this paper, we investigate usage of programmable HTML5 audio interfaces for real time user interaction within the context of web games by means of signal processing techniques applied to audio captured continuously.

These are the main contributions of this paper:

- We investigate how real-time audio processing can provide user inputs for web games.
- We present a an API specifically designed to support building such games.
- We evaluate the effectiveness of this approach by carrying out experimentation over a prototype web game developed using the proposed API requiring well-timed inputs in real-time.
- This prototype game runs on both Desktop and Android platforms, which corroborates that cross-platform web games can benefit from player inputs provided via audio processing.

This work is organized in the following manner. Section 2 presents a preliminary discussion about audio, games and the web. Audio processing definitions and subjacent mathematical models are presented in Section 3 as well the APIs available within the context of HTML5 development. Similar investigations are discussed throughout Section 4. The proposed API, its design and implementation details are covered in Section 5. Experimental evaluation and discussion about the results are the matter for Sections 6 , and 7, respectively. Finally, conclusions about this work and future research directions are pointed out in Section 8

2 GAMES, AUDIO AND THE WEB

Audio has been proved fundamental not only for digital games but also for many other digital media. This perceptual stimulus is an immersive agent that helps breaking the boundaries between users and characters [1] [14] [21].

2.1 Audio in Games

The first popular games that had the audio as the main form of interaction were in karaoke machines [7]. The processing techniques were quite primitive in the early products, taking into account only the energy of the signal for evaluating the singer's performance and, in some cases, no processing was available at all since scores were assigned randomly in some systems [18].

In 2000, Nintento released Voice Recognition Unit (VRU) as an accessory for the Nintendo 64 console. The VRU was as capable of comprehending human voice in a region-dependent fashion and its calibration was set to cope better with children's high-pitched voices because they were the target audience of *"Hey You, Pikachu!"*, the mainstream game supporting the technology.

Other interaction forms such as the Wiimote ®, the Kinect ®, and Oculus Rift ®and acquired substantial prominence despite none of those being established permanently as a *de facto* standard for interaction in games.

Games have used sound to indicate a number of different things. Here is a brief overview about how audio is used in electronic games:

- Simon was a hugely popular toy in the late 70's that used colored buttons corresponding to tones to challenge the player's memory. Basically, the player must memorize and reproduce a sequence that increases at each interaction [4].
- Super Mario World by Nintendo (1990) played background music faster when the protagonist is about to face death due to a timeout.
- Silent Hill, released in 1999, used broken, mute piano keys in an intriguing situation to challenge the player's ability to solve puzzles.
- Modern gaming consoles support voice command recognition. However, these are usually applied for interacting with menus instead of proper games.
- More recently, *A Blind Legend*[1] goes a level beyond. This is considered by many as the first *"audio-only action/adventure mobile video game"* since it does not resort to a visual representation. A huge sound design and programming effort is necessary to bring correct feedback to the player that interacts using standard the gesture recognition available on mobile platforms.

On the other hand, other products assume that audio must be used not only as a stimulus, but also as an user input form. There are games tn the market where there player interacts through audio, such as SingStar ®[2] and Karaoke Revolution [18]. Such games were inspired by the genre karaoke, which had its analog source, whose creation is attributed to Daisuke Inoue [7].

2.2 Audio Processing for the Web

Nevertheless, despite the remarkable technological advances in computing and the advent of alternative entries using cameras and microphones specially developed to a high degree of complexity, audio has been explored to a small extent as a source of user input.

Moreover, recent additions to the web software stack provide new opportunities for interactive hypermedia, specially with the advent of the HTML5 standard [6] that has been incorporated into modern browsers. As the HTML5 standard emerged, new opportunities for the development of robust and efficient APIs that support audio processing.

This powerful, widely adopted standard includes interactions with different media, protocols and programming languages. An HTML5 page can process video streams and audio captured directly from devices possibly available on the user's hardware [10]. Moreover, WebGL represents a significative boost on both graphics performance and visual programming flexibility with the introduction of programable visual effects via shaders that are coded in a high-level language and these interact with the web page by means of scripts.

It is also worth noting that since WebGL closely resembles the compact and efficient GL ES API widely incorporated into mobile devices such as smartphones, tablets and smart TVs, HTML5-compliant web pages can display interactive graphics that benefit from hardware acceleration in those devices. On the other hand, HTML5 also features the Canvas API [5] for programming two-dimensional graphics based on immediate-mode drawing. Both WebGL and the HTML5 Canvas are epitomes of flexibility and performance necessary for developing minimally convincing web games.

This possibility, combined with the evolution of JavaScript (JS) technology enabled the mass production of content and also domain-specific application programming interfaces (APIs), such as Web Audio API4 [3]. Modern browsers make extensive use of JavaScript as logical page control, which is becoming a popular alternative since such implementations do not require any plug-ins and other external programs. JS controls the many components available for rendering graphics and sounds, as well as the representation of the web page itself.

