# MY GALLERY INTERACTIVE

## AN INTERACTIVE MULTI-TOUCH PHOTOGRAPHY DISPLAY

A COLLABORATIVE PROJECT BETWEEN NICT AND MOPA

BY WESLEY HSU
HOST SITE NICT KEIHANNA LABS
FINAL PAPER

## Introduction

This project is somewhat of a maverick relative to the usual PRIME project, because it emerged from a very fortunate turn of events where certain connections and events fell into the right place at the right time.  Dr. Jason Haga of the Bioengineering Department formerly mentored PRIME alumni, Kevin Nguyen, in the summer of 2010, whose project transformed into building a Frustrated Total Internal Reflection (FTIR) multi-touch table, a multi-touch table made from highly affordable materials using relatively simple technology.  The table Kevin built was housed at the National Institute of Information and Communications Technology (NICT) Keihanna labs following his program.  Following the PRIME project, through Dr. Haga, the Deputy Director at the Museum of Photographic Arts in San Diego (MOPA), Mrs. Vivian Haga, took interest in the technology as a medium for presenting their photography collection at Tokyo Photo 2011.  As a relatively new photography exhibition, MOPA wanted to present its photography collection in an innovative way.  Furthermore, Ideum, a leading company in research and development of innovative museum displays, opened up its Adobe Flash Platform based *GestureWorks* framework as an open-source framework in the form of *Open Exhibits* in November of 2010.  Obtaining a download simply required registration through an approved educational or research institution.  Lance Castillo, another applicant for PRIME and a Computer Science major, was also slated to work on the same project, focusing on the programming.  Through the meetings held during the spring quarter with the MOPA staff, Dr. Haga, Lance, and myself, we came up with mock-ups of an interface that would allow the user to go through a collection of 50 hand-selected photos representing MOPA's collection.  They could pick up to ten photos and add them to their "Personal Collection" (later "My Gallery"), title their collection, then submit them to be viewed by others ("Personal Collection" and "My Gallery" will be used interchangeably throughout this paper, depending on the context).  The aim was to offer the user a sample of the gallery curation process.  Through these turn of events, a project was born that would revolve around user-interface design, museum interfaces, and multi-touch technology—a divergence from the primarily bioengineering-based projects in PRIME.

Indeed, this project is not scientific in the conventional 'scientific method' sense. That is not to say there is no science involved, quite the opposite. However, there are no hypothesis, no materials and methods of procedure, no experiments, and no research findings. This is a strictly interface design and programming project, where writing code, debugging, drawing wireframe layouts, pixel-perfect positioning, designing buttons, and running user tests are the way of life. As such, I would like to take a more personalized approach to this paper's structure.

Instead, I will focus on my thought process through the many stages and components of the development process, primarily detailing the parts that I have familiarity with. I will note the difficulties and roadblocks that I encountered along the way, what worked, what did not, and eventually what I ended up doing to solve the problem. I will also focus on some general user-interface design principles that I learned through the development process. Finally, at the risk of sounding cliché, I will wrap up with my growth as an individual through this project—probably the most important part of the experience at a personal level.

## Learning the Fundamentals

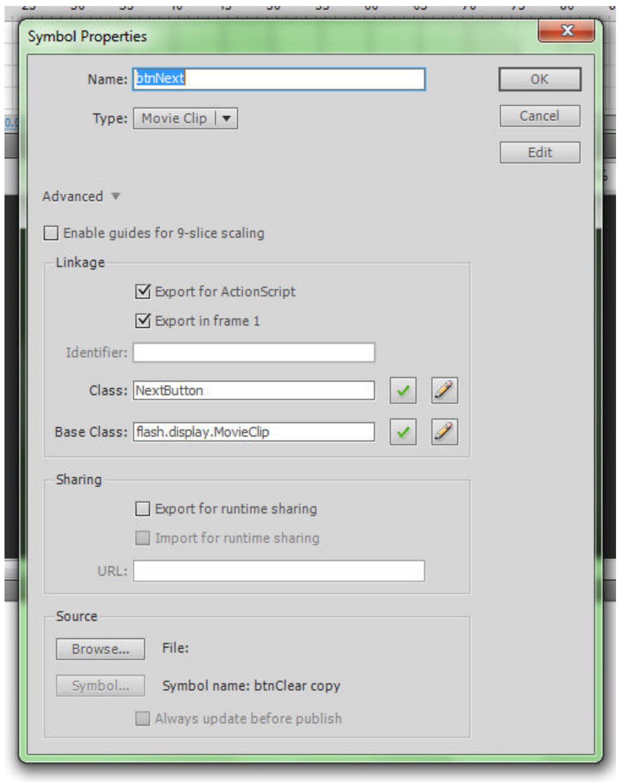
I will be perfectly honest: this project was intimidating. I had left the Computer Science B.S. major, swearing never to write for-loops and if-else if-else statements ever again, unless it was absolutely necessary. It had also been over a year since I had worked with the Flash Platform. When I had last worked with the Flash Platform, it was strictly in Adobe Flash deploying in a web environment (i.e. not Adobe AIR), and using it primarily as an animation and design tool. This project was using Adobe Flash and ActionScript 3.0 (AS3) as an interface development tool. Needless to say, my programming skills were rusty to say the least, and I needed a comprehensive reorientation with the Flash Platform. On top of that, Lance had already done a large bulk of work prior to arriving in Japan, and was already knee deep in AS3 code. I had plenty to learn. The pressure was on.

