Coherent UI javascript libraries

0.8.0.0

Contents

Chapter 1

JavaScript Animation library

1.1 Intro

We've been thinking about how to help developers using Coherent UI create better animations easier and this inspired us to make an animation library with a simple API. We are decided to use a JS-based design instead of a CSS3-based, because JS allows easier control on what is going on. After deep research we chose GSAP animation to be the core of our framework. Before choosing GSAP we experimented with 4 JS libraries: web-animations, web-animations-next, GSAP, jQuery. All libraries have roughly the same features. We compared their performance and stability. GSAP claims to be the fastest JS animation library and from our experiments we can confirm that – at least compared to the other 3 libraries. It is also very well documented and has all animation features imaginable. For full list of documentation and tutorials of GSAP, you can read at https://greensock.com/gsap

Coherent animation library

Coherent animation library is based on the GSAP animation library. We'll also provide direct access to GSAP without trying to hide anything. The features-list of GSAP is very long and we want our users to be able to use anything they might need. And for that reason, we made syntax closer to the way animations are declared in CSS3 and expanded the methods of GSAP to facilitate the making of animation.

1.2 Sample usage

In this example we will show you how to create a very simple animation. The animation will represent a moving div element, which will come from left to the current css definition of it.

1. Include the files

First of all you have to include several scripts files in your HTML page in following the order:

```
<script src="./js/gsap/TweenMax.js"></script>
<script src="./js/gsap/TimelineMax.js"></script>
<script src="./js/coherent_animations.js"></script>
```

Now let's define the css properties of the element.

```
#rect {
position: absolute;
  top: 50px;
  left: 250px;
  width: 200px;
  height: 200px;
  background: #000;
}
```

This will create a black rectangle, placed at an absolute position against the page body tag. Current left position it's 250px and top is 50px. Our idea is to show how easily to animate movement of this rectangle, starting from `left:0`, going to the current `250px` and then from `top:0` move to `top: 50px`.

2. Define the animation

The animation definition is an array of objects:

```
var moveRect = [
  {percent: 0.5, left:0},
  {percent: 1, top: 0}
]
```

Each object acts like a keyframe - the `percent` property corresponds to the time at which this keyframe is to be played and the other properties provided describe the frame itself.

3. Create the animation/timeline

The next step is to create a new `CoherentAnimation` object.

```
var anim = new CoherentAnimation();
```

4. Play the animation

Finally, we tie the animation and the div together and play the animation:

```
anim.caFrom({
  selector: '#rect',
  anims: moveRect,
  seconds: 2
}).play()
```

The `caFrom` method takes an object with the following properties as an argument:

- selector: select our element by id
- anims: the array of keyframes we defined above
- seconds: the duration of the animation in seconds. This property is required if our key frames are percent-based. See below for an alternative.

Let us continue with an in-depth look of the library.

1.3 Animation definition

There are two ways to define animation properties - percent-based and seconds-based.

1.3.1 1. Percent based

```
var moveDown = [
   {percent: 0.5, bottom: 10},
   {percent: 1, bottom: 20}
]
```

This percent is based on the time in seconds **provided when calling `caFrom`** as you already saw above. Must be an array of objects.

1.3.2 2. Seconds based

The option lets you specify the time of the key frame in seconds. You can pass an array with objects or a single object:

1. Array of objects

```
var moveDown = [
  {seconds: 1, bottom: 10},
  {seconds: 2, bottom: 20}
];
```

2. Single object

```
var moveDown = {seconds: 1, bottom: 10};
```

If you use an array of objects, animation will proceed in the order of elements in the array - the rectangle will firstly move 10 pixels down, then another 10 pixels down.