Some APIs, such as the Web Speech API, have complex built-in algorithms such as the voice recognition available in the Chrome browser. Such multimedia resources became more common even other for Web standards, as used in CSS Aural [4], which is now obsolete. However, this initiative has led to the proposition of a new standard called *CSS Speech* [5] that is still undergoing development at the time when this paper as written.

This paper focuses on audio decomposition methods applicable to in-game interaction that benefit from audio processing API supported by browsers. As far as we could search, there are no specific APIs for using real-time audio as input for web games. This can be explained by considering that implementations of the HTML5 are somewhat recent. From the developer's perspective which aims to offer innovative products, it is essential to research on possibilities that can be explored in order to enable this new kind of interactions in games. However, using these APIs requires standard-compliant implementations that are not yet available widely.

3 THEORY

Our world is filled with sounds that most living beings perceive and analyze in order to act in an environment. However, the term sound is more generic than audio since sound can be provoked by any source. Audio processing is a term widely employed by technicians and researchers in the signal processing field since this word refers to a special category of sounds originating from a recording, transmission or electronic device.

Despite the numerous efforts made over the years in this field, some challenges remain in existence since those were faced for the first time by researchers and enthusiast practitioners, specially considering the general case when the application's environment is uncontrolled [19] [17]. Examples of those are the extraction of sounds of interest and noise suppression that hinder the effectiveness of audio for a specific situation, such as trying to extract the human voice from audio recorded at a beach [9].

Therefore it is necessary to bear in mind that obtaining a digital audio processing implementation requires some background theoretical aspects regarding the nature of this task.

3.1 Physics

Sound may be defined as a biological perception of countless mechanical waves that propagate from sound sources through a

[1] http://www.ablindlegend.com/
[2] https://www.singstar.com/
[3] https://www.w3.org/TR/webaudio/
[4] http://www.w3schools.com/cssref/css_ref_aural.asp
[5] https://www.w3.org/TR/css3speech/

medium so that they finally arrive at the auditive apparatus. This comprises an extremely complex process even assuming simplifications such as an ideal and uniform physical medium with constant, medium-dependent speed [8].

Consequently, how the wave actually spreads through space is an intricate physical process that varies in time and depends on both the shape and the physical properties of virtually every single object on a propagation environment.

Effective analysis of this phenomenon requires knowledge of signal processing concepts, which can be divided into two categories according to the nature of the signal: analog signals and digital signals [12]. This work is bound to consider only digital discrete signals that are, in fact, a time series composed by samples represented by a sequence of values, each of them representing a point in time. Moreover, samples are considered to be equally spaced in time, i.e., the sampling rate is constant.

This means that audio has an intrinsic unidimensional nature, i.e., an ideal processing mechanism must be endowed with the ability of deftly identify and separate the constituent features that make up the time series as a whole.

The latter statement explains the difficulty in obtaining reliable computational methods for digital audio processing in poorly controlled environments, making this a highly challenging task when compared to its biological counterpart. Hence, it becomes clear that resorting to an alternative representation may better suit automatic processing by computer algorithms.

3.2 Mathematical Models

Representing audio as a time series presents many drawbacks that can be overcome by adopting a suitable mathematical model. Fourier series represent signals as a sum of infinite terms, each of them being a periodic function with a given frequency in terms of sines and cosines. This representation is of utmost importance for audio processing since the signal is observed as a decomposition over a wide range of frequencies [15]. Moreover, pure tones are modelled as periodic functions [8].

The Discrete Fourier Transform (DFT) is a simple procedure to determine the contents of the sample frequency, but is inefficient and costly to use a complex algorithm asymptotic $O(n^2)$, such n is the number of samples. Although its polynomial complexity, the required processing time grows very quickly as the number of sample points increase, having in mind that usually tens of thousands samples are used per second to represent audio [14] [15].

This limitation is often avoided by adopting the Fast Fourier Transform (FFT) described the technique proposed by Cooley-Tukey [2]. The FFT is closely based on the definition of DFT, but it becomes highly efficient in relation to its traditional counterpart by presenting a series of optimizations which are applicable assuming the number of samples is a power of 2 [12]. The algorithm for calculation of the FFT is usually asymptotic complexity O (nlog (n)), with significantly better performance than the DFT and it also yields greater precision in many cases.

Moreover, real-world signals contain undesired sounds, also known as noise. This spurious information needs to be suppressed or diminished, which is usually performed by filtering the signal. It is also important to consider that uncontrolled sound power levels may injure the users' hearing apparatus.. Taking the hearing threshold as a reference, sounds under 50dB are not considered harmful to hearing, while a normal conversation equates to 60dB. Sounds between 80dB and 100dB, in their turn, are considered troublesome and stressful, thus being harmful to health. Sound intensities equal to or greater than 120dB, known as the pain threshold, cause irreversible damage to the inner ear [8].

4 Related Work

4.1 Audio as Input for Games

Tsai and Lee [18], in their experiments to automate the evaluation of performance within karaoke systems, take into account characteristics such as pitch, volume or sound intensity, and rhythm of notes sung, as well the duration of each note in order to optimize these systems. Starting from those characteristics, a comparison is made between singer recordings and MIDI files in order to evaluate sound harmony. In this case, harmony is defined by the sequence of notes sung in a correct tone along with adequate duration of each note.

