Don't Quit

Woot! UT fanfic as well as a framework for online story experiences.

All code except for that in the "chapters" directory is published under the GNU Public License 3.0.

Documentation

Hello! I wrote two frameworks for my own use in my story experience. If you are looking for either a super-simple Audio framework or Story displayer in JavaScript, these could work for you. Both are written in plain HTML5 and JavaScript, so you can edit them to your whims. I've tried to make them as general as possible, but ended up making some mistakes, producing somewhat project-specific results. Again, if there's ANYTHING you need to change to make them work for you, hack away!

Story Framework

The framework is for interactive stories, as dynamic pages can be loaded and scripts for individual pages can be created. It is 100% complete for my purposes in the repository, but has many dependencies for now, so it may not be right for your purposes.

Dependencies

All of the following dependencies should be included BEFORE YOU USE THE ELEMENT.

- JQuery
- Polymer
- Materialize
- History.js
- Hammer.js
- Keypress.js
- AngularJS (maybe?)

I will probably package most of these with the framework sometime. Just not now.

Use

Attributes

To use, first add `<link rel="import" href="path/to/story.html">` to the `head` portion of the document. Next, add a `story-box` element to the `body` portion of the document with normal HTML attributes as well as the ones listed below:

- `chapters (Integer)`: **MUST BE DEFINED!!** This tells how many chapters the framework will recognize as existing. It goes by natural numbers; in other words, it begins at ONE, not zero.
- `chapter (Integer)`: the current chapter being displayed. This can be changed dynamically and will load the chapter on change. Default: `1`
- `name (String, optional)`: will change the innerHTML of any element with class `chara` to its value on chapter change or on value change.

Example setup: `<story-box chapters="10"></story-box>`

$Story variable

At any time, you can change ANY attribute, even a plain HTML one, or call a method with `$Story`. For instance, `$Story.chapter=3` will change the chapter to chapter 3.

Methods

These methods are very useful for simple-seeming tasks.

- `_next()`: moves to the next page. However, it is suggested that you use `$(".nxt").click()` instead for a few reasons: The button can be a central place to handle events for turning the page. Both swiping and using arrow keys call on the button's click handler instead so that adding and removing functions to all three events is simpler.
- `_previous()`: moves to the previus page.
- `_chara()`: changes all elements w/ `.chara` to `$Story.name`

Setup

Your project should look similar to this:

```
|-- Project
    |-- index.html
    |-- story.html
    |-- chapter
    |   |-- 1.html
    |   |-- 2.html
    |   |-- 3.html
    |   |-- ... (continue to increment as needed)
    |-- js
    |   |-- hammer.min.js
    |   |-- history.js
    |   |-- jquery.min.js
    |   |-- keypress.js
    |   |-- smartquotes.min.js
    |-- materialize
    |   |-- css
    |   |   |-- materialize.css
    |   |   |-- materialize.min.css
    |   |-- font
    |   |   |-- material-design-icons
    |   |   |-- roboto
    |   |-- js
    |       |-- materialize.js
    |       |-- materialize.min.js
    |-- polymer
    |   |-- polymer-micro.html
    |   |-- polymer-mini.html
    |   |-- polymer.html
    |-- webcomponentsjs
        |-- CustomElements.js
        |-- CustomElements.min.js
        |-- HTMLImports.js
        |-- HTMLImports.min.js
        |-- MutationObserver.js
        |-- MutationObserver.min.js
```

```
        |-- ShadowDOM.js
        |-- ShadowDOM.min.js
        |-- webcomponents-lite.js
        |-- webcomponents-lite.min.js
        |-- webcomponents.js
        |-- webcomponents.min.js
```