1.4 Timeline object parameters:

A `CoherentAnimation` can be configured with several options:

```
new CoherentAnimation({repeat: 1, onRepeat: repeatHandler, yoyo: true});
```

- repeat : Number - the umber of times that the animation should repeat after its first iteration. If repeat is 1, the animation will play twice (the initial play plus 1 repeat). Use -1 for an endless animation. This value must always be an integer.
- onRepeat : Function - A function that will be called each time the animation repeats.
- yoyo: Boolean - If true, every other repeat cycle will run in the opposite direction so that the tween appears to go back and forth (forward then backward). This has no affect on the `repeat` property though. So if repeat is 2 and yoyo is false, the animation order will be:

`start - 1 - 2 - 3 - 1 - 2 - 3 - 1 - 2 - 3 - end`

But if yoyo is true, it goes like:

`start - 1 - 2 - 3 - 3 - 2 - 1 - 1 - 2 - 3 - end`

You can see full list of parameters on `GSAP/TimelineMax`

1.5 Methods

- **caTo(params)**: Adds a tween animation that starts with the object's current style towards and animates it until the style defined in the animation is achieved.

  params: {Object} obj - this object must have a following parameters:

  – selector: {Object} DOM element object or javascript object.

  – anims: {(Object| Array)} a key frame or an array of key frames.

  – offset(optional): {Number} (default = +=0) — Controls the placement of the tween on the timeline (by default, it's the end of the timeline, like "+=0"). Use a number to indicate an absolute time in terms of seconds (or frames for frames-based timelines), or you can use a string with a "+=" or "-=" prefix to offset the insertion point relative to the **END** of the timeline. For example, "+=2" would place the tween 2 seconds after the end, leaving a 2 second gap. "-=2" would create a 2-second overlap

  - seconds(optional): {Number} if you are using percents in your animation definition, you have to set the duration in seconds. Otherwise, this can be omitted.

- **caFrom()**: Adds a tween animation - that starts with the style defined in the animation and animates it until the object's current style is achieved.

  Parameters: {Object} obj - takes the same parameters like `caTo()` method.

- **caStaggerTo()**: Tweens an array of targets to a common set of destination values, but staggers their start times by a specified amount of time, creating an evenly-spaced sequence.

  Parameters: {Object} obj - this object must have a following parameters:

  - selector: DOM elements('.myClass') or javascript array([element1, element2, element3])
  - anims: {Object} this is your snippet animations object
  - stagger: {Number} - Amount of time in seconds (or frames if the timeline is frames-based) to stagger the start time of each tween.
  - offset(optional): {Number} (default = +=0) — same like offset parameters of caTo() method.
  - seconds(optional): {Number} if you are using a percent in your snippet of animation definition, you have to set duration time in seconds.

- **caStaggerFrom()**: The animation will start from your snippet definition to the reach of the current values of the animated object.

  Parameters: {Object} obj - takes the same parameters like `caStaggerTo()` method.

You can see full list of methods on `GSAP/TimelineMax`

1.6 Advance animations

Menu animation example

In this example every menu button will enter the page from a different position.

```
<nav class="menu" id="menu-start">
  <ul>
    <li>
      <button type="button" id="start-btn" class="btn btn-default">
      Start
      </button>
    </li>
    <li>
      <button type="button" id="options-btn" class="btn btn-default">
      Options
      </button>
    </li>
    <li>
      <button type="button" id="credits-btn" class="btn btn-default">
      Credits
      </button>
    </li>
    <li>
      <button type="button" id="quit-btn" class="btn btn-default">
      Quit
      </button>
    </li>
  </ul>
</nav>
```

Style our menu:

```
.menu {
  position: absolute;
  top: 120px;
  width: 250px;
  left: 0;
  right: 0;
  margin-left: auto;
  margin-right: auto;
}
.menu ul {
  position: relative;
  padding: 10px;
}
.menu ul li {
  position: relative;
  margin: 2px 0;
  height: 45px;
}
.menu ul button {
  position: absolute;
  top: 0;
}
```

Now with this styling rules we set the menu's position in the center of the page and add four buttons to it. Let's create an animation for each button.

```
var moveFromLeft = {seconds: 1, left: -450, opacity: 0, ease: Circ.easeIn},
moveFromRight = {seconds: 1, right: -450, opacity: 0, ease: Circ.easeIn},
moveFromTop = {seconds: 1, top: -400, opacity: 0, ease: Circ.easeIn},
moveFromDown = {seconds: 1, top: 'auto', bottom: -300, opacity: 0, ease: Circ.easeIn};
```

We already saw what most of the code above does. The only exception is the `ease` parameter - easing allows you to change the speed of your animation over time.