Lima et al. [11] investigated using computer vision and voice command recognition applied to interact with games 3D environments. They tackle voice recognition from a machine learning approach using time-consuming baseline learning techniques K-Nearest Neighbors [3] and Support Vector Machines [20]. However, evaluation as presented in their paper did not present enough evidence supporting that an effective recognition was obtained.

Silva and Pozzer [16] applied voice commands for controlling interactive story generation by analyzing sentences emitted by users. Sentences are represented as binary trees for syntactical that correctly evaluates about 80% to 90% of the users' requisitions using both conventional and Kinect microphones for audio capture. This result can be classified as satisfactory since the system does not recognize some words.

Antas et al. [13] propose an approach that is similar to the latter, but considering typical point-and-click interaction for handicapped players. These authors employed bootstrapping to improve recognition performance, achieving an error rate inferior to 5%.

There Came an Echo[6] is a real-time strategy game focusing on the narrative released in February 2015. In this game, the player uses her voice to issue commands for controlling military units on the battlefield. The technology employed for speech recognition in this game requires certain conditions to ensure an acceptable functioning and implementation of an acoustic model suitable to the user accent. Smooth and paused speaker diction are necessary since the game requires real-time interaction without allowing interruptions in combat situations. Some sophisticated audio equipment seem incompatible with the technology for some mysterious reason, such as speaking near the microphone or advanced capture configurations.

4.2 Audio APIs for the Web

Web Audio API [7] is proposed for both the loading audio files via AJAX and the handling of frequency ranges. This allows for more advanced operations, such as sound synthesis, signal analysis, data conversion, signal filtering, spatialization of the sound, etc. This API also uses audiovisual elements of HTML5 for communication with multimedia devices for real-time reproduction. The API is composed by 33 programming interfaces, 2 of which are obsolete, each one with a specific functionality, connected to define a based routing graphs for general rendering of the sound. The most outstanding interfaces are depicted by Figure 1.

Timbre.js [8] is a library that provide functional audio processing and synthesis for web applications based in modern JavaScript technologies, such as JQuery and Node.js. By "Functional" it is meant that the API uses functional programming concepts for defining processing elements and sound synthesis. The library is composed of "Timbre Objects", referred to as "T-Objects", whose structure is the same Web Audio API. According to the author, this architecture has been proposed to address the next generation of audio processing technologies for web.

[6] http://www.playiridium.com/games
[7] https://www.w3.org/TR/webaudio/
[8] http://mohayonao.github.io/timbre.js/

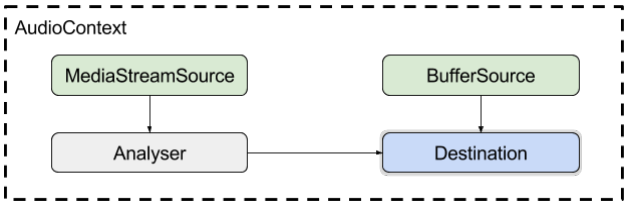

Figure 1: Programmable audio interfaces from Web Audio API.

The most outstanding feature of *Timbre.js* is to provide means to synthesize sounds with relative simplicity. Moreover, using this API only requires the addition of a JS library to the page, which can be done with just one line of code. Given these features, *Timbre.js* also has considerable value for use as a teaching tool. However, the programming interface available for audio processing is unintuitive despite being quite powerful, since it is necessary to master a very intricate functional notation when, on the other hand, the debugging support processing is quite limited.

Web Speech API [9] is an API that enables the incorporation of voice recognition on web pages. It allows developers to use speech recognition how input generating textual outputs. The programming model is little effective, because the own programmer must select the words that interest you from a list reported by API through a callback mechanism. In this way, the API fails on the level of abstraction of this problem by the developer.

5 Proposed Framework

In this section, it is presented the API model applied in modern web browsers and code structure responsible for processing the data obtained through the Web Audio re-utilized interfaces. The API uses JavaScript, a language in which were developed the modules of configuration, requisition and control of multimedia devices, data analysis, statistical processing for making decisions of game status, data clearance for audio context control, environment noise calibration, and audio loading.

5.1 Architectural Design

The proposed API in this paper is structured in modules. It was used JavaScript to develop the API because the language is accepted in virtually all modern browsers that follow HTML5. Also, recent advances in JavaScript translators had conferred better performance in execution time, making possible to do tasks more computationally intense, such as processing audio signals.

It was created a flowchart of the interfaces utilized on the Web Audio specification present on the HTML5 standard (Figure 2).

5.1.1 Configuration Module

The basic settings for the API are encapsulated by this module, as illustrated by Listing 1.

```
1  var _setup = function _setup( stream ) {
2
3    _mediaSource = _audioCtx.createMediaStreamSource( stream );
4    _analyser.fftSize = 1024;
5    _analyser.smoothingTimeConstant = 0;
6    _fftArray = new Uint8Array( _analyser.frequencyBinCount );
7    _mediaSource.connect( _analyser );
8
9  }
```

Listing 1: Configuration script for setting most low-level processing options.