### Crash Course through Flash

My first order of business upon arriving in Japan was to relearn (perhaps 'learn' is more appropriate) the basics. I purchased a copy of *Real-World Flash Game Development: How to Follow the*

*Best Practices AND Keep Your Sanity* by Christopher Griffith and *Foundation Game Design with Flash* by Rex Spuy prior to leaving for Japan. I also bookmarked a fantastic online tutorial, *AS3 Flash Games for Beginners* from *asgamer.com*. I chose to focus on game development on the Flash Platform because video games demand a high level of interactivity and animation. It is the perfect platform for learning how to use Flash as an interface development tool. My plan for the first week was to run through this crash course, and browse through the books for relevant topics.

It was during this time that I learned about one of the *most* useful and often used concepts throughout this project: Library Linkages. In the Adobe Flash interface, there is a "Library" window that contains symbols. These symbols are generally visual elements that can be drawn and designed by the user, usually of the type Movie Clip. By right-clicking a symbol and going into "Properties…", you are able to access the Symbol Properties menu, which gives access to Linkage options.



Name: btnNext

Type: Movie Clip

☐ Enable guides for 9-slice scaling

Linkage

☑ Export for ActionScript

☑ Export in frame 1

Identifier:

Class: NextButton

Base Class: flash.display.MovieClip

Sharing

☐ Export for runtime sharing

☐ Import for runtime sharing

URL:

Source

File:

Symbol name: btnClear copy

☐ Always update before publish

Figure 1. Symbol Properties window

By checking the "Export for ActionScript" box under Linkages, this really opens up the possibilities of Flash as an interface tool. Above, in the "Class" field, I have entered "Next Button". With this particular symbol exported for ActionScript (a next button graphic), the symbol can be dynamically added to a class by creating it as a new instance of the class. As an example, for code junkies:

```
var nextbutton:NextButton = new NextButton();
```

The 'nextbutton' variable literally becomes a graphical element on the stage (once addChild(nextbutton) is called, that is), whose visual properties can be manipulated around (e.g. x-coordinate, y-coordinate, alpha levels, etc). Any symbol that is exported for ActionScript can be instantiated by code. So if I need multiple next buttons, I could simply instatiate another one. Furthermore, this symbol can become even more customizable by writing an AS3 class of the same name (what I end up doing for the Personal Collection class, more on that later). In the above example, I could write a "NextButton" class of the same name, giving it any properties and functions necessary, even allowing it to inherit whatever class desired. In layman's terms (for those who are not programmers), this allows for the developer and designer to draw any kind of interface component that they want as a Library symbol, and give it any kind of properties necessary, making the design of the interface highly customizable.

This turned out to be a key concept to having the flexibility of designing the interface any way that we wanted.

## Touch Containers

From there, the next step was to understand the *Open Exhibits* framework, the foundation on which the multi-touch functionality of the application would be running on. I began with experimentation, taking apart the tutorials provided on the main page and altering them to fit the needs of our project. The Open Exhibits framework is structured such that any object that needs to be touchable needs to be placed into a TouchObjectContainer (hereon referred to as Touch Containers). That is, a container that has touch capabilities, which can hold a graphic. The graphic can be a button, text, an image, anything visual.

