Small
Data
Industries

Cooper Hewitt
Technical Analysis of 10,000 Cents

Abstract: the purpose of this report is to establish a clear and canonical record of the technical and functional behaviors of Aaron Koblin's 10,000 Cents, as a preliminary step prior to proposal and execution of a migration strategy. The inspection was conducted in the spring of 2023 by technician Emma Dickson, with guidance and editing from Cass Fino-Radin.

Table of Contents

Summary of Findings

As an initial step in the restoration of Ten Thousand Cents (2008) created by Aaron Koblin and Takashi Kawashima, we were provided with a folder full of additional assets that had not been

analyzed in the previous report prepared by Small Data. Using a MacBook Air with an M1 chip running Ventura 13.0.1 we explored these folders and have detailed my findings in this report.

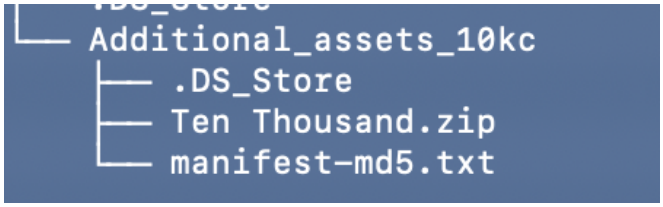
Having access to the .fla files allowed me to directly view the code that was used to parse the data strings being fetched from the designer's site. The code has been annotated and explained here in depth. A strategy for migrating this capability has been laid out along with further avenues of research and questions to be answered by the artist.

Contents of Additional Assets Folder

To get a file tree for the Additional_assets_10kc folder the following commands were run.

```
cd Downloads/Additional_assets_10kct
tree -a
```

This revealed that the asset folder contained another zipped folder named "Ten Thousand" and a manifest-md5.txt file.

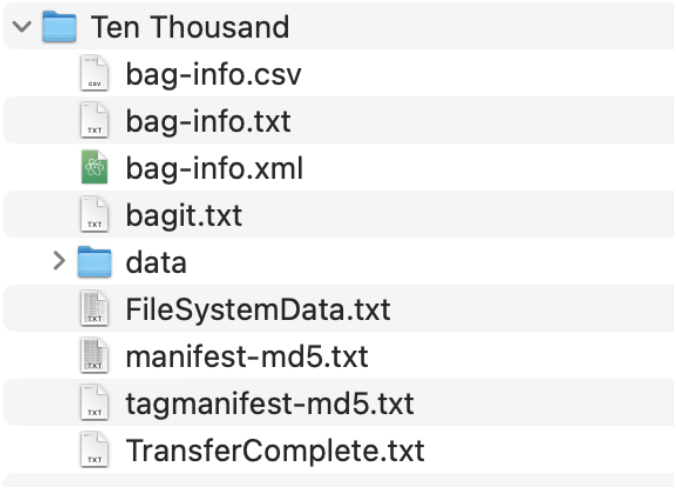


Screenshot of the filetree of the Ten Thousand folder

The "Ten Thousand" folder was then unzipped and the tree command was used to identify its internal structure.

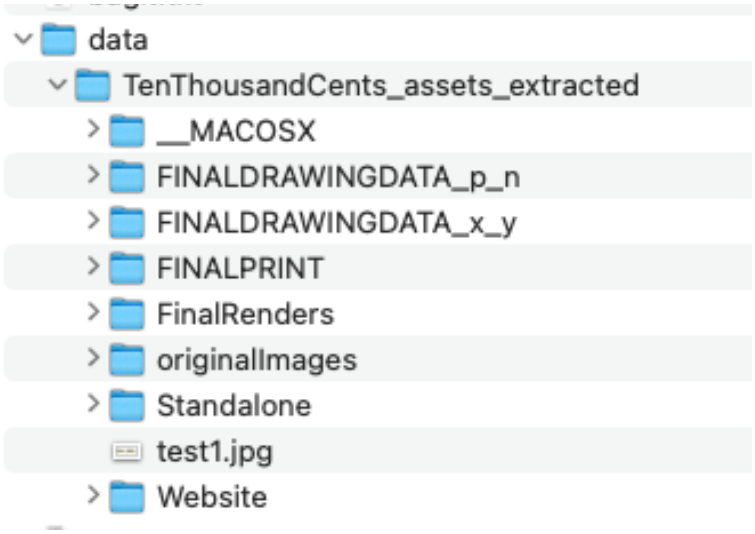
```
unzip Additional_assets_10kc/Ten\ Thousand.zip
cd Ten\ Thousand
tree -a
```

The unzipped tree structure contained 12 directories and 20120 files. To limit confusion we will present high level screenshots and descriptions of the resulting file structure and then go in depth into the contents of relevant folders. The initial level of “Ten Thousand” contains bagit related files such as bag-info.csv and another manifest-md5.txt file and a folder named “data” as seen in the screenshot below.



Screenshot of the filetree of the Ten Thousand folder

Opening the data folder reveals that it contains a single subfolder entitled **`TenThousandCents_assets_extracted`** that contains 8 subfolders, depicted below.



Screenshot of the filetree of the TenThousandCents_assets_extracted folder

Before delving into the in-depth analysis of individual components and particularly relevant files that pertain to the work, we will first provide a high-level overview of each of the subfolders depicted above and briefly explain their contents.

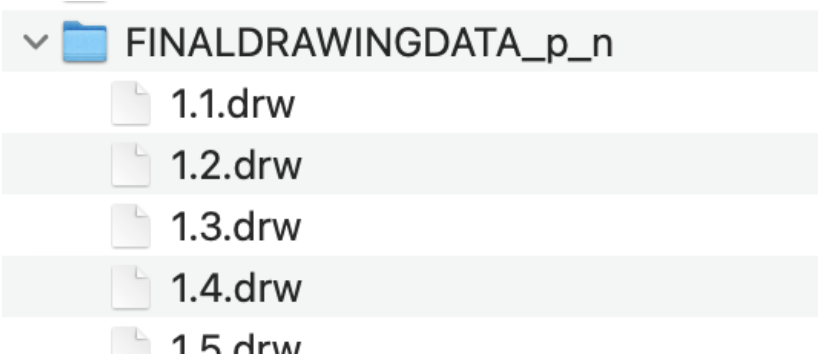
__MACOSX

The first folder __MACOSX contains a single folder entitled FinalRenders which itself is empty. Checking for hidden files within FinalRenders using the following command revealed nothing aside from a .DS_Store file. For the purposes of this report this folder will therefore be ignored.

```
FinalRenders % ls -a
.               ..              ._.DS_Store
```

