CSIT 6910A Independent Project Fall (2013/14)

FINAL REPORT

MULTIMEDIA WEB-APPLICATION DEVELOPMENT OF HTML5 MUSIC GAME

Project Title: Let's Learn Piano

Supervisor: Prof. David Rossiter

Email: rossiter@ust.hk

Student's Name: Lau Yan Yan

Student 's ID:10390612

Table of Contents

1. Introduction

1.1. OVERVIEW

This report is mainly focus on the development of multimedia web application for music game. The approach in development this project was to combine some feature of HTML5 such as canvas and audio; attempt more intricate drawing, making use of dynamic web programming to build a multimedia application combining HTML5 and JavaScript other technologic, including Web Audio API. Before we go down to understand the core of web application, we will have a brief overview and background of HTML5.

1.2. OBJECTIVE

This independent project aims at building online music game application base on HTML5 for user who is interested to learn how to play piano without to paid tutorial free for a piano lesson or cost for a real piano at their spare time by using their computer‘s keyboard and a compatible browser (For example, Chrome and Firefox), each song is consists of step sequence to guide players which musical keys should be played for that particular music note of a piano keyboard such as at C, C D E F G A B C, etc.

2. Background

To achieve our project goals building HTML 5 web application for manipulating audio processing between user interactions with input device, some of the HTML 5 new features are need HTML5 is a markup language used for structuring and presenting content for the World Wide Web and a core technology of the Internet. It is the fifth revision of the HTML standard (created in 1990 and standardized as HTML 4 as of 1997) and, as of December 2012, is a candidate recommendation of the World Wide Web Consortium (W3C). Its core aims have been to improve the language with support for the latest multimedia while keeping it easily readable by humans and consistently understood by computers and devices (web browsers, parsers, etc.). HTML5 is intended to subsume not only HTML 4, but also XHTML 1 and DOM Level 2 HTML.

2.1. NEW FEATURES OF HTML5

In particular, HTML5 adds many new syntactic features. These include the new <video>, <audio> and <canvas> elements, as well as the integration of scalable vector graphics (SVG) content (that replaces the uses of generic <object> tags) and MathML for mathematical formulas. These features are designed to make it easy to include and handle multimedia and graphical content on the web without having to resort to proprietary plugins and APIs.

In this project, we have used one of new feature to drawing graphic game panel to displays the virtual keyboard for play song.

2.2. WEB AUDIO API

The Web Audio API is high level of JavaScript API for processing and synthesizing audio in web applications, This API is built the concept of audio context which is a directed graph of audio nodes that defines how audio stream flows form source audio file to its destination (like computer speaker).

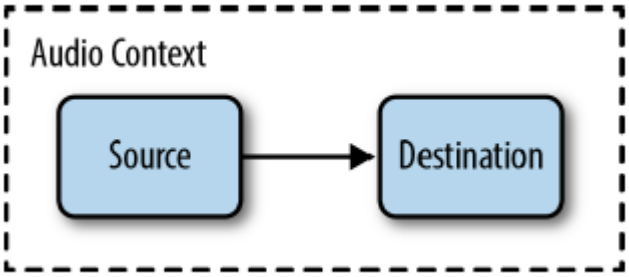

Figure2.1 Audio Stream flows to source

2.3. TYPES OF WEB AUDIO NODES

One of the main uses of audio contexts is to create new audio nodes. Broadly speaking, there are several kinds of audio nodes:

- Source nodes ~such as audio buffers, live audio inputs, <audio> tags, oscillators, and JS processors
- Modification nodes ~ Filters, convolvers, panners, JS processors
- Analysis nodes ~ Analyzers and JS processors
- Destination nodes ~ Audio outputs and offline processing buffers

Sources need not be based on sound files, but can instead be real-time input from a live instrument or microphone, redirection of the audio output from an audio element or entirely synthesized sound. Though the final destination-node is often the speakers, you can also process without sound playback (for example, if you want to do pure visualization) or do offline processing, which results in the audio stream being written to a destination buffer for later use.