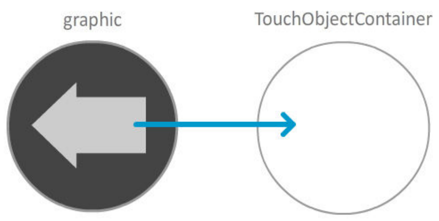
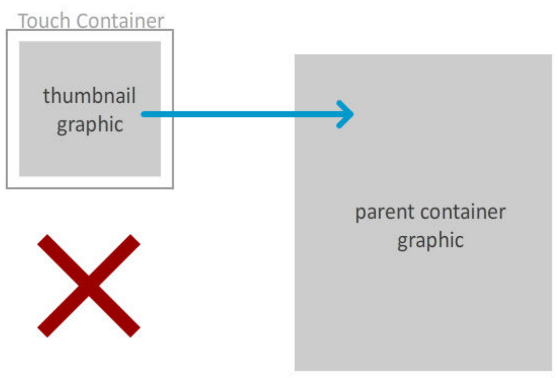


Figure 2. Placing graphic (back button) into a Touch Container

Once the graphic is placed into the Touch Container, the container itself has properties which can be edited to give it touch functionality, such as sensing clicks, drags, or flicks. Given that the user interface would include a Personal Collection, which would contain a bunch of interactive touchable thumbnails, it became obvious to me that we would need to work with nested containers—containers within containers. This is because during experimentation, I learned that Touch Containers need to be placed within *another* Touch Container in order for them to be touchable. That is, if a thumbnail is placed into a larger container, both the thumbnail and the larger container must be placed into their respective Touch Containers to be touchable. That is quite a mouthful, so here is a diagram to illustrate the concept:



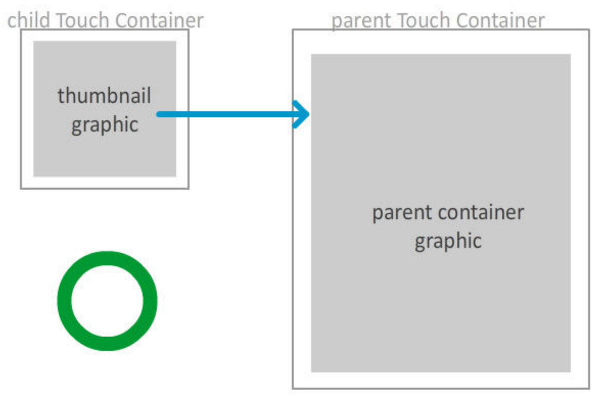


Figure 3. Placing container into a container.

Figure 4. Placing container into a graphic symbol

In Figure 3, placing a thumbnail within a graphic symbol would work with standard AS3 interactivity, such as mouse clicks. However, with Open Exhibits' touch capability, in order for the child container to be touchable within a parent, the parent needs to be a Touch Container as well. The thumbnail would become untouchable in Figure 3. When adding the child container, it must be added to the parent container, not the parent graphic.

I also discovered through heavy experimentation the difference between the so-called *target* and *currentTarget*, a crucial concept for nested containers. At this stage, I had almost daily exchanges with an Ideum representative via the Open Exhibits forum. He played a key role in helping me understand the *target* vs. *currentTarget* difference. Using the above example, a touchable parent container contains touchable thumbnails. Say the thumbnail needs to be draggable from within the parent. The drag event listener (more on event-driven programming later) can target either the *target* or the *currentTarget*. By targeting *target*, the object being dragged is the thumbnail; by targeting *currentTarget*, the object being dragged becomes the entire parent container. The importance of *target* and *currentTarget* is that it allows for you to either target specific touchable objects within a container, or the entire container itself. This becomes extremely useful when needing to reposition thumbnails within the entire Personal Collection container.

## Tweener Class

The *asgamer.com* tutorial also introduced me to what makes the application dance: the caurina Tweener class. Tweeing is Flash-speak for animation. Caurina Tweener is an open-source tweening class written by Zeh Fernando, Nate Chatellier, and Arthur Debert. AS3 certainly has its own Tween class (called 'Tween', unsurprisingly), however Tweener is much easier to implement, has faster performance, elegantly handles multiple tweens on a single object, has a slew of very customizable tween properties, and has a very useful 'onComplete' property, which allows you to call a particular function upon completion, or even write an anonymous function. Furthermore, with its 'delay' property, it even doubles up as an ad-hoc timer to call certain functions after a certain amount of time—very useful for animations.

This class became absolutely crucial in making the interface behave smoothly with animations and give it the extra sleek feel.

## Event-Driven Programming

The last foundational concept I learned from the crash-course period was event-driven programming. The concept is really quite simple to understand. In AS3, events are either ‘dispatched’ or ‘handled’.