FINALDRAWINGDATA_p_n

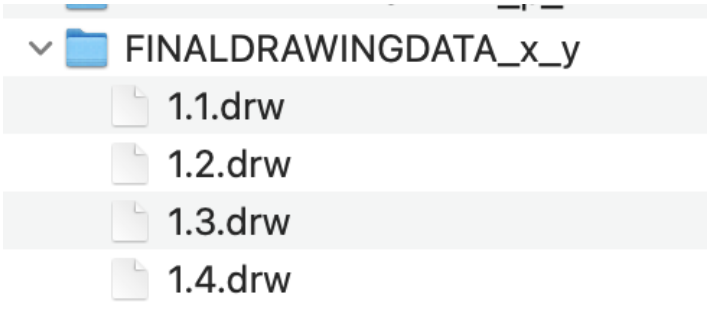
The FINALDRAWINGDATA_p_n folder contains 10,000 files with the naming convention of NUMBER_BETWEEN_1_AND_100.NUMBER_BETWEEN_1_AND_100.drw. This folder will be explored in greater depth later on in this report. A visual representation of a file name can be seen in the screenshot below.



Screenshot of the filetree of the FINALDRAWINGDATA_p_n folder

FINALDRAWINGDATA_x_y

The FINALDRAWINGDATA_x_y folder contains 10,000 files with the naming convention of NUMBER_BETWEEN_1_AND_100.NUMBER_BETWEEN_1_AND_100.drw. This folder will be explored in greater depth later on in this report. A visual representation of a file name can be seen in the screenshot below.



Screenshot of the filetree of the FINALDRAWINGDATA_x_y folder

FINALPRINT

The FINALPRINT folder contains a single pdf BH_TenThousandCent_Print_01_FIN.pdf which has a single image of a still from the flash piece. This image can be seen in the screenshot below.

Screenshot of the filetree of the BH_TenThousandCent_Print_01_FIN.pdf file

FINALRenders

The FinalRenders folder contains the following files.

- Dollar.tga
- Dollaranimation_h264.mov
- Dollaranimation_web3.flv
- tenthousandcentsWebOverview_h264.mov

The Dollar.tga file is an image similar to that within BH_TenThousandCent_Print_01_FIN.pdf. It is an overview of the piece with no pop up window visible. A screenshot of this image file can be seen below.

Screenshot of the filetree of the Dollar.tga file

The Dollaranimation_h264.mov file seems to be a movie of the initial load of all of the sketched images within the piece. It has a length of 45 seconds which is much longer than the loading process takes in the actual flash file.

The Dollaranimation_web3.flv is also a movie of the initial load of all of the sketched images however it only has a duration of 5 seconds.

The tenthousandcentsWebOverview_h264.mov is a promotional video explaining the piece and its motivations.

originalImages

The originalImages folder contains 100 files with the naming convention of benjamin_00.jpg up to benjamin_99.jpg. Since the final project had 10,000 segments of the picture drawn it's unclear how this folder relates precisely to those final 10,000 images.

Standalone

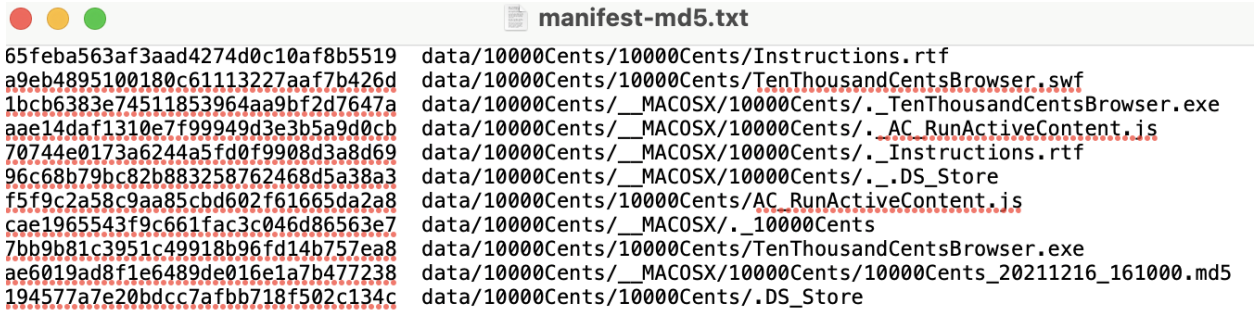
The Standalone folder contains the following files

- TenThousandCentsBrowser.exe
- TenThousandCentsBrowser.fla
- TenThousandCentsBrowser.html
- TenThousandCentsBrowser.swf

Using mac's built in md5 command we were able to get the hash of each of these files and compare them to the assets that were reviewed in our initial condition assessment. This enabled us to understand which elements needed to be analyzed further. The TenThousandCentsBrowser.exe file hash was obtained by running the following md5 command.

```
md5 TenThousandCentsBrowser.exe
MD5 (TenThousandCentsBrowser.exe) = 7bb9b81c3951c49918b96fd14b757ea8
```

The resulting hash **`7bb9b81c3951c49918b96fd14b757ea8`** matches the hash of the TenThousandCentsBrowser.exe file provided for the aforementioned report. Since this application has already been thoroughly analyzed it will be ignored for the purposes of this report. The screenshot below is an image of the manifest-md5.txt file from the original report which shows the matching hash.



Screenshot of the original manifest-md5.txt file.

The TenThousandCentsBrowser.swf file hash was found to be **a9eb4895100180c61113227aaf7b426d** using the built in md5 command.

```
md5 TenThousandCentsBrowser.swf
MD5 (TenThousandCentsBrowser.swf) = a9eb4895100180c61113227aaf7b426d
```

This does not match the hash of the TenThousandCentsBrowser.swf file provided for the original report. Instead it matches the .swf file that was pulled from the designer's site.

Since the TenThousandCentsBrowser.swf file matches the flash file on the designers' site and the differences between the Cooper Hewitt provided swf file and the designer provided swf file was already extensively explored it will be ignored for this report.

The screenshot below was taken from the original report to show the discrepancy between materials found by scraping the designers site and materials provided by Cooper Hewitt. It is provided as proof that these two hashes match.