Here's a quick boilerplate:

```
<!doctype html>
<html lang="en" ng-app>

<head>
        <title>My Cool Story</title>

        <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=0">

  <!-- Base target of "_blank" is recommended so that users can keep your story tab open -->
  <base target="_blank" />

  <!-- Material Design Icons -->
        <link href="//fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">

  <!-- Materialize CSS -->
        <link rel="stylesheet" href="/materialize/css/materialize.css">

  <!-- Polymer Web Components -->
        <script src="/webcomponentsjs/webcomponents.js"></script>

  <!-- JQuery -->
        <script src="/js/jquery.min.js"></script>

  <!-- HistoryJS -->
        <script src="/js/history.js"></script>

  <!-- Hammer -->
        <script src="/js/hammer.min.js"></script>

  <!-- Keypress -->
        <script src="/js/keypress.js"></script>

  <!-- Story framework -->
        <link href="/story.html" rel="import">


  <style>
                .primary {
                        background-color: #ACFD7C !important;
                        color: rgba(38, 50, 56, .80) !important;
                }
                .accent {
                        background-color: #E2FD00 !important;
                        color: rgba(38, 50, 56, .80) !important;
                }
                story-box {
                        min-height: 100%;
                }
                hr {
                        border: 0;
                        height: 1px;
                        background-image: linear-gradient(to right, #E2FD00, #ACFD7C, #E2FD00);
                }
        </style>
</head>

<body class="blue-grey darken-1">
        <div class="navbar-fixed">

  <!-- A quick header that has a dropdown with chapter selection -->
                <ul id="dropdown1" class="dropdown-content">
                        <li><a target="_self" href="javascript:chapter(1)">One</a>
                        </li>
                        <li><a target="_self" href="javascript:chapter(2)">Two</a>
                        </li>
                        <li><a target="_self" href="javascript:chapter(3)">Three</a>
                        </li>
                </ul>
                <nav>
                        <div class="nav-wrapper primary">
                                <a href="/" target="_self" class="brand-logo primary" id="title"><span class="hide-on-med-and-down">   </span>Title</a>
                                <a class="dropdown-button right primary" id="chapterdd" target="_self" data-activates="dropdown1" href="javascript:void(0)">Chapters<i class="material-icons right">arrow_drop_down</i></a>
                        </div>
                </nav>
        </div>

<!-- story box declaration -->
        <story-box chapters="9" id="story"></story-box>

<!-- Materialize JS -->
        <script src="/materialize/js/materialize.js"></script>

<!-- Now this is interesting: apparently I need Angular in my main project? I don't know why. Better include it just to be safe. -->
        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0-rc.1/angular.min.js"></script>

<!-- Smartquotes -->
        <script src="/js/smartquotes.min.js"></script>
<!-- Main JS file if you'd like -->
        <script src="/index.js"></script>
</body>

</html>
```

Chapter files

Inside of the chapter HTML files, absolutely anything goes that would go in the body of a regular HTML file. I suggest wrapping paragraphs in `<p>` tags, because that's actually what belongs in there. You can also include scripts, CSS, anything! If you want to make a Polymer element, simply create the element, then define it afterward. Example:

```
<dom-module id="my-chapter4-element">
<template>
...
</template>
<script>
Polymer({
  is: "my-chapter4-element",
  properties: {
    myProperty: Boolean
  }
});
</script>
</dom-module>

<!-- THIS is the important part: -->
<my-chapter4-element></my-chapter4-element>
```

Dynamic serving

This is important: until I add an option to turn it off, this framework REQUIRES some backend trickery. It ***REQUIRES*** dynamic routing where every request that includes the fake path `/page/` serves the index.html file. Some example URLs could include:

- `http://delpinothedragon.com/stories/az/page/*` serves `http://delpinothedragon.com/stories/az/index.html`
- `https://example.com/page/*` serves `https://example.com/index.html`

SUPER-DUPER SIMPLE NODE SOLUTION:

```
var express = require('express');
var app = express();

var port = process.env.PORT || 8080;

app.use(express.static('./public'));

app.all("/", function(req, res){
  res.sendFile(__dirname + "/public/index.html");
```

```
});

app.all("/page/*", function(req, res){
  res.sendFile(__dirname + "/public/index.html");
});

app.listen(port);
```

And yes, this is the absolute entirety of my "compiled" `server.js` file (maybe). It is the only file that I will publish *only* in "compiled" form.

Audio

Dependencies

Again, I need to minify all of this stuff together, but for now, here are the things you need:

- Polymer
- Howler.js
- Materialize

Use

Cool, so there are considerably fewer dependencies needed for this framework. The setup is similar in this one. In the head of the document, after including everything else, add `<link rel="import" src="/controls.html" />` and in the body, use an `audio-controls` element. The attributes are:

- `playing`: *Number.* **Don't touch this.** Changes what state the audio is in. `0` = stopped, `1` = playing, `2` = paused. Default = `0`
- `mute`: *Boolean.* If true, it mutes all current and future music (music playing when attribute is changed fades). Default: `false`
- `init`: *JSON Object.* Sounds and music to load when everything is ready. This must be a well-formed JSON object (with DOUBLE quotes arround the attributes) defined in the declaration of the element. The object attributes:
  - `sfx`: *Object.* container for sound-effects related stuff.
    - `sounds`: *Array.* Array of sound names. Same as `sounds` argument in the `loadSFX` method. refer to that for more details.
    - `ft`: *String.* File type of ALL of the sounds defined here. Yes, they all have to be the same file type, like `ogg`, `wav`, or `mp3`.
  - `music`: *Array.* Array of music names to be loaded; same as the `music` argument in the `load` method, except the last value is the file type, for example: `["spooky", "scary", "rainbows", "mp3"]` would load "spooky.mp3," "scary.mp3," and "rainbows.mp3."
- `audio`: *String.* **Don't change this, but do use it to your advantage!** Keeps track of the currently-playing music track.
- `musics`: *Object.* **Don't touch this unless you want to add music that I have not included a way to use yet.** An object that contains music that is loaded and ready to play. If you need to edit this object, it's structure is like this:

```
musics: {
  "Name of Song (may contain spaces)": {
    play: function(){
      // fired when $Aud.play("Name of Song") is called. Must be able to be resumed.
    },
    pause: function(){
      // fires when $Aud.pause() is called and this song is playing.
    },
    stop: function(){
      // fires when $Aud.stop() is called and this song is playing.
    }
  }
}
```

