The Canvas Games Engine

Tutorial

Andrew Parton

# Overview

The Canvas Game Engine uses ECMA 6 (using Babel to transcode in Grunt).

Although all ECMA6 features are available, it only makes use of Classes and Class inheritance (and the occasional String function).

# Features

The system makes use of a hierarchical structure of Entities (Views, Sprites and Text), while also handling server communication.

To create a new game, create a directory using the game name in the Skins folder

## Entity

This is the base class that most visual objects are derived from. It holds various position and size variables – x, y, scaleX, scaleY, width, height, rotation, opacity, etc.

It also defines several useful utility functions which are useful in manipulating derived objects, such as the ability to interpolate any numerical values over time.

Entities can also contain child entities, and all transformations are carried through to the children.

|  |  |
| --- | --- |
| Field | Description |
| X | The X position of the object |
| Y | The Y position of the Object |
| width | The Width of the Object (only valid for Views) |
| Height | The Height of the Object (only valid for Views) |
| rotation | Rotation of the Object (in radians) |
| visible | (Boolean) sets visibility on/off |
| mouseX & mouseY | The localized position of the mouse pointer |
| name | (optional) the name of the Object. Can be used to later find the object if a child of another object |
| Handle | Object defining the handle of the object. By default, this is {x:0.5,y:0.5}, meaning all transformations are at the center of the object. Setting this to, say, {x:0,y:0} would move the transform origin to top-right |

|  |  |
| --- | --- |
| Function | Description |
| Constructor(data={}) | The constructor. **Data** is an optional object which is merged into the class. For example constructor({width:100,height:100}) will set the width and height of the Object |
| animate(anim, time, callback=null, easefunction=”EaseInOut”) | Use animate to interpolate a numeric field from it’s current value to another over time.  For example, this.animate({x:100,y:100},5,function(p), “EaseInOut”) will interpolate x and y from current values to 100,100 over 5 seconds. On completion, *callback* will be called. The movement will use the EaseInOut curve. |
| stop() | Stops any current animations. This is useful in stopping several animations clashing with each other, and can be chained – i.e. this.stop().animate({x:0,y:0},1); |
| onMouseMove(x,y)  onMouseDrag(x,y)  onMouseDown(x,y)  onMouseEnter(x,y)  onMouseLeave(x,y) | Override these to add mouse handling to your derived class. X and y are localized to the Object |
| onKeyDown(k) | Keyboard handler. K is the window event parameter |
| onWindowTouchEnd(x,y) | This is triggered on a touchend. X and y are global coordinates. Only useful for some audio-related IOS functionality. |
| Update(delta) | Override to perform functionality per frame. *Delta* is the amount of time (in seconds) since the last update (usually a very small number). This can be used to make sure animations happen at a constant rate, even though the frame update may be slow on some devices. |
| Draw() | The draw override function. Your drawing will be translated and clipped to the position, width and height of the object. Useful if you want to separate the drawing from the general update functionality, though drawing can be performed in both functions. |
| add(entity) | This adds *entity* to the end of this object’s child list. Returns *entity*. |
| unshift(entity) | His inserts *entity* at the start of the object’s child list. |
| remove(entity) | Removed an entity from the child list |
| disableMouse() | Disables all mouse input |
| enableMouse() | Enables mouse input |

## View (extends Entity)

The View Object represents a rectangular area of screen, which can contain other objects. It can be scaled, rotated, and moved around within it’s parent. All transformations are passed down to its children. It can automatically be cleared each frame, and optionally have a background color. All draw / mouse functions are translated and clipped to it’s area.

It is created by, for example, var window = new View({x:0,y:100,width:500, height:500, backgroundColor:”red”, opacity:0.5, rotation 3.142, clearScreen:true});

This will create a rectangular area at (0,100), of dimensions (500,500). The screen will be cleared to red every frame. It will be semi-transparent, and displayed upside down.

Once created, it must be added to a parent View – either GameManager (top level) or another View.

|  |  |
| --- | --- |
| Field | Description |
| clip | (Boolean) whether graphics are clipped to the window |
| zoom | Ratio to zoom into the window (experimental) |
| zoomX, zoomY | The center of the zoom area (experimental) |
| clearScreen | (Boolean) whether the screen should be cleared each frame |
| backgroundColor | The background color the screen should be cleared to |
| backgroundImage | The filename, sprite ID or URL of the background image. This will be scaled to whatever size and shape of the view. |

## Sprite (extends Entity)

Sprite sources can be from a Sprite Sheet (generated by dropping graphic files into the Sprites/one and Sprites/two directories), or direct from local images. You can also specify a file online by using a fully qualified URL.

They can be moved, flipped, scaled etc.