You can see full list of easing effects on `GSAP/easing` - *By default easing effect is linear*

Next, create a `new CoherentAnimation` and call `caFrom`. By default `CoherentAnimation` it's always pauses and we need to call `play()` to start it.

```
var coherentMenu = new CoherentAnimation();

coherentMenu.caFrom({
    selector: "#start-btn",
    anims: moveFromTop
  }).caFrom({
    selector: "#options-btn",
    anims: moveFromLeft,
  offset: "-=1"
  }).caFrom({
    selector: "#credits-btn",
    anims: moveFromRight,
    offset: "-=1"
  }).caFrom({
    selector: "#quit-btn",
    anims: moveFromDown,
    offset: "-=1"
}).play();
```

Here's the result:

Create a blink effect by clicking on the buttons

To create a blink effect on our buttons we can use this snippet:

```
// Array with objects definition for animations
var blink = [
  {percent: 0.2, opacity: 0},
  {percent: 0.4, opacity: 1},
  {percent: 0.7, opacity: 0},
  {percent: 1, opacity: 1}
];
```

Now with jQuery we can attach a click event on every button in the menu.

```
$('.menu .btn').on('click', function() {
    var blinkBtn = new CoherentAnimation();
    blinkBtn.caTo({
    selector: $(this),
    anims: blink,
    seconds: 0.3
  }).play();
});
```

1.7 Using with requirejs

To use Coherent Animation with requirejs you have to check the paths in `coherent_animation.js` file.

```
require.config({
  // By default coherent_animations.js is in the directory
  // js/libs/coherent_animation/coherent_animations.js starting
  // from your index.html file. If you change the path you need
  // to change the paths on TimelineMax and TweenMax below.
  paths: {
  TimelineMax: '../../../js/gsap/TimelineMax',
  TweenMax: '../../../js/gsap/TweenMax',
  },
  shim: {
  'TimelineMax': ['TweenMax'],
  }
}
```

Now in `main.js` just include the `coherent_animation.js` file in require.config and use it.

```
require.config({
  paths: {
  CoherentAnimation:'../../../js/coherent_animations'
  },
}

// Correct order to load css files
require(['CoherentAnimation'], function(CoherentAnimation) {
  // your code starts here

});
```

Chapter 2

JavaScript Menu Kit

Coherent Menu Kit helps you create great-looking menus for your game in an efficient and easy way.

2.1 Files to include

To use the Menu kit you have to include the following files in your HTML page:

```
<link rel="stylesheet" type="text/css" href="MenuKit/css/main.css" />
<script type="text/javascript" src="MenuKit/js/menu.js"></script>
```

2.2 How to use

Firstly, you have to declare the buttons of your menu by creating an array of objects. Each object has the following properties:

- **label:** {string} name of the button
- **style:** {string}(optional) this is the style name of the button. The default style name is ’primary’; See all buttons styles in examples of Flat-UI at `Flat-UI`
- **callback:** {function} this is the function to be executed when the button is clicked.

Here’s an example:.

```
var newGameButtons = [
  {label: 'Easy', style: 'success', callback: easy},
  {label: 'Medium', style: 'info', callback: medium},
  {label: 'Hard', style: 'danger', callback: hard},
  {label: 'Back',
    callback: function() {swapMenus(newGameMenu, startMenu);}
  }
];
```

Next, you have to create the menu and pass the buttons to it. The `GameMenu` constructor takes an object with the following properties:

- **buttons:** {array} put your buttons array here
- **originPos(optional):** {object} or {string} by default the menu will be positioned be at the center of its parent. You can pass an object for specific position on menu. For example `{top: 100, left: 100};` will place the menu 100px below and 100px to the right of the **parent’s origin**. Alternatively, pass a one of the predefined positions as a string. Acceptable values are *top-left, top-right, center-top, center-left, center-right, center, center-bottom, bottom-left, bottom-right.*

- **fixedOffset(optional):** {bool} if this value is `false` the menu will use percents for calculating its size and offset. Setting it to true will force the usage of pixels. Defaults to `false`.
- **parent(optional):** {string} or {object} if this parameter is omitted, the the menu's parent element will be the body of the document. You can set your parent element by passing a string selector, like "#content .inner" or a DOM element like ‘document.getElementById(’content’);‘.