2.4. IMPLEMENTATION OF WEB AUDIO API

In the figure2.2 has shown the example of coding for initial Web Audio context and adding two event listener using JavaScript to listen the keyboard event when the player press the key form computer keyboard. It also support cmusic note sound frequency to

```
function initAudio() {  // (1)audio start
    try {
            audioContext = new webkitAudioContext();
    }
    catch(e) {
            alert('Web Audio API is not supported in this browser');
    }
    window.addEventListener('keydown', keyDown, false);
    window.addEventListener('keyup', keyUp, false);
```

Figure2.3 sample of create audio context

In fact the web audio API can generate sound in real-time sound processing, but it has limited in four sound wave forms; saw tooth , sine and triangle wave, those should not enough to generate piano sound for this music game.

2.5. WEB MIDI API

Web MIDI API is Web MIDI API is one of API of Web Audio API designed for web developers to enumerate, manipulate and access MIDI devices - for example interfaces that may provide hardware MIDI ports with other devices plugged in to them and USB devices that support the USB-MIDI specification. Having a Web API for MIDI enables web applications that use existing software and hardware synthesizers, hardware music controllers and light systems and other mechanical apparatus controlled by MIDI. The Web MIDI API does not transmit audio signals: instead, it sends event messages about musical notes, controller signals for parameters such as volume, vibrato and panning, cues and clock signals to set the tempo, and system-specific MIDI communications (e.g. to remotely store synthesizer-specific patch data). This same protocol has become a standard for non-musical uses, such as show control, lighting and special effects control.

The following is sample code for requesting Access to the MIDI System with System Exclusive Support and sending MIDI Messages to an Output Device to produce sound by the specified note number with MIDI messages

```
midi = null;  // global MIDIAccess object

function onMIDISuccess( midiAccess ) {
  console.log( "MIDI ready!" );
  midi = midiAccess;  // store in the global (in real usage, would
probably keep in an object instance)
}
function onMIDIFailure(msg) {
  console.log( "Failed to get MIDI access - " + msg );
}
navigator.requestMIDIAccess().then( onMIDISuccess, onMIDIFailure );

Listing Inputs and Outputs
function listInputsAndOutputs( midiAccess ) {
  for (var input in midiAccess.inputs) {
    console.log( "Input port [type:'" + input.type + "'] id:'" +
input.id +
      "' manufacturer:'" + input.manufacturer + "' name:'" +
input.name +
      "' version:'" + input.version + "'" );
  }
```

Figure2.4 sample for request to access MIDI device

This example sends a middle C note on message immediately on MIDI channel 1 (MIDI channels are 0-indexed, but generally referred to as channels 1-16), and queues a corresponding note off message for 1 second later.

```
function sendMiddleC( midiAccess, indexOfPort ) {
  var noteOnMessage = [0x90, 60, 0x7f];    // note on, middle C, full
velocity
  var output = midiAccess.outputs.entries[indexOfPort];
  output.send( noteOnMessage );  //omitting the timestamp means send
immediately.
  output.send( [0x80, 60, 0x40], window.performance.now() + 1000.0 );
// Inlined array creation- note off, middle C,
  // release velocity = 64, timestamp = now + 1000ms.
```

Figure2.5 example sends a middle C note to output port

From the previous finding of Web MIDI API, it should be the API to support the HTML5 music game to play piano sound.

3. Structure of Game Design

The design of virtual keyboard is two full octaves worth of piano key , both black and white and use some styling to make it look like a real piano keyboard, when user click on the key with mouse , it will play corresponding note.

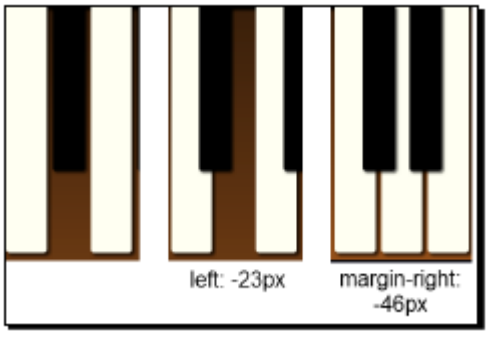

Figure 3.1 layout of the piano keyboard

For playing the note of song at correct time, this game will have two different panel splash panels, which is the starting point of the game, it will display start game button and song name for player to select the song want to play and tempo for selected song.