[9] https://dvcs.w3.org/hg/speech-api/raw-file/tip/speechapi.html

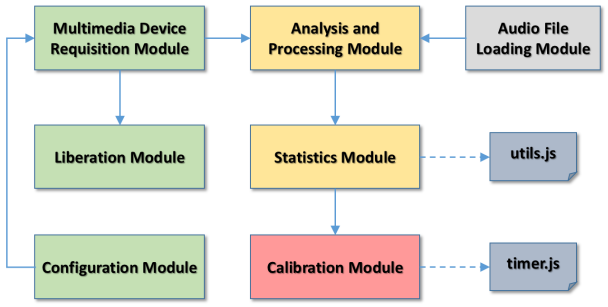

Figure 2: Proposed architecture and its modules for: (left, in green) configuration, multimedia device requisition and liberation; (middle, in yellow) analysis and statistics; (middle bottom, in red) callibration; (top right, in gray) audio file loading; and (bottom right, in light blue) utility functions. These modules are built on top of Web Audio API interfaces.

The interface *_mediaSoruce* is generated and it receives a stream as a parameter. The stream is assigned on the moment that is granted the use permission of the multimedia device. The parameters *fftSize* and *smoothingTimeConstant* define the FFT window's size and the delay of the time which the samples will be processed, respectively. It is important to mention that from the various types of windowing of Fourier transform, the Web Audio only uses the *Blackman window*. The windows are useful to minimize the effects of a phenomenon known as spectral leak. The spectral leak commonly occurs in truncated waves responsible for generating discontinuities in the signal. These discontinuities appear in the FFT amplitudes as components of high frequencies that are not present in the original signal. There are several different types of windows provided with specific characteristics that may be applied to a signal. The Blackman window has its use widespread [10].

The object *_fftArray* is the structure in which the audio samples are stored, so that only half of them are considered to prevent mirroring property of the signal. At the end, a connection is made with *_analyser* interface, so that all the captured information is processed.

5.1.2 Multimedia Device Requisition Module

The request for the user to use a multimedia device is performed by the *liveInput()* method. In this case it is requested access to the microphone. If so, the configuration module is called as a success callback, as shown in Listing 2. Otherwise, an error message is displayed. Additionally, this module is configured to match their operation with major browsers at the time of this work, such as Firefox, Chrome and Opera.

[10] http://www.ni.com/whitepaper/4844/en/

```
this.liveInput = function liveInput() {
 var constrains = window.constrains = { audio: true, video: false };

 navigator.mediaDevices.getUserMedia( constrains )
 .then( function _onSuccess( stream ) {

      _setup( stream );

     } )
     .catch( function _onError( error ) {

      console.log( error.name + ': ' + error.message );
     } );
}
```

Listing 2: The API gently asks for user confirmation in order to behave correctly.

5.1.3 Liberation module

The Liberation Module is responsible for liberating the audio context and its interfaces from the web browser. This module is necessary to ensure a good functioning of the API in order to promptly release resources that become idle, which helps to avoid harmful interference with other multimedia applications that are perhaps competing for the same resources. This approach is important to provide compatibility with mobile devices and it is depicted by Listing 3.

```
this.free = function free() {

 if( navigator.getUserMedia )
  navigator.getUserMedia = undefined;

 if( _audioCtx != undefined ) {
  _analyser.disconnect();
  _mediaSource.disconnect();
  _audioCtx = undefined;

 }

}
```

Listing 3: Audio context liberation avoids undesirable erratic states after interaction.

5.1.4 Real-time Analysis and Processing Module

A Fast Fourier Transform is applied on this module to convert the sampling in the time domain to the frequency domain. After that, the structure *_fftArray* is proved and returned with the newly processed frequencies. This is shown in Listing 4

```
this.realTimeAnalysis = function realTimeAnalysis() {

 _analyser.getByteFrequencyData( _fftArray );

 return _fftArray;

}
```

Listing 4: Overview of the analysis and processing module. A callback function provides programmer-defined code with FFT data.

5.1.5 Statistical module

Module responsible for calculating representative values of the samples on the frequency domain. Based on real-time processing, it calculates the overall average of the samples. Moreover, this module is built on top of the *utils.js* script, which provides prototype properties and methods that are inherited from other instances, such as the mean method described in Listing 5.

```
Uint8Array.prototype.mean = function mean() {

 var result = 0;

 for( var i = 0; i < this.length; i++ )
  result += this[ i ];

 return ( result / this.length );

}

this.getFrequencyAverage = function getFrequencyAverage() {

 var array = this.realTimeAnalysis();

 return array.mean();

}
```

Listing 5: Statistics are used for processing incoming signals in order to detect salient features such loud sounds or distinctive frequencies.