They are especially useful when you derive new classes from them, and use their update() function to add specialized behavior.

|  |  |
| --- | --- |
| Field | Description |
| flipX, flipY | (Boolean) whether to flip horizontally or vertically |
| Source (path) | Manually sets the image source of the sprite, This can be a sprite ID, the local path of an image, or the fully qualified url of an image (begin with ‘http://’) |

|  |  |
| --- | --- |
| Function | Description |
| constructor (source, data) | *Source* is the image source – whether a sprite id, local path, or url of an image. Data is an optional object which will be merged with the Sprite object |
| draw() | This draws the sprite. If it is overridden, you MUST call this function first (super.draw()) |
| addFrameAnimation(name, animData) | This adds a sprite animation to the sprite. *Name* is the name that you will use to refer to the animation. *animData* is the actual animation definition (see *animDef*) |
| startFrameAnimation(name) | Starts the animation named *name,* which you added in *addFrameAnimation()* |
| stopFrameAnimation() | Stops the current animation |
| hasAnimation(name) | Returns whether this Sprite contains the named animation |
| showFrame(name, frameNum) | Shows frame number *frameNum* of animation *name* |
| hideAnimation() | Hides all animations. Returns to the original sprite image |

## GameManager

The GameManager controls the backend, image manager, State Manager and Canvas. It also manages the game loop

The first thing your App should do is:

**var game = new GameManager({width:w, height:h});**

As soon as this is done, a canvas is created, attached to a user-supplied div with an ID ‘container’, initializes various background processes, and starts the game loop.

## An example game setup

Firstly make a copy of the skins/default director, and rename it to the name of your game.

This directory has all the directories and files you will need to build the game. There is also some example code in js/main.js

You are then free to add any files to any of these directories; they will be picked up by the Grunt file, built and copied to the dist directory.

## Building the Game

Once you have created your game folder, open a terminal in the folder and type **sudo npm install.**

This will install all the modules you need to build the game.

To build the game, and start background watching and building, use the following:

Grunt –skin=*dirname* –spritedirect

This will perform a full build of the game, then begin an auto-building process that will watch your directories, and build as necessary.

The –spritedirect flag is important, as it builds the sprite sheets directly to the dist folder. When optimizing the game, you can omit this flag. The sprites will be built to an intermediate directory (spritestmp). You should run these sprites through TinyPNG, and put the results in the spritesbuilt folder. They will then be picked up by Grunt and placed in the dist folder. It has to be done this way as the TinyPNG plugin for Grunt doesn’t work properly.

If you would like to include Jade modules, put the included files in the views/includes directory and use, for example, **include includes/reality.jade** to include them in your markup. This will avoid double-compilation.

You can add however many scss files in the scss folder; however, each one must be added to the index.jade files as a css link.

To add sprites to the sprite sheets, just drop your individual sprite files into the sprites/one or sprites/two directories. These will be picked up automatically and compiled into a spritesheet.

There are two sprite directories because of limits in the size of PNG files. It makes no difference which directory the files go into – just try to keep the numbers balanced.

The logo directory holds the sprites for the Nektan logo, as they need to be kept separate for speed of loading. There is a loading screen class in other games, which you can copy and use. I’ll add another tutorial for that.

To create the audio sprites, drop audio WAV files into the audiosource directory. You will then need to build these manually (they cannot be built on the server) by running:

Grunt –skin=*dirname* –audio

This only needs to be done when new audio files are added. The compiled audio sprites are created in the audiobuilt directory, and subsequently copied to the dist directory when you do a normal build.

*You may have to install FFMPEG on your Mac for this to work correctly. There are plenty of articles on the InterWebs showing how to do this.*

## Create a server

In your MAMP HTDOCS directory, you need to make a symbolic link to your work directory (actually the dist directory).

Open a terminal in the MAMP/HTDOCS directory. Open another Finder window and navigate to the root of your working directory.

In Terminal, type **ln –s** *and* *drag the dist folder into the terminal window. The full path should be added to the line you just typed. Press ENTER.*

You should now have a symbolic link called *dist* in your MAMP/HTDOCS directory. Rename it to something more helpful – e.g. *games.*

Now, in the browser, type *localhost:8888/games/yourgamename*

And your game should appear.

## Fooling the GameServer

To get the game server to work, you have to fool it into thinking that you are on the mfuse network. To do this, edit the Hosts file (/etc/hosts). You will have to remove write protection.

Map 127.0.0.1 to something that includes the word **mfuse** – e.g. games.mfuse.net

Save the hosts file. You *may* have to restart before this takes effect,

Now in your browser, you can type **server.mfuse.net:8888/games/your*gamename*** and the server will think you’re running from mfuse.

You will also have to remove web security from chrome. To do this, use the **AUTOMATOR** app.

For document type, choose **Application**

Now in the scrolling activity list, select **Run Shell Script**

Clear the right hand box, and type **open -a "Google Chrome.app" --args --disable-web-security**

Save the app to the desktop with a name like **Chrome No Security**.

Make sure you have completely quit Chrome, then run the app you just created. Chrome should now run with CORS disabled. You’ll get a warning dialog when it first starts up. You should now be able to access GameServer with no problems.