File Name	Museum Provided Materials	Designer Provided Materials
AC_Run_Active_Content.js	f5f9c2a58c9aa85cbd602f61665da2a8	f5f9c2a58c9aa85cbd602f61665da2a8
TenThousandCentsBrowser.swf	212e8a818c5be81abb7e0df11e68a32b	a9eb4895100180c61113227aaf7b426d

Screenshot of the original report showing the difference between Designer Materials and Cooper Hewitt provided materials.

The TenThousandCentsBrowser.html does not match the index.html file on the designers site that was analyzed in the previous report. It's likely that this is simply an export of Adobe Animator but further analysis might be necessary.

The TenThousandCentsBrowser.fla is an Adobe Animator file. This is the original way in which the flash animation was developed. No .fla files were analyzed in the original report and it will provide extremely valuable insight into how best preserve the life of this piece. This file will be

analyzed extensively in this report, as within it lies much of the behavior and logic that will need to inform our conservation efforts.

Website

The Website folder contains the following files.

- DollaInterface9.fla

This is another Adobe Animator file that we will explore extensively and in comparison to the TenThousandCentsBrowser.fla file.

In depth exploration of critical components of the work

FINALDRAWINGDATA_p_n and FINALDRAWINGDATA_x_y folders

The first step in understanding the relevance of these folders is to remember what was discovered in the original report. That document laid out how the “drawing” mechanism of the piece was done using requests to the designers site using the format http://www.tenthousandcents.com/imagebits/x_coord.y_coord.jpg for an image and http://www.tenthousandcents.com/getDrawing.php?/x=x_coord&y=y_coord for data strings which were decoded to create the drawing effect. It seems likely therefore that the FINALDRAWINGDATA_p_n/ and FINALDRAWINGDATA_x_y/ folders may contain files related to this process.

A quick glance at one of the files within each of these folders, `FINALDRAWINGDATA_p_n/1_1.drw` and `FINALDRAWINGDATA_x_y/1_1.drw` show that they contain similar content. Files are in the same format as those found at `http://www.tenthousandcents.com/getDrawing.php?/x=x_coord&y=y_coord` with only one notable exception: The strings returned from the designer website always start with ‘drawing=’ and end with the character ‘&’ while the individual files in the respective `FINALDRAWINGDATA_p_n/`

and `FINALDRAWINGDATA_x_y/` folders don't do this. The purpose and implications of this difference is unclear at this point; it may simply be to make identifying the stop and end sequences of the drawing command clear.

An analysis that removed these characters and compared their hashes in full to each of the folders might allow us to prove which of these folders contains the files meant to be used in the piece, although Aaron Koblin may be able to provide clarification as well.

The next step in exploring these folders was to find out if they were identical. To do this we used the md5sum command which allows you to generate lists of hashes on the command line for entire folders. We ran these commands and have included the full list as a separate file along with this report.

```
md5sum FINALDRAWINGDATA_p_n/* > p_n_hashes.txt
md5sum FINALDRAWINGDATA_x_y/* > x_y_hashes.txt
```

In total, of the 10,000 files contained in each respective folder, 9,900 files were found to be unique, while 100 were shown to be identical. These extensive differences will require more analysis.

Another interesting and potentially useful aspect to explore would be attempting to view these files in the original program that created them. Some research into the file format revealed that it is associated with a Windows platform CAD software. According to https://fileinfo.com/extension/drw this is the historical context for DRW files.

> ***What is a DRW file?***
> *A DRW file is a drawing format used by Micrografx Designer and Micrografx Windows Draw programs. It contains a technical drawing if it was created by Designer and an artistic drawing if it was created by Windows Draw.*
>
> ***More Information***
> *The DRW format was originally used to save drawings created by Micrografx Designer and Micrografx Windows Draw in the 1990s. Designer was used in engineering, construction, and other related fields while Windows Draw was used in content publishing-related fields.*

> *Micrografx was acquired by Corel in 2001 and Corel renamed Designer as DESIGNER and integrated it into CorelDRAW Technical Suite. Corel also integrated Windows Draw into CorelDRAW Graphics Suite.*
>
> *You can still open DRW files with CorelDRAW software but not all formatting is supported. For example, text and shading in a drawing may be moved or removed when opened in CorelDRAW software.*[1]

Viewing the files in the software used to create them may provide important contextual information for the purpose of these files.

TenThousandCentsBrowser.fla and DollaInterface9.fla files

The first step was to compare the hashes of these two files to verify if they were or were not unique. TenThousandCentsBrowser.fla has a hash of **`a9e38519714117fae1596b0f39c05845`** and DollaInterface9.fla has a hash of **`f640e2a0d96250f47e7ae3193f9b1051`**.

The TenThousandCentsBrowser.fla was located within the Standalone folder with the .exe file and other files related to Cooper Hewitt's collection of the piece. It seems then that this .fla file may have been the one used to create these files that were then given to Cooper Hewitt.

The DollaInterface9.fla file on the other hand was alone with the Website folder. The name of the folder implies that perhaps this .fla file was used to create the flash animation visible on the designers site. The presence of two files might explain why the standalone version of the app had a black background which the designer version didn't as noted in the original report. This is a question that we should follow up with the artist about.

In order to ascertain what meaningful differences may exist between these two fla files, a cursory analysis and comparison of these two fla files was performed. This analysis primarily focused on the code regarding the drawing functionality. In the original report it was noted that the strings being returned to the swf file were being decoded somehow and translated into

[1] https://flylib.com/books/en/4.13.1.428/1/ Accessed April 2023

drawing. At the time of writing that report, absent inspection of the fla code, the exact mechanism used to do this was unclear. The decompiled swf file did not show full code snippets and so it was difficult to determine what the exact functionality was.

Now having the original .fla files used to create these swf animations meant that when opened in Adobe Animator we were able to read and analyze the “Actions” used to turn the strings into drawing. Both the TenThousandCentsBrowser.fla and DollaInterface9.fla file contained two distinct action files, DrawCode and MainCode. In order to compare these files we copied and pasted the contents of DrawCode and MainCode into individual files for both TenThousandCentsBrowser.fla and DollaInterface9.fla and then used the md5 command line command to compare them, beginning with the MainCode action.

```
md5 dollaInterface9MainCode.txt
MD5 (dollaInterface9MainCode.txt) = ebda635ceab943df49b31a7456a47574

md5 tenthousandcentsMainCode.txt
MD5 (tenthousandcentsMainCode.txt) = c9032107c63217b3a624d3b79626a9a5
```

These hashes revealed that the files were different between the two .fla files. To better understand these differences we compared them on a line by line basis using Meld, a Mac based app used to get comprehensive file diffs.