Here is an example:

```
var newGameMenu = new GameMenu({
  buttons: newGameButtons,
  originPos: 'top-left',
  fixedOffset: true,
  parent: "#content .inner"
})
```

2.3 Using the Menu Kit with require.js

To use the Menu kit with requirejs also you need to include the require-css plug-in. Check the requirejs demo to see it in action.

You need to modify the provided `menu.js` file and set the correct paths to the require-css plug-in and your css files.

```
require.config({
  // map require-css plugin to load css files
  map: {
    '*': {
      'css': 'components/require-css/css'
    }
  }
});
define('GameMenu', ['css!css/main'], function() {
  return GameMenu;
});
```

You must also follow the correct loading order to load the GameMenu's css files and your custom css file in your main script:

```
require.config({
  paths: {
    GameMenu: 'MenuKit/js/menu'
  }
});

// Correct order to load css files
require(['GameMenu'], function(GameMenu) {
  require(['css!css/main'], function(css) {
    // your code starts here
  });
});
```

Chapter 3

JavaScript Options Kit

The Coherent Options Kit helps you to create an options dialog for your game. The kit provides a rande of controls that can be used - sliders, color pickers, dropdowns, checkboxes and textboxes for numbers. Also, you can have multiple options menus and switch between them. Every options menu.

3.1 Files to include

To use the Options Menu Kit you have to include the following scripts and css files in your HTML page in the order:

3.1.1 CSS files to include

```
<link rel="stylesheet" type="text/css" href="../../components/flatui/bootstrap/css/bootstrap.css" />
<link rel="stylesheet" type="text/css" href="../../components/flatui/css/flat-ui.css" />
<link rel="stylesheet" type="text/css" href="../../components/colorpicker/css/layout.css" />
<link rel="stylesheet" type="text/css" href="../../components/colorpicker/css/colorpicker.css" />
<link rel="stylesheet" type="text/css" href="../../lib/css/main.css" />
```

3.1.2 JavaScript files to include

```
<script src="../../components/jquery/dist/jquery.min.js"></script>
<script src="../../components/flatui/js/jquery-ui-1.10.3.custom.min.js"></script>
<script src="../../components/flatui/js/jquery.placeholder.js"></script>
<script src="../../components/flatui/js/jquery.tagsinput.js"></script>
<script src="../../components/flatui/js/bootstrap.min.js"></script>
<script src="../../components/flatui/js/bootstrap-select.js"></script>
<script src="../../components/flatui/js/bootstrap-switch.js"></script>
<script src="../../components/colorpicker/js/colorpicker.js"></script>
<script src="../../components/handlebars/handlebars.js"></script>
<script src="../../lib/js/options.js"></script>
```

3.2 Usage

In this sample we will create an options menu with all of the available ontrols. In picture below you can see how it looks like.