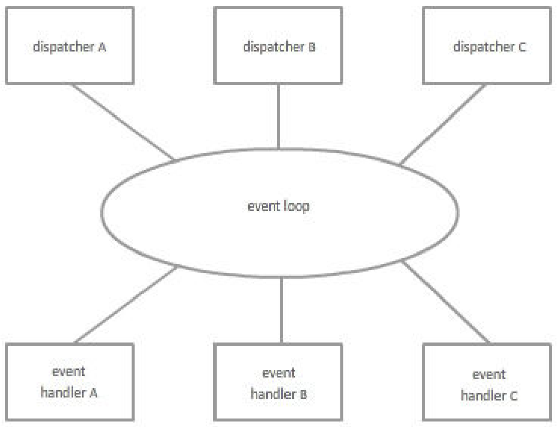


Figure 5. Three event dispatchers and three even handlers

This is the key to creating interactivity and flow to the program. Certain actions will dispatch events. These actions can be a mouse-click, a mouse-hover, a drag, and so on. In the case of Open Exhibits, the actions of interest are gestures, such as touching, releasing, holding, dragging, flicking, and so on. Events can also be dispatched explicitly through code by invoking a dispatchEvent() method. Once an event is dispatched, it enters the event loop to be handled by an event handler. The event handler is a function that will be run when a specific event has occurred. For example, when a “Next” button is touched, the TouchEvent.TOUCH_UP event is called, and it should handler function which advances to the next page.

The interactivity and flow of AS3 is done through event handling. Understanding this key concept is what allowed for us to a workflow in the application.

I was certainly anxious about spending perhaps too much time familiarizing myself with the fundamentals, and not enough time just diving into the program. There was also certainly pressure on MOPA’s end to see me make tangible changes, because until this point, I had only been experimenting in my digital laboratory.  Despite the foundational steps taking the better half of the first two weeks, fortunately they proved crucial to the rest of the development process.  Without the extra 'digging' time, it would have taken longer to discover these treasures along the way.  Now it was time to dive in.

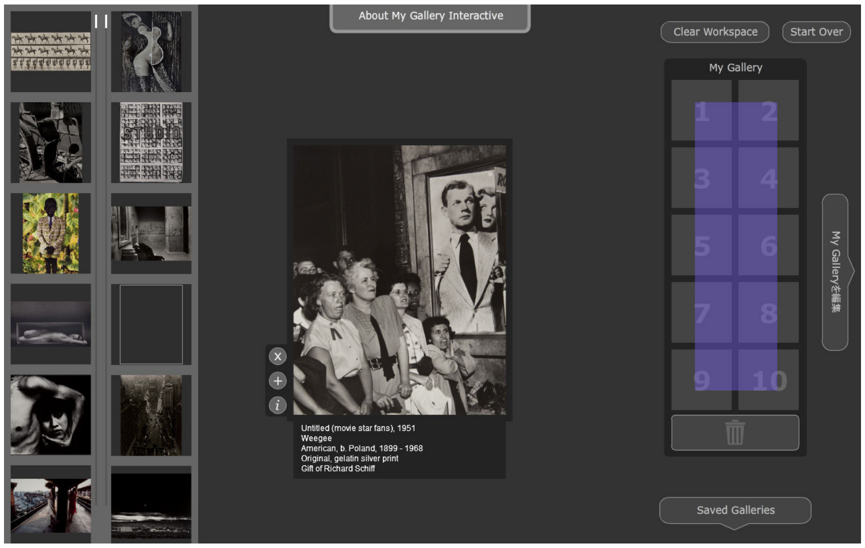
## Personal Collection

The PersonalCollection class is essentially the heart of the application.  It is where users will drop photos they want to add into their collection; it is what users will be attaching a title and author name to; it is what becomes saved and stored to be viewed by other users.  Essentially, the PersonalCollection needed to be able to adapt to each of the different modes in which it would be appearing, visually and programmatically.  The task of implementing a class of this scale took plenty of planning and iterative designs.  It was the bulk of my work during the entire project, because I needed to constantly revise and add additional functionality to the PersonalCollection class.  In all honesty, covering the specifics and details of each of these steps would be tedious and mostly too specialized to be useful.  Instead I would like to focus on two of the most important key concepts that I learned through this development process: Collision Detection and the Bumping Algorithm.