This revealed that the dollaInterface9MainCode.txt file seems to have its content repeated twice. It has the contents of tenthousandcentsMaincode.txt but repeated, which accounts for the differing hashes. This seems most likely to be some sort of copy and paste error in the creation process and is perhaps an indication that this is not the final version of the piece.

Content-wise this file confirms our previous findings that the piece is making calls to the designers website on line 91 which reads

```
popup.imageHolder.loadMovie("http://www.tenthousandcents.com/imagebits/"+theX
+"."+theY+".jpg");
```

And line 100

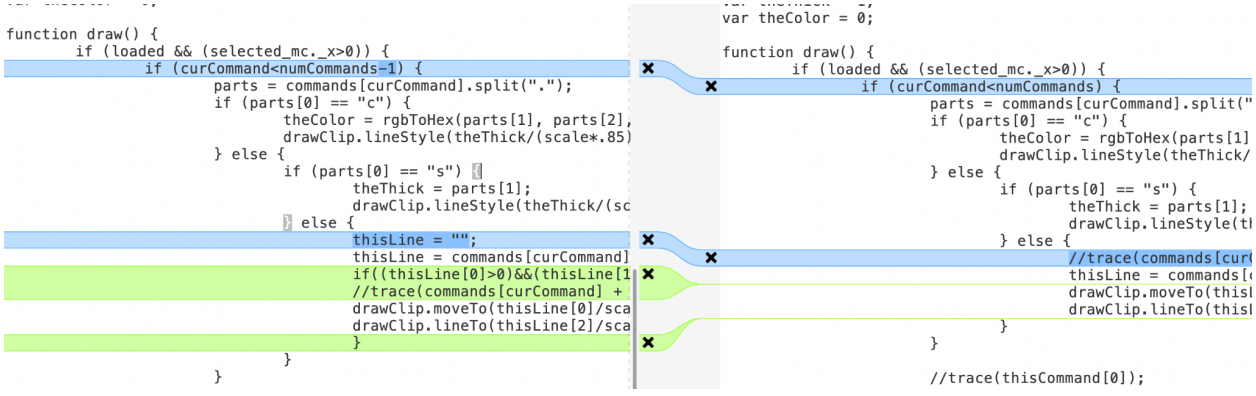
```
sheepLoad.load("http://www.tenthousandcents.com/getDrawing.php?x="+theX+"&y
="+theY);
```

The rest of the code mostly details how mousing around the piece will affect the display, i.e when to call a popup and what dimensions it should assume.

A comparison of the extracted DrawCode files showed that they also had differing hashes.

```
md5 tenthousandcentsDrawCode.txt
MD5 (tenthousandcentsDrawCode.txt) = f0b21524a2bd7ed4bb13ece6b95105ff
md5 dollaInterface9DrawCode.txt
MD5 (dollaInterface9DrawCode.txt) = ae77e02fd5c758273aa9981a50622577
```

A qualitative comparison of these two files again using Meld showed that the tenthousandcentsDrawCode.txt has a key difference in the draw() function. In this version it subtracts 1 from the total number of commands and if the subsequently split string has a character that isn't a 'c' or an 's' it sets the line back to a blank string. This seems to correlate to the idea expressed earlier in this report that perhaps the "drawing=" and '&' that were present in the strings returned from the designer website were added to aid in parsing them into drawings. A visual representation of these differing lines is available in the screenshot below.



Screenshot of the difference between the DrawCode in tenthousancentsBrowser.swf and dollaInterface9.swf

TenThousandCentsBrowser.swf Code

The MainCode Action within both .fla files provides instructions on how user interactions with the piece outside of parsing the drawing strings should function. dollaInterface9.fla's MainCode content is identical to TenthousandcentsBrowser.swf MainCode function, just repeated twice. Because of this we will only review the version within TenthousandcentsBrowser.swf for clarity.

MainCode Line by Line Analysis

The first 20 lines of the MainCode file are visible in the code block below.

```
stop();

var overX = -200;
var overY = -200;
var overNum = 0;
var selNum = 0;
//var delayTimer = 0;
var yOff = dolla._y;
var xOff = dolla._x;
//var scene = "main";
var playSpeed = 5;
//create the listener object
var mouseHandle:Object = new Object();
//create the event
_root.onEnterFrame = function() {
      //if (scene == "main") {
      for (s=0; s<playSpeed; s++) {
            draw();
      }
};
```

The first action stop() is an ActionScript built in action that stops all animation. Meaning that whenever this is called whatever previously was being animated will stop.

```
stop();
```

Next the code sets up some initial variables, overX, oerY, overNum, selNum, yOff, xOff and playSpeed.

```
var overX = -200;
var overY = -200;
var overNum = 0;
var selNum = 0;
//var delayTimer = 0;
var yOff = dolla._y;
var xOff = dolla._x;
//var scene = "main";
var playSpeed = 5;
```

Comments work the same way in ActionScript as in other languages. When the symbols // are placed before a line of code it means that it is not executed, so the line //var delayTimer = 0 and var scene = “main”; will have no effect on the functionality of the piece.

After setting some variables a mouseHandle Object is created by calling new Object(). The Object class is the root of all classes in ActionScript and allows users to add their own values and properties easily.

```
//create the listener object
var mouseHandle:Object = new Object();
```

Finally a function is associated with onEnterFrame. onEnterFrame is an ActionScript specific property.

```
//create the event
_root.onEnterFrame = function() {
      //if (scene == "main") {
      for (s=0; s<playSpeed; s++) {
            draw();
      }
```

The book ActionScript for Flash MX: The Definitive Guide, 2nd Edition written by Colin Moock describes onEnterFrame this way.

> *The onEnterFrame( ) event handler is the callback form (and more modern analogue) of the legacy onClipEvent (enterFrame) event handler. The onEnterFrame( ) handler is executed once for every tick of the frame rate of a movie, such as when the playhead advances or when the Player rerenders the current frame while the playhead is stopped. In either case, the frequency of the onEnterFrame( ) event is directly related to the frame rate achieved by the Player. If the frame rate specified at authoring time under Modify → Document is 20 frames per second, then a tick of the frame rate will theoretically occur every 1/20th of a second.*[2]

In our .fla file the FPS is set to 30. So every 30th of a second this function will be called and run the draw() function which is inside the DrawCode function 5 times.