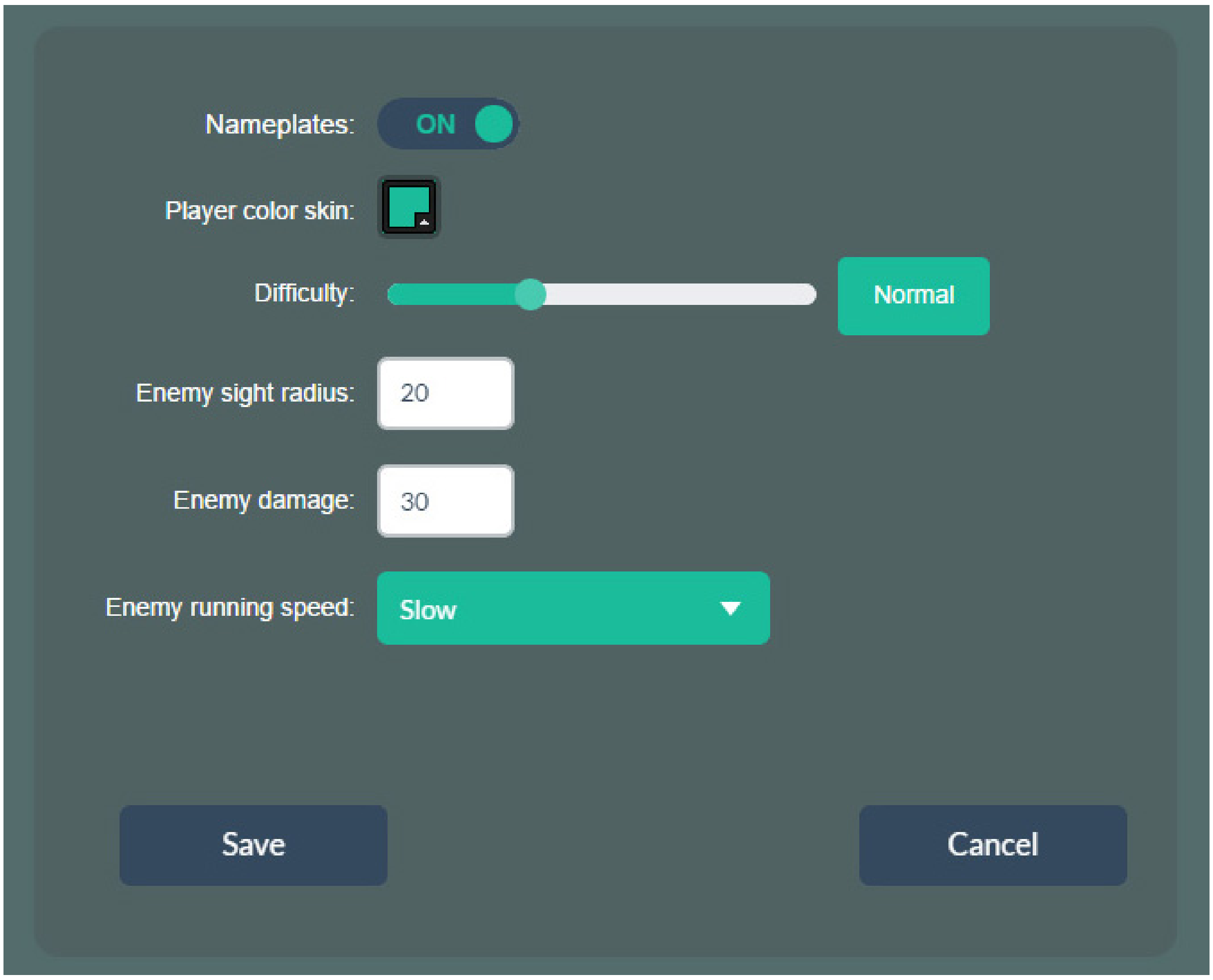
Nameplates: ON
Player color skin:
Difficulty: Normal
Enemy sight radius: 20
Enemy damage: 30
Enemy running speed: Slow

Figure 3.1: Sample options dialog

As you can see, we are displaying all five types of options controls. The first is a `checkbox`, used to switch something to 'on' or 'off'. The second option control is the color picker. The third one is a `slider`, which has four different levels (easy, normal, hard and nightmare) in this case. The `slider` can also contain sub options of type `number`. The `number` control can exist as an independent control or as sub option of a slider. The last type is the `select` (also known as dropdown) control. The code to create all the controlls is the following:

```
var gameOptionsParams = [
  { id: 1, name: 'Nameplates', type: 'checkbox', defaultVal: 'checked'},
  { id: 2, name: 'Player color skin', type: 'colorpickerCUI', color: '#1abc9c'},
  { id: 3, name: 'Difficulty', type: 'slider', values : ['Easy', 'Normal', 'Hard', 'Nightmare'], defaultVal
       : 'Easy',
   subOptions: [
     { name: 'Enemy sight radius', type: 'number', min: 10, max: 100, presets: [{ Easy: 15, Normal: 20,
      Hard: 50, Nightmare:80 }], id:'cui_sub1', defaultVal : 15 },
     { name: 'Enemy damage', type: 'number', min: 10, max: 100, presets: [{ Easy: 20, Normal: 30, Hard: 40,
       Nightmare: 70 }], id: 'cui_sub2', defaultVal : 20}
   ]
  },
  { id: 4, name: 'Enemy running speed', type: 'select', options: ['Slow', 'Normal', 'Fast', 'Super Fast'],
      value: '0'}
];
```