The mean method is responsible for compute the average of the samples stored in an array of unsigned 8-bit integer. The *getFrequencyAverage()* method invokes the analysis module and real-time processing and returns the average of the signal. This average is considered as the loudness of signal set decibels.

5.1.6 Calibration Module

The Calibration Module also depends on the *timer.js* script that is responsible for creating a countdown timer, which is defined in seconds. Basically, with the use of other modules, the overall average of ambient noise denoted by parameter *_noiseAverage* is calculated and the obtained result is measured in decibels. This result is then stored as the parameter *ambientNoise*. This is depicted by the code snippet in Listing 6.

```
this.calibrate = function calibrate() {

 // avoids already calibrated state
 if( this.isCalibrate )
   return;

 // starts a 5s countdown limiting the setup
 if( !_timer.isStarted ) {
   _timer.start();
   _timer.counter = setInterval( countdown, 1000 );

   _noiseAverage = 0;
 }

 if( _timer.timeRemaining > 0 )
   _noiseAverage += this.getFrequencyAverage();
 else if( !this.isCalibrate ) {
   // restarts if needed
   _timer = new Timer( 5 );
   var time = _timer.countDown * FPS;
   // computes the noise over time
   this.ambientNoise = _noiseAverage / time;
   // (...)
 }
```

Listing 6: Callibration module's logic uses timeouts for considering a given time period which is usually set to 5 seconds.

Depending on the result, a new calibration can be done as the user sees fit. The parameter *isCalibrate* indicates whether the calibration was done successfully. It is important to notice that the estimated average is equal to the sound intensity.

5.1.7 Audio File loading Module

This module is responsible for loading temporal series representing audio clips. This process is performed in the following man-

ner, given an asynchronous *_xhr* instance representing a standard HTTP request that accesses underlying data. Several events are implemented, including the main event called *onload()*, so that audio file data contained in a *ArrayBuffer* is decoded asynchronously and then these data are loaded from the response attribute defined by the *responseType*. The decoded *AudioBuffer* is then re-sampled to *AudioContext*'s sampling rate, which, by default, is set to 48kHz. Finally, the result is assigned, by callback, to the buffer of a sound source. This process is shown in Listing 7. It is important to note that very large files can cause sluggishness, including errors in loading.

```
this.loadSound = function loadSound( url ) {

      _xhr = new XMLHttpRequest();

      //(...)

      // Complete callback
      _xhr.onload = function onload() {

       var audio = _xhr.response;
       //Decode the data
       _audioCtx.decodeAudioData( audio, function( decodedAudio ) {
        _bufferSource.buffer = decodedAudio;
       });
      }

      //(...)
     }
```

Listing 7: Audio loading module implementation.

5.1.8 Audio Recording Module

This module is utterly useful for testing and experimentation purposes. For example, it is possible to record audio from a user interaction in order to reproduce any detected processing problems and address them more effectively by inserting markers at the timeline. This supports underlying developments which are necessary to evolve the API and to fix eventual bugs. Code listing is omitted for presentation purposes.

5.2 Measuring Noise

The noise measurement is a simple procedure that aims to estimate the intensity of background sounds emitted in the environment. This is performed according to audio information captured from the microphone during a given time interval. Once samples are captured within a window for that time period, the module then computes the mean loudness as the noise level perceived in that environment.

5.3 Generating Events: Peak Detection

Peak Detection is an event-generating process. It checks amplitude peaks in a frequency-domain signal. This event makes it possible to take several decisions, since programmers can check a wide range of discrepant amplitudes of any frequencies at the signal. This definition is used in interaction proposed by this work. The interaction is described later. It is noteworthy that, in addition to the Peak Detection, it also exists other types of detection. For example, *Beat Detection* that takes into account the rhythmic perception of beats, among other types of detection.

5.4 Initialization

The API can be initialized in only one line of code by creating an *AudioController* instance, as shown in Listing 8.

The target audio context is created when the API initializes. Alternatively, an existing context can be reused. In addition, the API

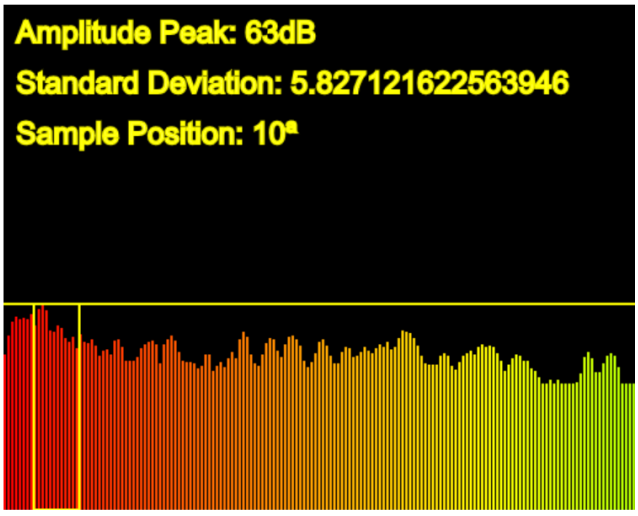

Figure 3: Peak detection procedure.

```
var API = new AudioController();
// ... or ...
var API = new AudioController( audioContext );
```

Listing 8: API initialization requires a new *AudioController* object.

can be configured by changing parameters such as FFT size, minimum and maximum decibels for feature detection. Otherwise, default values 1024, -90 and 0, respectively, are assigned to these parameters.