The next section of code I'll look at is from line 22 to 73 and it assigns a function to the onMouseMove event.

```
mouseHandle.onMouseMove = function(){
      if (distance(selected_mc._x, selected_mc._y, _xmouse, _ymouse)>10) {
            selected_mc._x = -1000;
            selected_mc._y = -1000;

            selected_mc._visible = false;
            popup._visible = false;
            downArrow._visible = false;
            upArrow._visible = false;
      }
      if (mouseInDollar()) {
            Mouse.hide();
            if (_xmouse>437) {
                  if (_ymouse>yOff+160) {
                        image_mc.gotoAndStop(4);
                  } else {
                        image_mc.gotoAndStop(2);
                  }
            } else {
```

[2] https://flylib.com/books/en/4.13.1.428/1/ Accessed April 2023

```
                  if (_ymouse>yOff+160) {
                        image_mc.gotoAndStop(3);
                  } else {
                        image_mc.gotoAndStop(1);
                  }
            }
            overX = Math.floor((_xmouse)/8)*8;
            overY = Math.floor((_ymouse-yOff)/3.4)*3.4;
            image_mc._x = overX;
            image_mc._y = overY+yOff;
            //mc_rollOver._x = overX;
            //  mc_rollOver._y = overY;
            if (overNum != ((overX/7)*80)+((overY)/4)+1) {
                  overNum = ((overX/7)*80)+((overY)/4)+1;
                  //delayTimer = 0;
                  if (overNum>0) {
                        lookAt.text = overNum;

//image_mc.imageHolder.loadMovie("medium/Sheep_"+overNum+".jpg");
                  }
            }
      } else {
            Mouse.show();
            image_mc._x = -200;
            image_mc._y = -200;
            mc_rollOver._x = -200;
            mc_rollOver._y = -200;
            //overNum = -100;
            //if ((selNum>0) && (lookAt.text != selNum)) {
      //          lookAt.text = selNum;

//image_mc.imageHolder.loadMovie("medium/Sheep_"+selNum+".jpg");
      //    }
      }
}
```

The ActionScriptfor Flash MX book describes the onMouseMove event this way.

> *The onMouseMove( ) event lets us detect changes in the mouse pointer's position. Whenever the mouse is in motion, onMouseMove( ) events are issued repeatedly, as fast as the processor can*

> *generate new events. Each time onMouseMove( ) executes, we know the mouse pointer has changed position. We can check that position relative to any movie clip by querying the clip's _xmouse and _ymouse properties.*

This gives us an explanation for what the _xmouse and y_mouse variables we see in the second line of the code snippet are doing. But we still don't have an explanation for selected_mc. Mc is often used in ActionScript code by developers to refer to a specific Movie Clip but this sort of variable would have to manually be set somewhere. In fact it's set in a function that exists a bit after this in the file associated with the mouseHandle.onMouseDown event.

What this means practically is that while the onMouseMove function will execute every time someone moves the mouse it will be missing certain variables it needs unless the user has previously pressed the mouse button.

If the user has previously pressed the mouse button the function gets the distance between where the mouse was when the user pressed down and where it currently is. If that distance is greater than 10 then we set many variables to false or large negative numbers. Essentially if the user clicked down somewhere within the dollar but now their mouse is far away from that point we take no action.

```
if (distance(selected_mc._x, selected_mc._y, _xmouse, _ymouse)>10) {
            selected_mc._x = -1000;
            selected_mc._y = -1000;

            selected_mc._visible = false;
            popup._visible = false;
            downArrow._visible = false;
            upArrow._visible = false;
      }
```

After this if check there is another if statement.

```
if (mouseInDollar()) {
```

This is referencing another function on line 148-154

```
function mouseInDollar() {
      if (inDollar(_xmouse, _ymouse)) {
            return true;
      } else {
            return false;
      }
}
```

Which is a simple wrapper around another function inDollar, passing it the current mouse x and y positions and returning true or false based on those results.

The inDollar function is on lines 140 to 146.

```
function inDollar(x1, y1) {
      if ((y1<dolla._y+dolla._height && y1>dolla._y) &&
((x1<dolla._x+dolla._width-5) && (x1>dolla._x))) {
            return true;
      } else {
            return false;
      }
}
```

This function references a dolla object that isn't declared within the MainCode Action or the DrawCode Action. Presumably this is referring to the dimensions of the larger $100 bill but some more analysis will be needed.

This function checks if the mouse position was within the parameters of the bill and returns true if so and false otherwise.

So the block of code in this next section is executed only if the user's mouse was within the confines of the $100 bill.

```
if (mouseInDollar()) {
            Mouse.hide();
            if (_xmouse>437) {
                  if (_ymouse>yOff+160) {
                        image_mc.gotoAndStop(4);
                  } else {
                        image_mc.gotoAndStop(2);
                  }
            } else {
                  if (_ymouse>yOff+160) {
                        image_mc.gotoAndStop(3);
                  } else {
                        image_mc.gotoAndStop(1);
                  }
            }
            overX = Math.floor((_xmouse)/8)*8;
            overY = Math.floor((_ymouse-yOff)/3.4)*3.4;
            image_mc._x = overX;
            image_mc._y = overY+yOff;
            //mc_rollOver._x = overX;
            //  mc_rollOver._y = overY;
            if (overNum != ((overX/7)*80)+((overY)/4)+1) {
                  overNum = ((overX/7)*80)+((overY)/4)+1;
                  //delayTimer = 0;
                  if (overNum>0) {
                        lookAt.text = overNum;

//image_mc.imageHolder.loadMovie("medium/Sheep_"+overNum+".jpg");
                  }
            }
      }
```

The first section visible here hides the user's mouse and based off of their x and y positions and the yOff variable declared at the beginning of the file assigns the function gotoAndStop with a value between 1 and 4. gotoAndStop( ) is a global function for actionscript that moves the playhead to a given frame and stops the current clip.

```
Mouse.hide();
            if (_xmouse>437) {
                  if (_ymouse>yOff+160) {
                        image_mc.gotoAndStop(4);
                  } else {
                        image_mc.gotoAndStop(2);
                  }
            } else {
                  if (_ymouse>yOff+160) {
                        image_mc.gotoAndStop(3);
                  } else {
                        image_mc.gotoAndStop(1);
                  }
            }
```

Viewing the TenThousandCentsBrowser.swf file in Ruffle (an emulator for running Flash files) it becomes clear what the practical effect of this section is. When users click somewhere on the dollar bill it opens up a pop up window which shows the original image and the draw image rendering side by side. Depending on where in the dollar bill the user clicks this box opens relative to a different point of their mouse. When clicking in the upper left hand of the dollar bill the pop up opens relative to the bottom right of the mouse. This is `image_mc.gotoAndStop(1);` and can be seen in the screenshot below.