### Collision Detection

An important functionality that the interface needed to feature was the ability to drag and drop, as that gesture would be used to drag thumbnails out of the scrolling area, to move photos into "My Gallery", and to rearrange photos when editing "My Gallery", to name a few.   The challenge was finding a reasonable technique for determining when an object being dragged from one area was within the bounds of dropping into another area.  Fortunately, during my crash courses, I came across an incredibly useful function called hitTestObject(), which is inherited from the DisplayObject class.  Everything on the Flash stage is a DisplayObject, which means any visual component has access to this function.

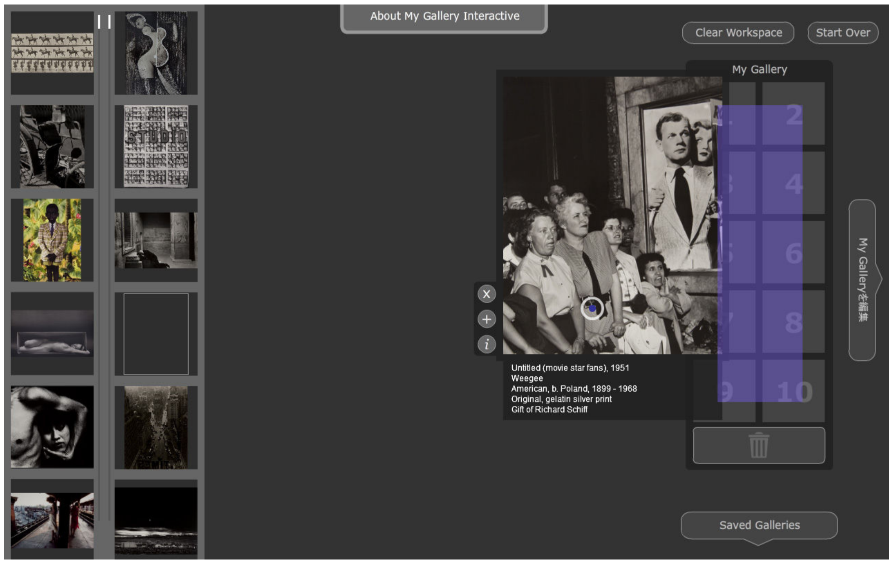
The hitTestObject() function is AS3's collision detector.  What it allowed for me to do is check for any collision that occurs between two objects on the stage during drags, and during drops.  During a drag, you can track the position of the object because you can add a drag event to the object, which dispatches a GESTURE_DRAG touch event every time the object is being dragged.  Within the drag handler function, I can check for whether there is collision between the dragged object and another object on the stage. As such, as I learned in *asgamer*.com's tutorial, I could draw an invisible bounding box within a specific visual component on the stage that would act as a hit area, so to speak. Any time an object crosses into the hit area, it is considered within its bounds. This is particularly useful in something like the My Gallery component on the main screen.



Untitled (movie star fans), 1951
Weegee
American, b. Poland, 1899 - 1968
Original, gelatin silver print
Gift of Richard Schiff

Figure 6. The blue box represents My Gallery's hit area for dropping in.

For dropping a large photo into My Gallery, the hit area drawn is shown in blue. As you can see, the hit area is actually smaller than the actual visual component. Visually, this makes sense. When the user drags the large photo into My Gallery, it should only be droppable into My Gallery when a significant enough portion of the photo is overlapping with My Gallery.



Untitled (movie star fans), 1951
Weegee
American, b. Poland, 1899 - 1968
Original, gelatin silver print
Gift of Richard Schiff

Figure 7. A photo is dragged within the bounds of the hit area.

This allows for a more natural experience for the drag and drop gesture. When the user has dragged a photo significantly into the bounds of My Gallery, then it becomes obvious that it is a deliberate drag into the bounds of My Gallery. On the other hand, if the bounds were set at the borders of My Gallery, as the user is dragging a large photo around in the central workspace, it would make no sense for the photo to be droppable into My Gallery upon the slightest contact. Once the user releases the photo, a separate event is dispatched called the TOUCH_UP event. In the handler function for TOUCH_UP, it runs the same collision detection check to see whether the photo is within the bounds of My Gallery's hit area. In Figure 7, the photo would be added to My Gallery.

This same technique was used countless times throughout the interface's design, particularly with the drag and drop into the garbage can, dragging thumbnails back out of My Gallery, dragging thumbnails directly between the scroller area and My Gallery, and rearranging the order photos. It became a key technique for implementing the drag-and-drop feature. This becomes an essential technique for the bumping algorithm.

## Bumping Algorithm

The bumping algorithm took a significant amount of time to write. Once a user has added the desired photos into their gallery, they may enter the next screen to edit their gallery. On this screen, the user is asked to attach a title and an author name to their gallery. They also have the option of reorganizing the photos, because the aim of the curation process is to have the user think about whether there is any connection between the photos, and have users consciously think about the order of their gallery.