5.5 Integrating the API into a Game

Integration between the proposed API and games is also a fairly simple process, after it is initialized and the analysis and real-time processing modules gets a vector filled with samples in the frequency domain. With this structure programmers can set events for their applications. In the game prototype presented in this paper, events are generated based on the *Peak Detection* approach and using the average amplitude of each sample as loudness estimate. This event is fired whenever the resulting sound intensity by the player is above a certain threshold.

6 Evaluation

Next, we have a description of a "*endless runner*" game prototype, with some screenshots and their relevant details. Such a game is characterized by rapid and synchronized actions that steer a character undergoing constant movimentation through a scenario. This genre was chosen because both attention and a considerable control level over the mechanism used for interaction are demanded from players.

Finally, it will be presented the results about our implementation and its actual behavior during tests. It is noteworthy that the calibration and game prototype experiments are important not only to validate the research line of this work, but also to foster future works in the area and development of any nature applications like this.

6.1 Developing a Prototype Game

The application, in general terms, consists of two phases. The first comes down to audio calibration, which acquired a noise average in the environment. This process starts at the time that guarantees the permission microphone use the browser being finalized after five

seconds of continuous measurements. In this final step, the state resulting noise in the environment is obtained, defined by parameter *ambientNoise*. After that is provided to the user, depending on the intensity noise average value, performs a recalibration. The following figures illustrate the states of noise.

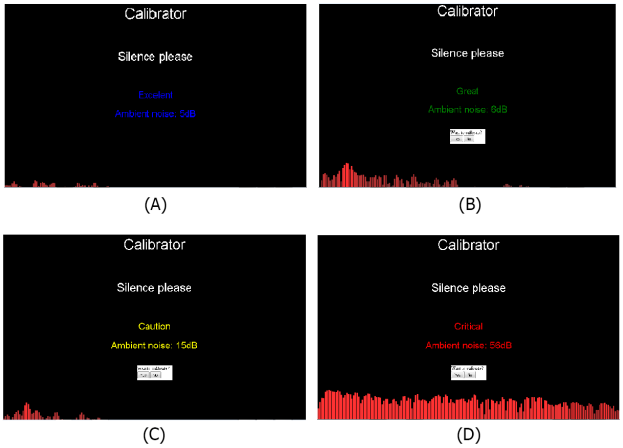

Figure 4: Screenshots of calibration states.

The noise states are defined as: *Excellent*, *Great*, *Caution* and *Critical*, classified according to the value of *ambientNoise* for decibels intervals defined from 0 to 5, 6 to 10, 11 to 15, and 16 ahead, respectively. Recalibration is not required when the state is Excellent, however it is mandatory if the status is Critical. Recalibration is optional in the other states.

After calibration, the application enters its second phase, the game, in the infinite runner style. At this stage, the interaction is done by audio input, the value of parameter *ambientNoise* is considered in the processing of getting the parameter *freq_points* that records the sound intensity associated with noise. This parameter is defined by the average total samples obtained at run time, in the following code snippet.

```
1  if( this.GAME_START && _timer.timeRemaining <= 0 ) {
2
3   this.freq_points = this.audio.getFrequencyAverage();
4   var total_freq = this.freq_points - this.audio.ambientNoise;
5   //(...)
6  }
```

Listing 9: The calibration step conveniently hides most of the game's initialization from users.

If the total *_freq parameter* exceed a threshold of 20dB, an event is triggered causing the character jump.

Moreover, the character has its inventory size items equal to one and a Finite State Machine (FSM) that manages their lifecycle. They are: Normal, Weak and Invincible. The damage that the character takes on collision with the enemy is related to its current state, in Normal mode is discounted a unit of life parameter, in Invincible so there are no discounts and the Weak mode is deducted twice the normal damage.

The states of the character varies according to the item obtained throughout the game where this item is stored in your inventory until the timer parameter equal to 500 (processing cycles) is zero, ie, similar to the concept of cooldown. Only then the inventory slot is released to store another item. The points parameter is increased by one when there is no collision with an enemy. In turn, one hundred

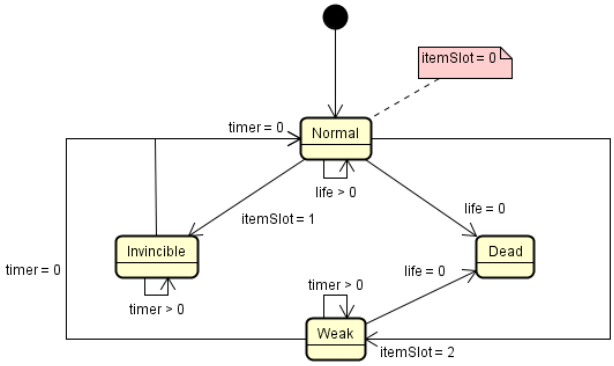

Figure 5: FSM managing the player's avatar status.

points are scored when it acquires a coin. The following figures report the states of character and existing items, respectively.