The game options parameters have to be an array of objects. Every object must have the folowing structure: id, name and type. Each type of option has a different value, therefore, the structure of value depends on the type of the option. See below for description of objects with options. When you are done with your options array, you have to create `new GameOptions` object.

```
var gameOptions = new GameOptions({
  gameOptions: gameOptionsParams,
  callbackSave: function() { callbackSave(gameOptions); },
  callbackCancel: callbackCancel,
  menuParentEL: '.options-holder'
})
```

The `GameOptions` take as parameter an object with parameters: gameOptions, callbackSave, callbackCancel and menuParentEL. The `gameOptions` property takes an array with the desired option parameters, the `callbackSave` and `callbackCancel` properties take a function, which will be called when the buttons `save` and `cancel` are clicked. The `menuParentEl` property holds a css selector(string) of the parent element to which the game menu will be appended.

3.3 Defining option controls

To create a control we have to define an object which has the folowing structure: id, name and type. Every single control has a diferent value, therefore, the structure of value depends on the type of the option.

3.3.1 checkbox {object}

- id {Number}: The id of the checkbox option control (unique for all controlls).
- name {String}: The name of the checkbox control.
- type {String}: Must be set to '`checkbox`'.
- defaultVal(optional) {Number}: By default the checkbox is unchecked. You can change it to checked by setting its value to 1.

3.3.2 slider {object}

- id {Number}: The id of the slider option control (unique for all controlls).
- name {String}: The label the slider control.
- type {String}: Must be set to '`slider`'.
- values {Array}: Takes an array parameter with a variable length. Every element in this array will indicate a segment of the slider control. The slider can take an optional property `subOptions`.
- subOptions(optional): The subOptions parameter must be an array with objects. Every subOptions object must have the following properties:
  – id {Number}: Sting name of the subOption item.
  – name {String}: Label of the number control.
  – type {String}: Must be set to '`number`'.
  – min {Number}: minimum value of number.
  – max {Number}: maximum value of number.
  – presets {Array}: takes array with object. These object properties depend on the slider values. For example if you have the slider values - 'values : ['Easy', 'Normal', 'Hard', 'Nightmare']', your object in presets should look like this - { Easy: 20, Normal: 30, Hard: 40, Nightmare: 70 }. The number suboption control is two way bound with the slider control. If you move the slider segment, the number of your subOption will be changed or vice versa.
  – defaultVal {Number}: The default value of the number subOption control.

3.3.3 colorPicker {object}

- id {Number}: The id of the option control (unique for all controlls).
- name: Label of the colorpicker control.
- type: Must be set to ’`colorpickerCUI`’.
- color: Default color of the color picker.

3.3.4 number {object}

- id {Number}: The id of your option control (unique for all controlls).
- name {String}: Label of your option control.
- type {String}: Must be set to (string/literal) ’`number`’.
- min {Number}: minimum value of number.
- max {Number}: maximum value of number.
- defaultVal {Number}: Default value of the number control.

3.3.5 select {object}

- id {Number}: The id of your option control (unique for all controlls).
- name {String}: Label of your option control.
- type {String}: Must be set to ’`select`’.
- options {Array}: For each element in array you have to set a string name which will be displayed of your select list.

3.4 new CoherentAnimation object parameters:

When you are ready with your options definitions for the menu, you have to create `new GameOptions` object, and pass the definitions as arguments to the GameOptions object. `GameOptions` constructor takes four arguments:

```
var myGameMenu = new GameOptions({
  gameOptions: [{ id: 1, name: ’On/Off’, type: ’checkbox’],
  callbackSave: callbackSave,
  callbackCancel: callbackCancel,
  menuParentEL: ’body’
})
```

- gameOptions {Array}: An array with objects for the options controls.
- callbackSave {Function}: A function that will be called when the ’save’ button is clicked.
- callbackCancel {Function}: A function that will be called when the ’cancel’ button is clicked;
- menuParentEL {String}: A css selector string for the parent element to which the options menu will be appended.

3.4.1 Stored user-selected options.

In every menu you can access the user-selected values. You access them from to the `savedOptions` property of the options menu.

```
console.log(myGameMenu.savedOptions);
```

This will return an javaScript object, which holds the generated menu controls and their values.