Discussing with MOPA early on in the project, we decided on going with a 'bump' technique for reorganizing the photos in the user's gallery. That is, when a thumbnail is dragged out from one slot and placed into another slot, all thumbnails in between will shift out of its way to provide room for the moved thumbnail to be dropped. Here is an example:

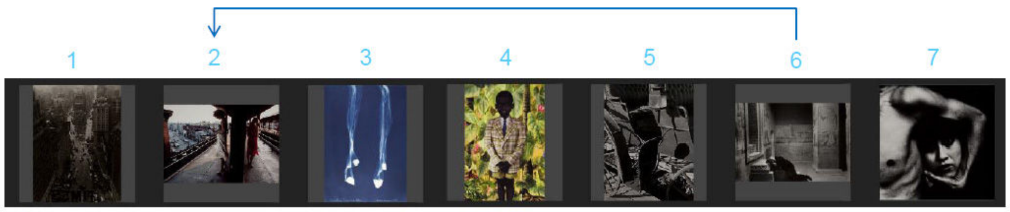


Figure 8. In this example, the user wants to shift the photo at slot 6 to slot 2.

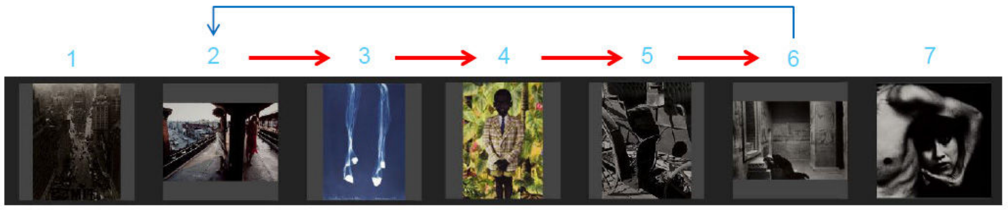


Figure 9. In this case, all images in slots 2 through 5 must shift upward (i.e. to a higher number slot)

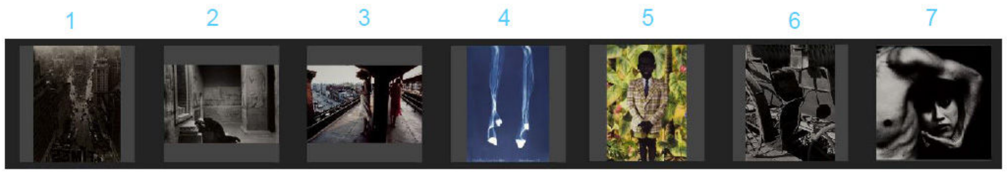


Figure 10. This is how the rearranged sequence would appear.

The difficulty of this algorithm comes from the fact that this all needs to happen on-the-fly. Animations need to occur as the user is dragging to know that the images are being reorganized. So during a drag, each of the slots need to check for whether the dragged thumbnail is overlapping with its respective hit area. The algorithm I wrote is rather complex. I will provide a simplified version of it for clarity's sake. Also for clarity, I will refer to the thumbnail being dragged by the user as the 'dragged thumbnail'. I will refer to the slot that the dragged thumbnail is being dropped into as the 'target slot'.

When a collision with a target slot hit area happens, an inequality check occurs. Each of the thumbnails holds a 'position' value, indicating which slot it is currently in. When the dragged thumbnail's original position is greater than the target slot's number, the target slot's thumbnail and all thumbnails in between must shift up a slot; when the dragged thumbnail's original position is less than the target slot's number, the target slot's thumbnail and all thumbnails in between must shift down a slot. Returning to the above example:

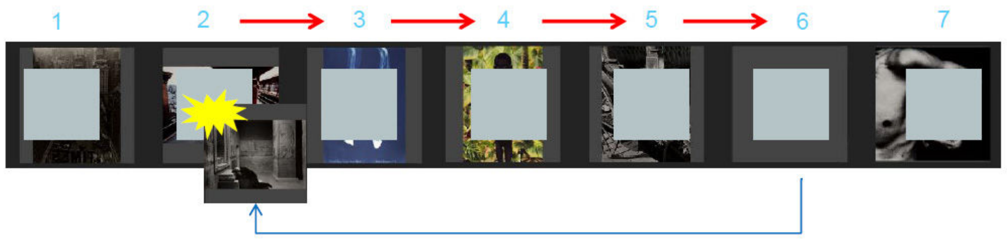


Figure 11. A collision detection occurs. 6 > 2, so thumbnails 2 through 5 must shift up, as indicated by the red arrow.