Figure 6: Sprites currently used as placeholders in the prototype game. All characters are property of their respective owners. These sprites and images are free.

Initial experiments and development adopted a PC with a i7-4500U CPU@1.80Ghz, 16GB RAM running at a 64-bit operating system. Audio was captured using the device's built-in microphone. A capture of the actual game running on a notebook is shown in Figure 7. The prototype game was also executed using a modest Android cell phone, as shown in Figure 8. This device has a 1.3GHz Dual Core processor, 457MB RAM, with Android 4.4 *KitKat*.

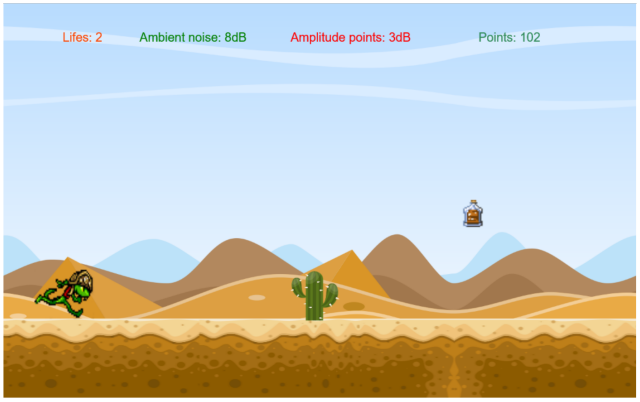

Figure 7: Prototype game executing on the PC.

6.2 Experiment I: Calibration

Calibration is an essential practice in applications that have the audio has the main agent. The calibration is useful for optimization, since the suppression of noise in the signal is partial or total, depending on how it is implemented.

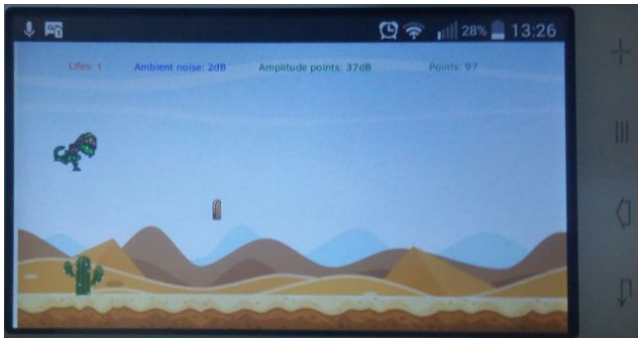

Figure 8: Prototype game running on a modest Android cell phone with a 1.3GHz Dual Core processor and 457MB RAM. Execution did not require any modifications to the game's resources or code except adapting to the screen dimensions.

At first, the prototype did not use calibration and displayed many problems, such as intermittent jumps and, sometimes, no event jumps were detected at all. Calibration significantly increased accuracy when triggering the jump event, and offers feedback to the user about how well the environment is fit for audio processing applications.

6.3 Experiment II: Testing the Game

The game was run on both Desktop and Android platforms in an ordinary room with no soundproofing. In fact, there were two air conditioners functioning in the room plus eventual people walking nearby the testing site.

A group of 5 players was invited to assess how well they could perform the simple actions in the game. Moreover, they were given total freedom to explore interaction. At first, two players were screaming to make the character jump. Henceforth some users started resorting to different alternatives to produce beats, such as snapping fingers and hitting the table.

It must be observed that the game required users to focus on strict timing. After a short adaptation period of less than a minute, players could jump either to avoid obstacles or to collect potions. Despite eventual loud sounds caused by external sources, the interaction was considered eligible by the focal group evaluating the product. In fact, this kind of interaction demands headphones for a better experience because audio feedback is important for a greater immersion.

7 Discussion

The objective of this work, which proposed a study on real-time audio processing and its application for gaming interaction on the web has been achieved. We have also show that such technology fits modern mobile devices such as smart phones and tablets, as game execution did not require anything but an HTML5-compliant web browser.

Due to technical limitations usually displayed by more sophisticated processing methods, our main focus on this paper was applying baseline processing techniques in order to achieve real-time performance. Moreover, if the sampling window is set to wider values, there may be two reasons for sluggish frame rates: (a) the window captures events shifted in time because samples represent a large time period and (b) processing requires more computing time. We observed that a sampling window about 1024 samples wide usually provides both performance and a reliable interaction. Nevertheless, interactive methods based on complex pattern recognition may resort to larger sampling windows in order to cope with words and sentences spoken in different rhythms.

Therefore we exert effort on carefully choosing a game genre that suits our evaluation purposes. We adopted a simple game design requiring fast-paced real-time interaction instead of voice command recognition [11] [16] [13]. Player interaction was considered eligible for real-time interaction on actual games: we corroborate through experimentation that players can easily adapt to this interaction method in no time.