Screenshot of the ruffle rendering of TenThousandCentsBrowser.swf file after clicking the top left of the dollar bill.

Whereas, when clicking in the upper right hand of the dollar bill the pop up opens relative to the bottom left of the mouse. This is `image_mc.gotoAndStop(2);` and can be seen in the screenshot below.

Screenshot of the ruffle rendering of TenThousandCentsBrowser.swf file after clicking the top right of the dollar bill.

When clicking in the lower left hand of the dollar bill the pop up opens relative to the top right of the mouse. This is `image_mc.gotoAndStop(3);` and can be seen in the screenshot below.

Screenshot of the ruffle rendering of TenThousandCentsBrowser.swf file after clicking the bottom left of the dollar bill.

When clicking in the lower right hand of the dollar bill the pop up opens relative to the bottom right of the mouse. This is `image_mc.gotoAndStop(4);` and can be seen in the screenshot below.

Screenshot of the ruffle rendering of TenThousandCentsBrowser.swf file after clicking the bottom right of the dollar bill.

This is the next section of code within this same if statement.

```
overX = Math.floor((_xmouse)/8)*8;
            overY = Math.floor((_ymouse-yOff)/3.4)*3.4;
            image_mc._x = overX;
            image_mc._y = overY+yOff;
            //mc_rollOver._x = overX;
            //  mc_rollOver._y = overY;
            if (overNum != ((overX/7)*80)+((overY)/4)+1) {
                  overNum = ((overX/7)*80)+((overY)/4)+1;
                  //delayTimer = 0;
                  if (overNum>0) {
                        lookAt.text = overNum;

//image_mc.imageHolder.loadMovie("medium/Sheep_"+overNum+".jpg");
                  }
            }
```

In this section we set the variables overY, image_mc.x and image_mc.y and then attempt to set the variable lookAt.text if certain position checks are met. The only other references to lookAt within this file have all been commented out just as the other action of loading a movie within this if check has been. Because of this we are ignoring this section of code in our analysis as it seems to have no effect.

The next section of code in the MainCode action is associating a function with the “onMouseDown” or click event.

```
mouseHandle.onMouseDown = function() {

      if (mouseInDollar()) {
            selNum = ((overX/7)*80)+((overY)/4)+1;
            selected_mc._x = overX;
            selected_mc._y = overY+yOff;


//selectedSheep_mc.imageHolder.loadMovie("medium/Sheep_"+selNum+".jpg");
            //loading_mc._alpha = 100;
            drawClip = popup.drawing.createEmptyMovieClip("drawClip", 10);
            drawClip.setMask(popup.drawMask_mc);
            drawClip._xscale = 50;
            drawClip._yscale = 50;
            theX = (overX/8)-4;
            theY = (overY/3.4+1);

popup.imageHolder.loadMovie("http://www.tenthousandcents.com/imagebits/"+th
eX+"."+theY+".jpg");
            selected_mc._visible = true;
            //drawing._x = selected_mc._x;
            //drawing._y = selected_mc._y;
            popUp(selected_mc._x,selected_mc._y);
            //drawMask_mc._x = drawing._x;
            //drawMask_mc._y = drawing._y;
            //trace((overX/8)-4 + " " + (overY/3.4+1));
            //trace(theX + " " + theY);

sheepLoad.load("http://www.tenthousandcents.com/getDrawing.php?x="+theX+"&y
="+theY);
      }
```

```
};
```

The Initial if check uses the mouseInDollar() function which was previously discussed. If a user's mouse is within the dollar parameters we set the values selNum, selected_mc._x and selcted_mc_y.

```
if (mouseInDollar()) {
            selNum = ((overX/7)*80)+((overY)/4)+1;
            selected_mc._x = overX;
            selected_mc._y = overY+yOff;
```

After setting these values this section of code is executed

```
//selectedSheep_mc.imageHolder.loadMovie("medium/Sheep_"+selNum+".jpg");
            //loading_mc._alpha = 100;
            drawClip = popup.drawing.createEmptyMovieClip("drawClip", 10);
            drawClip.setMask(popup.drawMask_mc);
            drawClip._xscale = 50;
            drawClip._yscale = 50;
```

The first lines are commented out so that the first thing actually being executed is `popup.drawing.createEmptyMovieClip("drawClip", 10);`

Popups are commonly used in ActionScript to layer events over each other. ActionScript for FlashMx describes the createEmptyMovieClip functionality as

> *mc*.createEmptyMovieClip(*instanceName*, *depth*)
>
> Arguments
>
> *instanceName*
>
> A string specifying the instance name of the clip to create. The name must adhere to the rules for creating an identifier outlined under Section 15.5. To enable code hinting in the Flash authoring tool, use clip instance names that include the suffix "_mc", as in ball_mc.
>
> *depth*
>
> An integer between -16384 and 1048575 (inclusive), specifying the level on which to place the new clip inside *mc*. Higher depths obscure lower depths, so a depth of 6 is in front of 5, which is in front of 4, and so on. Depths above -1 are reserved for dynamically generated content.

> Therefore, in most cases, you should use a depth between 0 and 1048575 with *createEmptyMovieClip( )*. [3]

So from this we can understand that this line of code is creating an emptyMoveClip that will be rendered on top of the dollar bill with the name of “drawClip”.

We then execute

```
drawClip.setMask(popup.drawMask_mc);
drawClip._xscale = 50;
drawClip._yscale = 50;
```

Which sets a mask over the drawn clip and applies ._xscale and .y_scale variables. Masks are usually used to create unique shapes. The ._xscale and ._yscale variable do as the names imply and scale the original content to the percentage indicated. So in this case we scale the content on both the x and y axis by 50%.