If the user releases the thumbnail into the target slot, then the layout and the array remain as it is, the dragged thumbnail simply animates to fit into the position of the target slot.

However, a separate condition is made to handle when the user drags the thumbnail away from the target slot *without* releasing the thumbnail. In this case, the thumbnails that shifted over should reposition back to their original locations. I handle this condition by creating a copy of the positioning of the thumbnails the moment a drag occurs. Then, if the original positioning needs to be restored, I just use the copy to restore everything back to its original position.

To better help understand the algorithm, here is a simplified version of the algorithm in pseudo-code:

```
for each slot
      if the dragged thumbnail collides with this target slot
            if the dragged thumbnail's position > this target slot
                  shift everything in between up and update all
                  positions
            if the dragged thumbnail's position < this target slot
                  shift everything in between down and update all
                  positions
      else if there is no collision
            move thumbnails to original positions
```

Note that this is a simplified version. Since this algorithm is run during a drag handler, the code is run constantly as long as there is any drag occurring. To prevent the code from running through unnecessary iterations, several true/false switches need to be added as well, which adds a significant layer of complexity to the algorithm.

There were many more algorithms and implementation techniques that I learned while writing the Personal Collection class, and certainly many very useful techniques to be learned from those procedures as well. However, they are beyond the scope of this paper. The hit area technique and the bumping algorithm are most worth discussing in this case.

## Japanese Keyboard

An unexpected roadblock was designing the Japanese keyboard. Given that the interface is designed to be shown at exhibitions taking place in Japan, it was natural to require a Japanese input system. Japanese input systems vary far more than the English input system, largely due to the complexity of the Japanese writing system. In a nutshell, in Japanese, there are three different scripts: *hiragana*, *katakana*, and *kanji*. *Hiragana* and *katakana* are phonetic letters, somewhat similar to the English alphabet, but each letter represents a spoken syllable. *Hiragana* is used for native Japanese words, while *katakana* is used for borrowed foreign words. *Kanji* is not phonetic. Rather, it is logographic system, each *kanji* with at least one *hiragana* pronunciation. On top of that, as a non-native speaker, it was difficult for me to design a Japanese keyboard for native speakers.

Originally, we had assumed that the soft keyboard could completely imitate actual keyboard keystrokes. However, it turns out that due to limitations of the Flash Player, it is not possible. Instead, we can only map particular keys to particular Unicode letters. Inputting *hiragana* and *katakana* would be as simple as mapping a key to the appropriate Unicode. However, inputting kanji is a different story, because kanji input is generally handled by typing a pronunciation, and a pop-up window with a list of candidate kanji appears. So we opted to forgo the ability to input kanji. It would shorten the development time, and it would also keep titles shorter and simpler.

Another problem emerged from this decision. *Hiragana* and *katakana* input generally have two primary input systems on keyboards. The more popular input system is typing in *romaji*, which is a Romanized version of Japanese pronunciation. The user types with alphabetic letters, and the keyboard layout is essentially the same as the English keyboard, with exceptions to punctuation and script switching keys. The second input system is typing in *kana*, where each of the *hiragana* and *katakana* are mapped to a key on the keyboard.

Since we can only map keys to Unicode characters, it made the first *romaji* input system impossible. It seemed that the *kana* input system was the only viable option given the amount of time we had. However, Japanese users used to the *romaji* input system are generally unfamiliar with the *kana* input system. I spoke with our lab manager, Masaki Chikama, who types strictly in the *romaji* system, and he said he struggles to input in the *kana* system, because he is unfamiliar with the layout of the kana characters. His recommendation was to follow the order in which *hiragana* and *katakana* is taught in school. Much like the English alphabet, *hiragana* and *katakana* follow an order as well. He said most Japanese soft keyboards, such as ATMs, follow this system as well. I worked with Chikama-san to look up layouts of existing soft keyboards as a reference.

Collaborating with Chikama-san proved to be very helpful, because he was able to see issues with the keyboard design from a native-speaker's perspective. He recommended adding an alphanumeric mode for Japanese speakers, because some users may want to type English words, but are unfamiliar with the English keyboard layout. He was also able to help me make tweaks to the positioning of keys and the symbols used on the keys to become more intuitive for Japanese users.