The game panel contains the piano keyboard and an area that shows the notes to play dropping down from above it. If the user plays the correct note at the correct time, they get points. At the end of the song, the player's score and some statistics are displayed. When the game is done, the application will transition back to the splash panel where the user can select options and play again.

The flowchart has illustrated game transition from one state to another.

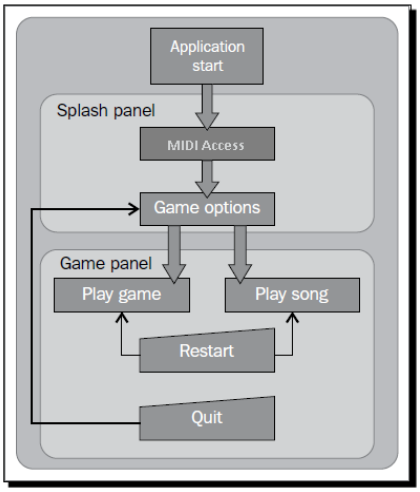

Figure 3.2 Game transition flowchart

3.1. GAME IMPLEMENTATION

3.1.1. Splash Panel

The implementation of this project have split up in two different part, firstly let start on splash panel, and define a new object called SplashPanel in it that will contain all of the code to control the splash panel. The constructor will take one parameter, a reference to audioManager: We define a $div object to hold a reference to the splash panel's root element, and an error variable to set if there was an error loading the audio. Next we define the public show () and hide() methods. These will be called by the main application object to show or hide the panel:

The showOptions() method is called after all audio has been loaded. First we hide the element with the "loading" class, and then fade in the element with the "options" class. This hides the progress section and shows the section that contains the game options.

Our error handler function calls showError(), passing it the audio element that failed: The last thing we need in our splash panel is to hook up event handlers to the buttons. There

are two buttons, Start Game and Play Song. The only difference between them is that the

Play Song button plays the song without scoring, so the user can hear the song and practice:

```
$(".options button", $div).click(function() {
var songName = $("#select-song>option:selected", $div).val();
var rate = Number($("#select-rate>option:selected", $div).val());
var playGame = ($(this).attr("id") == "start-game");
app.startGame(songName, rate, playGame);
});
```

Figure 3.2 JavaScript function for user selection

First we get the options that the user selected, including the song and playback rate. You can find the selected <option> element in jQuery using the: selected selector. We determine which button the user pressed by looking at the button's id attribute. Then we call the startGame() method on the global app object passing in the selected options to start this game in the spash panel.

Figure 3.3 Spash panel of music game

Secondly, this game still need a game panel by adding markup for an area to hold animated note elements, and an area to show the score. These are in addition to our keyboard we created in the previous chapter. Then, we created a JavaScript object to hold all of the code for the game panel, including all of the code we wrote previously for the piano keyboard.

At this point there's not much left in our main application object, music game application. We moved all of the code to load the audio to the SplashPanelobject, and all of the code to make the keyboard work to the GamePanel object. at last, we will creating an audio sequencer object to play the game, we need some way to play songs on the piano by playing back notes in a certain order, at the correct time, and at the correct speed. We will create an object called AudioSequencer that takes an array of musical event objects and turns them into music.

3.1.2. **Audio S**equencer

To implement our audio sequencer we need to define a format for our music events. We will roughly follow the MIDI format, but much more simplified. MIDI is the standard to record and play back music events. Each event contains information about how and when to play notes, or turn notes off.