- `sf`: *Object.* **Don't touch this unless you want to add sfx that I have not included a way to use yet.** Same as musics but is used for Sound Effects and the only method ever fired is `play()`;

$Aud

Like `$Story`, `$Aud` is the framework's variable for calling any method or accessing any attribute.

Methods

Some methods:

- `_mute()`: **Only used by the framework**
- `_player()`: Don't bother with this. It's my overly-complex way of dealing with play states.
- `load(song)`: Loads songs for use.
  - `song`: *Array.* Array of strings that are names of .mp3 files (minus the extension) located in the `music` directory. I'll add a `type` argument sometime for other file extensions.
- `loadSFX(sound, type)`: Loads sound effects for use.
  - `sound`: *Array.* Array of strings that are names of sound files (minus the extension) located in the `sfx` directory that have the **SAME FILE EXTENSION**.
  - `type`: *String.* File extension shared by **ALL** files referenced in this call. (Ex: in `$Aud.loadSFX(["Boom", "Crash"], "ogg")`, both "sfx/Boom.ogg" and "sfx/Crash.ogg" are loaded)
- `loadRand(sound, type, number)`: Oooh, toughie to explain. I created this because I needed random voice grunts. Every time this "sound" is played, a random sound out of a set is played.
  - `sound`: *Array.* While you can have more than one String in the Array, you should only really have one. This is because you will most likely have a different number of files for each "sound." The **string**(s) is/are the name of a subdirectory in the `voices` directory where files incrementally named are stored.
  - `type`: *String.* File type of **ALL** the files loaded in this call.
  - `number`: *Number.* The number of files in the named sound subdirectory starting with `1`
- `pause()`: pauses the currently-playing song.
- `sfx(sound)`: plays a sound effect
  - `sound`: *String.* The name of the loaded sound that you want to play. This is the same as one of the strings in the `sound` argument passed when you call either `loadSFX` or `loadRand`
- `stop(song)`: stops playing a song that is currently playing.
  - `song`: *String.* This is the same as one of the strings in the `song` argument passed when you call `load`. **Hint:** to stop the currently-playing song, call `$Aud.stop($Aud.audio)`
- `fade(song, v1, v2, duration, stop)`: fades a song from one volume to another.
  - `song`: *String.* This is the same as one of the strings in the `song` argument passed when you call `load`. **Hint:** to stop the currently-playing song, call `$Aud.stop($Aud.audio)`
  - `v1`: *Number.* The starting volume. Volume must be a floating-point value between 0.0 and 1.0.
  - `v2`: *Number.* The volume to fade to. Volume must be a floating-point value between 0.0 and 1.0.

  - `duration`: *Number.* The amount of time the fade will take in milliseconds.
  - `stop`: *Optional Boolean.* If true, the music will be stopped when the fade is complete.
- `unload(song)`: tries to remove a song's array buffer from memory to clear space. For some reason, on iOS, no method that I can find will clear audio memory when using Howler.
  - `song`: *String.* This is the same as one of the strings in the `song` argument passed when you call `load`. **Hint:** to stop the currently-playing song, call `$Aud.stop($Aud.audio)`
- `xfade(song, duration, cross, fadeUp)`: crossfades two songs: the currently-playing song is silenced, while/then another song fades in.
  - `song`: *String.* This is the same as one of the strings in the `song` argument passed when you call `load`. **This CANNOT be the currently-playing song!**
  - `duration`: *Number.* The length of time in milliseconds it takes for the first song to fade out.
  - `cross`: *Boolean.* If `true`, the second song will fade in while the first song fades out. It will reach `1.0` volume at the same the first one reaches `0.0`. If `false`, the first song will fade out, and when it finishes, the second song will begin its fade.
  - `fadeUp`: *Number.* If `cross` is `false`, this determines the length of time, in milliseconds, that the fade in will take.

Other things you can use

Because I have published this under the GNU GPL, feel free to use and edit any part of this code (this does **NOT** include the story portion) you wish as long as I get credit and you use some kind of copyleft license to publish the remixed code. For your reference, I will include documentation for other useful parts of my code.

Write function

~~I needed a way to print text JRPG style for the "live" part of the story. Therefore, I wrote this handy function to achieve this. It can be found in the index.js file. It currently requires my Audio Framework.