For practical reasons, an audio output should be applicable preferentially when the player is using headsets. Moreover, the player has to switch between the use of voice and other forms to produce amplitude peaks, such as impacts, in order to avoid feeling tired.

8 Conclusion

As proven by practical experimentation we carried out, Web audio API for browsers presented good efficiency even for mobile platforms. Our prototype game developed using HTML5 Canvas shows it is possible to build cross-platform applications and games for the web with processed audio as input.

It is important to observe that more sophisticated mechanisms based on recognition of complex patterns can experience several operating restrictions. Therefore, we opted for a baseline analysis using sound intensity and the manipulation of the respective signal in frequency domain. By using our own API specially designed for this task, peak detection was used to interact in real time in a typical runner game: players managed to use jump obstacles, collect items and also time their jumps to avoid undesired potions.

Other applications for digital games were discussed throughout the development of this work process, which require a multidisciplinary development team and an improvement of the techniques used to date.

There are several fronts of future development made possible from this study. Here are some examples of research we believe stood out for further investigation:

- Proper study about effective use of audio inputs in different game genres and mechanics. This would enable to build guidelines for development teams and their game designer for using audio inputs.
- API extensions and adaptations for others platforms and programming languages;
- Proposition, development and evaluation of other ways to extract useful information for user interaction from audio. This possibly includes the proposal or adaptation of more sophisticated techniques for speech recognition, timbres and specific sounds recorded by the user;
- Computer-aided voice health care;
- Some audio applications, such as soundtracks and special effects, sensitize the spectator at specific times during the application, for example, triggering fear or euphoria. The audio processing in this way could be used as a parameter to direct the production of automatic game content.

9 Acknowledgements

This work is supported by the TEJO-UFC research group and it was partially developed at Bimo-UFC lab. We would also like to thank João Ramos Filho for the artwork and his invaluable help, as well the anonymous reviewers for their considerations.

References

[1] K. Collins. *Game sound: an introduction to the history, theory, and practice of video game music and sound design*. Mit Press, 2008.

[2] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comput.*, 19:297–301, 1965.

[3] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[4] O. Edwards. Simonized: In 1978 a new electronic toy ushered in the era of computer games. *Smithsonian Magazine*, 2006.

[5] S. Fulton and J. Fulton. *HTML5 Canvas*. O'Reilly Media, Incorporated, 2013.

[6] I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, and S. O'Connor, E. an Pfeiffer. Html5 a vocabulary and associated apis for html and xhtml. 2014.

[7] S. Hosokawa and T. Mitsui. *Karaoke around the world: Global technology, local singing*. Routledge, 2005.

[8] D. M. Howard and J. Angus. *Acoustics and psychoacoustics*. Taylor & Francis, 2009.

[9] N. Kirch and N. Zhu. A discourse on the effectiveness of digital filters at removing noise from audio. *The Journal of the Acoustical Society of America*, 139(4):2225–2225, 2016.

[10] A. Kostiainen, I. Oksanen, and D. Hazaël-Massieux. Html media capture. 2014.

[11] E. E. S. Lima, C. Ruby, C. T. Pozzer, and C. N. Silva. Visão computacional e reconhecimento de comandos de voz aplicados na interação com jogos e ambientes 3d. In *Proceedings of SBGames 2010*, volume 1, pages 350–353. SBC, Nov. 2010.

[12] R. G. Lyons. *Understanding digital signal processing*. Pearson Education, 2010.

[13] R. A. M. Antas, G. Souto and R. Valentim. Interface adaptável para jogos digitais: Jogando com a voz. In *Proceedings of SBGames 2015*, volume 1, pages 248–251. SBC, Nov. 2015.

[14] M. K. Mandal. *Multimedia signals and systems*, volume 716. Springer Science & Business Media, 2012.

[15] K. Pohlmann. *Principles of digital audio*. McGraw Hill Professional, 2010.

[16] L. J. S. Silva and C. T. Pozzer. Controle de geração de histórias interativas através de comandos de voz. In *Proceedings of SBGames 2014*, volume 1, pages 1038–1041. SBC, Nov. 2014.

[17] A. Spanias. Advances in speech and audio processing and coding. In *Information, Intelligence, Systems and Applications (IISA), 2015 6th International Conference on*, pages 1–2. IEEE, 2015.

[18] W.-H. Tsai and H.-C. Lee. Automatic evaluation of karaoke singing based on pitch, volume, and rhythm features. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(4):1233–1243, 2012.

[19] V. Välimäki and J. D. Reiss. All about audio equalization: Solutions and frontiers. *Applied Sciences*, 6(5):129, 2016.

[20] V. Vapnik. The nature of statical learning theory, 1995.

[21] S. Webel, M. Olbrich, T. Franke, and J. Keil. Immersive experience of current and ancient reconstructed cultural attractions. *Digital Heritage International Congress (DigitalHeritage)*, 1:395–398, 2013.