The next section sets the variables, theX and theY then loads a jpg from tenthousandcents.com, makes it visible, creates another pop up of this content using a function defined and then loads the drawing string from tenthousandcents.com

```
theX = (overX/8)-4;
theY = (overY/3.4+1);

popup.imageHolder.loadMovie("http://www.tenthousandcents.com/imagebits/"+th
eX+"."+theY+".jpg");
            selected_mc._visible = true;
            //drawing._x = selected_mc._x;
            //drawing._y = selected_mc._y;
            popUp(selected_mc._x,selected_mc._y);
            //drawMask_mc._x = drawing._x;
            //drawMask_mc._y = drawing._y;
            //trace((overX/8)-4 + " " + (overY/3.4+1));
            //trace(theX + " " + theY);
```

[3] https://flylib.com/books/en/4.13.1.428/1/ Accessed April 2023

```
sheepLoad.load("http://www.tenthousandcents.com/getDrawing.php?x="+theX+"&y
="+theY);
```

The function referenced, popUp, is defined a few lines later on 115-138. This function takes an x and a y value, makes the popup which holds the image and drawing defined previously visible, adjusts the popup's x and y position according to the x and y values

```
function popUp(nX, nY) {
      popup._visible = true;
      if (nY>dolla._y+dolla._height/2) {
            popup._x = nX+5;
            popup._y = nY-popup._height/2-8;
            downArrow._x = nX+5;
            downArrow._y = nY-1;
            downArrow._visible = true;
            upArrow._visible = false;
      } else {
            popup._x = nX+5;
            popup._y = nY+popup._height/2+6+7;
            upArrow._x = nX+5;
            upArrow._y = nY+6;
            downArrow._visible = false;
            upArrow._visible = true;
      }
      if (nX<dolla._x+180) {
            popup._x = dolla._x+180;
      }
      if (nX>dolla._x+dolla._width-180) {
            popup._x = dolla._x+dolla._width-180;
      }
}
```

The last section of the file contains the inDollar, mouseInDollar, and distance function which have been covered in the analysis of other code blocks.

DrawCode Line by Line Analysis

The DrawCode Action within both .fla files clearly lays out how the piece translates the strings returned from the designer site into actual drawing motions. In this section of the report we will break down parts of the tenthousandcentsBrowser.swf DrawCode function draw() line by line and explain what action it is taking. We are ignoring dollaInterface9.fla for now because as discussed previously, our working hypothesis is that tenthousandcentsBrowser.swf is the relevant file and the two DrawCode Actions are very similar.

```
var scale = 1;
loading_mc._alpha = 0;
var loaded = false;
//create the LoadVars object
var sheepLoad:LoadVars = new LoadVars();
//load the content

sheepLoad.onLoad = function() {
      //loading_mc._alpha = 0;
      //--reinitialize to make sure no data is kept from last drawing
      drawClip = popup.drawing.createEmptyMovieClip("drawClip", 10);
      drawClip.setMask(popup.drawMask_mc);
      drawClip._xscale = 50;
      drawClip._yscale = 50;
      commands = "";

      drawClip.lineStyle(0,0x000000);
      loaded = true;
      curCommand = 0;
      //skip1 = false;

      commands = sheepLoad.drawing.split("_");
      numCommands = commands.length;
      theThick = 0;
      theColor = 0;

};

//loadVariables("http://www.thesheepmarket.com/viewDrawingByID.php?command=
get&request=1", _root);
```

```
var curCommand = 0;
var skip1 = false;
var commands = "";
var numCommands = commands.length;
var theThick = 1;
var theColor = 0;

function draw() {
      if (loaded && (selected_mc._x>0)) {
            if (curCommand<numCommands-1) {
                  parts = commands[curCommand].split(".");
                  if (parts[0] == "c") {
                        theColor = rgbToHex(parts[1], parts[2], parts[3]);
                        drawClip.lineStyle(theThick/(scale*.85),theColor);
                  } else {
                        if (parts[0] == "s") {
                              theThick = parts[1];

drawClip.lineStyle(theThick/(scale*.85),theColor);
                        } else {
                              thisLine = "";
                              thisLine = commands[curCommand].split(".");

if((thisLine[0]>0)&&(thisLine[1]>0)&&(thisLine[2]>0)&&(thisLine[3]>0)){
                              //trace(commands[curCommand] + " p");

drawClip.moveTo(thisLine[0]/scale,(thisLine[1])/scale);

drawClip.lineTo(thisLine[2]/scale,(thisLine[3])/scale);
                              }
                        }
                  }

                  //trace(thisCommand[0]);
                  curCommand++;

            } else {
                  //trace("done");
                  loaded = false;
            }
      }
```

```
      // exampleClip.lineTo(_xmouse,_ymouse);
}

function rgbToHex(r, g, b) {
      return (r << 16 | g << 8 | b);
}
```

The first section sets a few variables, the scale, loading_mc.alpha, loaded and a new LoadVars object. This means that when this action is first loaded we are using the object's original scale, the loading_mc is fully transparent, we have nothing in the loaded variable and a fresh LoadVars object.

Adobe's documentation describes LoadVars as *"The LoadVars class lets you send all the variables in an object to a specified URL and lets you load all the variables at a specified URL into an object."*[4]. Which makes sense considering that we will be querying www.tenthousandcents.com for a drawing string.

```
var scale = 1;
loading_mc._alpha = 0;
var loaded = false;
//create the LoadVars object
var sheepLoad:LoadVars = new LoadVars();
//load the content
```

The following section of code takes the LoadVars object that we just created and attaches a function to its onLoad action. Meaning that whenever this object receives data this function is called.

```
sheepLoad.onLoad = function() {
      //loading_mc._alpha = 0;
      //--reinitialize to make sure no data is kept from last drawing
      drawClip = popup.drawing.createEmptyMovieClip("drawClip", 10);
      drawClip.setMask(popup.drawMask_mc);
```

[4] https://flylib.com/books/en/4.13.1.428/1/ Accessed April 2023

```
      drawClip._xscale = 50;
      drawClip._yscale = 50;
      commands = "";

      drawClip.lineStyle(0,0x000000);
      loaded = true;
      curCommand = 0;
      //skip1 = false;

      commands = sheepLoad.drawing.split("_");
      numCommands = commands.length;
      theThick = 0;
      theColor = 0;

};
```

Within the function we create a pop up empty movie clip called “drawClip” with a depth of 10. We then set a mask over the entire clip and scale its contents to 50 percent.

```
drawClip = popup.drawing.createEmptyMovieClip("drawClip", 10);
      drawClip.setMask(popup.drawMask_mc);
      drawClip._xscale = 50;
      drawClip._yscale = 50;
```

Next we initiate an empty string called commands, set the lineStyle to have a thickness of 0 and be black in color. After this we set the loaded variable to true and the curCommand variable to 0.

```
commands = "";

      drawClip.lineStyle(0,0x000000);
      loaded = true;
      curCommand = 0;
```