- `write(text, selector, voice, callback, options)`
  - `text`: *String.* A string to write character-by-character into a specified element. There are some in-line commands that translate into HTML elements to that tags can be created instantly.
    - `\\y`: makes the color of the text yellow should you define (`.y{color:yellow}`). It changes into `<span class="y">` when written. At the end of any yellow text, you should include `\\/span;`
    - `\\n`: newline. Changes into `<br>`
    - `\\func:[instruction];`: runs ONE instruction. Anything that can be done before needing the use of a semicolon can be done with one call of this. Unfortunately, for now, no special commands should be used immediately after this one due to the nature in which I programmed it, and should be separated by at least a space. Example of possible command: `"Hello,\\func:console.log('hi'); world!"` If you need to run more than one line at one time, function calls, naturally, are accepted. Example: `"Hello, \\func:hey();world!"`
    - `\\/[tag];`: provides an ending tag of the tag of your choice. You should really only need this for spans for now, but as I add functionality, there may be more of a use.
  - `selector`: *String.* Class of the element you want to write to. To write to this element: `<span class="example"></span>` you would use `"example"` in this argument. Full CSS selectors may come l8r.
  - `voice`: *String.* Name of sound effect loaded into Audio Framework to play every time a non-space character is written.
  - `callback`: *Optional Function.* callback for when the entire string is written into the defined element. Unlike the `\\func:` command, this can have as many lines of code as you'd like.
  - `options`: *Optional Object.* The only attribute that this can have for now is `delay`. It changes the delay to whatever you want, in milliseconds. Don't ask me why I did it like this, just request that I change it if you really need me to. Again, though, you can just change it. It's free software.~~

Yeah, I totally beefed this thing up. Forget all that stuff.

RPG Text Framework

This is a text box modeled after **UNDERTALE**'s dialogue boxes. It allows letter-by-letter printing of dialogue as well as inline HTML, JavaScript, and custom shortcodes. The box is fully customizable by editing the HTML and CSS in `speech.html`.

The element to include is `<dialog-box [attributes]></dialog-box>`

Dependencies

- Polymer
- JQuery
- (If using voices) my audio framework

Attributes

attribute	type	description
`position`	String	For now, the two options here are `"top"` or `"bottom"`. Changing this value will move the entire dialog box to either the top-center of the screen or the bottom-center. I may add other options later. Default: `bottom`.
`face`	Boolean	Whether or not the face sprite can be seen. `true` means that the sprite will take up ⅓ of the box's space and the dialogue will take up ⅔. `false` means that the dialogue will take up the entire box. Default: `false`.
`sprite`	String	Path to sprite if face sprite is being used. Default: `null`.
`visible`	Boolean	Toggles visibility of the entire dialog box. *Note:* running `dialogue.start()` WILL enable visibility.

dialogue

Just like my other libraries, there is a set variable for ease of use. Call `dialogue` for the HTML element and its methods.

Methods

All of these methods return `this` so they can be chained!

Method	Arguments	Description

`write(dialogue, func)`		Sets up the framework with the specified dialogue string. Ex: `dialogue.write("* Howdy!")`
.	`dialogue`	This string is added to the queue be printed in the dialog box letter-by-letter. It can also contain HTML, JavaScript, or custom commands set off by `\\` and ended with `;`. An example string would be: `"* Howdy!\\ * I'm \\c:y;FLOWEY!\\</span>;\\b:;* \\c:y;FLOWEY\\</span> the \\c:y;FLOWER\\</span>!"`
.	`func`	This function will be executed when the string defined at the same time is removed from queue and printed. `func` is optional and can be left alone.
`start(options)`		Removes the oldest dialogue and begins the process of printing it into the dialog box
.	`options`	Object that contains options for how the dialogue will be printed. The currently-available options are: `delay, voice, next,` and `skip.` `delay` is the time (in milliseconds) that separates when each character is printed. The default is `50` `voice` will be the sfx name registered with my audio framework that is played every time a character is printed. The default is `"none"` `next` is the key that can be pressed to continue the dialogue (for example, this key would be `"z"` in Undertale). The default is `"z"` `skip` is the key used to skip through dialogue. The default is `"x"` Some other notes about `options:` The first time `start()` is called, `options` must be defined. However, all attributes are optional and will be filled with their defaults, so an empty object is fine. i.e. `dialogue.start({})` However, after the first call, `options` can be left as `undefined`
`changeSprite(sprite)`		Changes the sprite by simply updating `dialogue.sprite;` however, it returns `this` so the methods can be chained (really useful).
.	`sprite`	String that is the path to the sprite image.
`(enable/disable)Face()`		Simply enables or disables the face sprite. Like `changeSprite,` though, it returns `this`
VERY	IMPORTANT	*(or at least reccommended).* The other methods should not be called in normal circumstances. Many are helpers for me and can be used in custom commands.