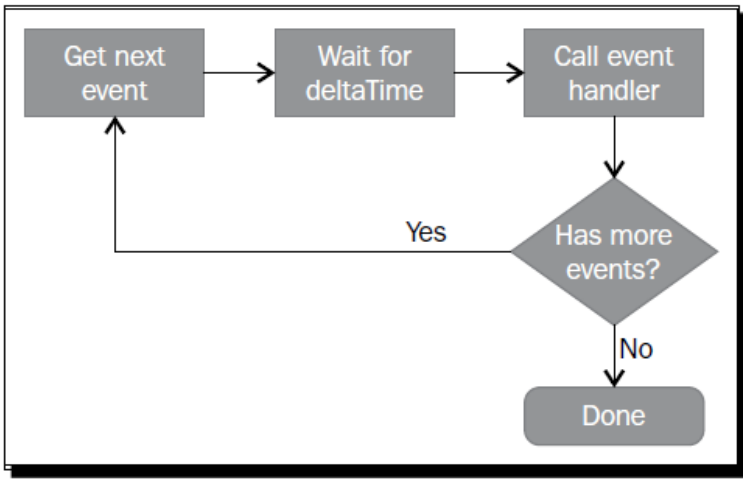

Figure 3.2.2 Flowchart of audio sequencer

Our event object will contain three fields:

•**delta Time:** (Amount of time to wait before executing the event)

•**event:** This is an integer event code that determines, it told that what the event does.

It can be one of the following:

- Turn a note on
- Turn a note off
- Cue point will be at the beginning of a song
- End of track will signal that the song is over

• **note:** This is the note to play.

It contains the octave and note, and matches our audio file names, for example, 3C. The audio sequencer will work by looking at the delta Time field in each event to determine how long to wait before firing the event. The client will pass in an event handler function that will be called when the event is fired. The client will then look at the event data and determine which note to play. This loop continues until there are no more events left. Finally, the music game will shown as bellows

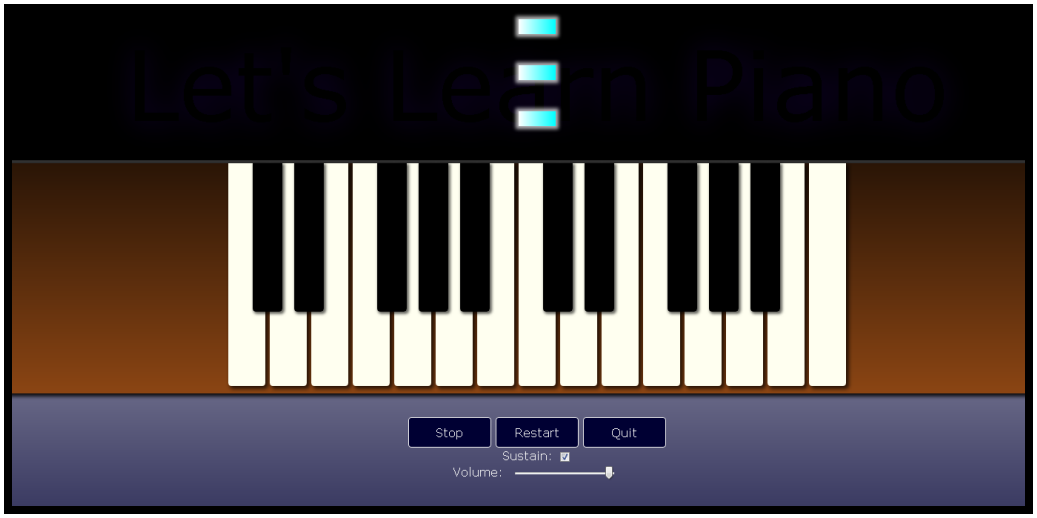

Figure 3.2.3 Layout of Game panel

4. Conclusion

In this project we brief introduces and implemented some of the HTML5 new element and features those are easy to make use in our web development and create a dynamic content of web application by using some HTML tags such as <audio> and <video> , <canvas> are most useful technology to allow us to build rich client and multimedia web application.

Although, HTML5 dynamic programming is repaid the development time on web application, but it still have the limitation of use its provided API <canvas> for example on key press event, control the graphic movement is not a easy things. In fact, the new feature and <audio> and <video> tag can provide web application more dynamic animation.

5. Demonstration

The demonstration is at the URL at below

https://github.com/enfys/s-webmidi

Reference

1. Web Audio Api - B Smus (O'reilly, 2013) (Ecv) Ww.

2. Apress - Pro Html5 Programming, Powerful Apis For Richer Internet Application Development (2013).

3. Oreilly Html5 Canvas 2Nd Edition Apr 2013.

4. Working with MIDI Original Manual: Synkron Revision and Quality Control for Nuendo 3