The last thing we do in this function is to fill the commands variable with the received data string split on the "_" characters. For example the string: "_s.20_c.0.0.0_c.255.255.255" would become ["s.20","c.0.0.0","c.255.255.255"]. We also set the thickness and color variables to 0.

```
commands = sheepLoad.drawing.split("_");
      numCommands = commands.length;
      theThick = 0;
      theColor = 0;
```

Sequentially this block of code is the next thing executed. It exists outside of a function so it would theoretically be loaded initially when the Action script is called. Because all of these variables are set elsewhere or not used or commented out in the case of the reference to sheepmarket this section has no effect. As noted in the previous report it seems that the designer may have reused some code from his previous work sheepmarket.

```
//loadVariables("http://www.thesheepmarket.com/viewDrawingByID.php?command=
get&request=1", _root);
var curCommand = 0;
var skip1 = false;
var commands = "";
var numCommands = commands.length;
var theThick = 1;
var theColor = 0;
```

The draw function has the most important effect in terms of visual output, and the look and feel of the work. It is called within the MainAction as previously explained but declared in this file.

```
function draw() {
      if (loaded && (selected_mc._x>0)) {
            if (curCommand<numCommands-1) {
                  parts = commands[curCommand].split(".");
                  if (parts[0] == "c") {
                        theColor = rgbToHex(parts[1], parts[2], parts[3]);
                        drawClip.lineStyle(theThick/(scale*.85),theColor);
                  } else {
                        if (parts[0] == "s") {
```

```
                              theThick = parts[1];

drawClip.lineStyle(theThick/(scale*.85),theColor);
                        } else {
                              thisLine = "";
                              thisLine = commands[curCommand].split(".");

if((thisLine[0]>0)&&(thisLine[1]>0)&&(thisLine[2]>0)&&(thisLine[3]>0)){
                              //trace(commands[curCommand] + " p");

drawClip.moveTo(thisLine[0]/scale,(thisLine[1])/scale);

drawClip.lineTo(thisLine[2]/scale,(thisLine[3])/scale);
                              }
                        }
                  }

                  //trace(thisCommand[0]);
                  curCommand++;

            } else {
                  //trace("done");
                  loaded = false;
            }
      }
      // exampleClip.lineTo(_xmouse,_ymouse);
}
```

The first action within the function is to check that we have commands that need to be analyzed. This check asks if we have received data, have a pop up window, and have a curCommand value that is less than the total length of received Command.

```
if (loaded && (selected_mc._x>0)) {
            if (curCommand<numCommands-1) {
```

If so we proceed to break a string into a list of individual parts wherever there is a period.

```
parts = commands[curCommand].split(".");
```

So something like this string _c.255.255.255 would become [_c, 0, 0, 0]

```
if (parts[0] == "c") {
                        theColor = rgbToHex(parts[1], parts[2], parts[3]);
                        drawClip.lineStyle(theThick/(scale*.85),theColor);
                    }
```

Now we check the first position of the list and if it is a ‘c’ then we use the rest of the string to create a color using the rgbToHex function. Then we set the line style using the resulting color. By default the variable theThick is set to 1.

If the string at the beginning had been an ‘s’ we would use the rest of the string to specify the thickness of the line we are creating.

```
else {
                        if (parts[0] == "s") {
                              theThick = parts[1];

drawClip.lineStyle(theThick/(scale*.85),theColor);
                        }
```

And if neither a ‘c’ or an ‘s’ were present we would use the rest of the string to specify the starting and ending point of a line, draw the line and reset our string to be analyzed to 0.

```
else {
                              thisLine = "";
                              thisLine = commands[curCommand].split(".");

if((thisLine[0]>0)&&(thisLine[1]>0)&&(thisLine[2]>0)&&(thisLine[3]>0)){
                              //trace(commands[curCommand] + " p");

drawClip.moveTo(thisLine[0]/scale,(thisLine[1])/scale);

drawClip.lineTo(thisLine[2]/scale,(thisLine[3])/scale);
                              }
                        }
```

The rgbToHex function is declared at the end of the file. As the name suggests it converts the received rgb value into a hex value.

```
function rgbToHex(r, g, b) {
      return (r << 16 | g << 8 | b);
}
```

Plain language summarization of code

Our inspection of the work's ActionScript source code, as well as our previous condition assessment of the compiled SWF has given us a complete understanding of the work's functional behaviors and properties.

Led by the viewer's interaction, the work watches for mouse position and mouse click events. When a click occurs, the work sends out a request to the artist's website. The request returns an image (depicting the portion of the $100 bill that was clicked), as well as an encoded string of instructions on how to draw the illustrated version of this image (including color, thickness, and the coordinates of each individual line). The code also reveals to us the decision-making behavior around the relative position that this image and drawing are placed on-screen.

Ultimately this comprises the full functionality of the work as it pertains to perceived behavior.

Proposed Conservation Strategy

Now that we have code from the .fla files that clearly connects the received string data with specific drawing actions it will be simple to convert these actions to pure HTML and JS. All of the functionality we have observed through direct observation, and source code inspection can be easily replicated through simple HTML, JavaScript, and CSS. The code within the fla files seems focused around parsing colors and creating lines, no shapes or other more complex objects. The easiest way to recreate and manage these drawing actions would be to write JavaScript that takes these same strings and creates lines on an svg canvas.

An SVG canvas gives the user more direct control over the appearance of objects within the canvas as opposed to a normal canvas tag. This will allow for the implementation of the piece to more easily address tricky granular visual questions about how much of the original “look” and “feel” of a flash animation should be preserved.

In the course of our conservation treatment, we could also take measures to eliminate external dependencies currently present in the work, by for instance storing data strings as JSON files hosted alongside the HTML and Javascript. As well, we will eliminate the dependency on the artist’s website as an external image server. Both these images and data strings were systematically retrieved and stored for later use during our previous condition assessment.

Outstanding Questions

- A comparison between the FINALDRAWINGDATA_p_n, FINALDRAWINGDATA_x_y folders with the files from the designer site with the beginning and ending strings removed.

Outstanding Questions for the Artist

- In the original report it was noted that the .swf animation hosted on your website does not have a black background while the standalone .swf animation does. Why was this change made and when?
- What were the images within the originalImage folder used for?
- What were the files within the FINALDRAWINGDATA_p_n folder used for?
- Do you remember what the difference between TenThousandCentsBrowser.fla and DollaInterface9.fla was?