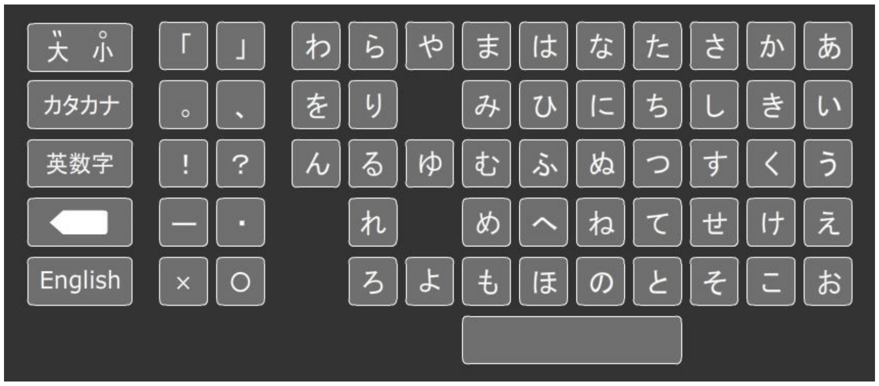

「 」 わ ら や ま は な た さ か あ
カタカナ 。 、 を り み ひ に ち し き い
英数字 ! ? ん る ゆ む ふ ぬ つ す く う
ー ・ れ め へ ね て せ け え
English × ○ ろ よ も ほ の と そ こ お

Figure 12. Japanese keyboard in hiragana mode

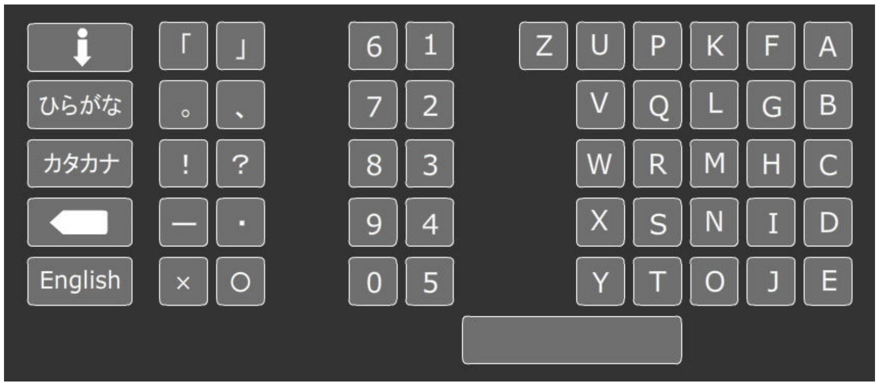

「 」 6 1 Z U P K F A
ひらがな 。 、 7 2 V Q L G B
カタカナ ! ? 8 3 W R M H C
ー ・ 9 4 X S N I D
English × ○ 0 5 Y T O J E

Figure 13. Japanese keyboard in alphanumeric mode

## Interface Design Principles

I come from a mostly design background. This project was a heavily interface design focused project, particularly for me. My focus in the project was designing the interface to be friendly and intuitive. The application is not meant to be too complex. The experience is meant to be casual and manageable within a few minutes. By keeping the look simple with low colors, the photography can come forward with more complex colors. In the process of designing the interface, I encountered several important interface design principles. These principles are worth mentioning, because they will be guiding paradigms to abide by in future design projects.

### User-Centered Design

The overarching design principle for this entire project was user-centered design. Too often, design processes occur without enough emphasis on the user. Sacrifices are often made in intuitiveness for ease of implementation. My goal was to work with an end in mind, and find the means to accommodate those ends. With that in mind, user testing was crucial to the process. At the half-way point when the interface was at a stage where users could experiment with the interface, I would often leave the table on in the lab as we worked. On the occasions that we had visitors, if they had an opportunity to interact with the table, I would watch closely to how they used the interface, and get any feedback I could. MOPA had access to the latest working copy of the application as well, and we scheduled three user-tests with MOPA visitors as well. One of the three groups were children. It was important to conduct user tests from a wide-ranging audience of people who had no relation to the project.

The user-testing process proved to be one of the most informative and rewarding steps in the project. Conducting user tests give us the opportunity to see where the strengths and weaknesses were in our application. It is nearly impossible to predict all of the actions a user will take without actually having users actually test the application. The user tests revealed many bugs in our code that we had not tested for, which was very useful for the debugging process and for making improvements. The user tests also reveal what works well. On those occasions, the sense of reward I got from a design and implementation succeeding was pretty remarkable.

For a project like this, the user experience is truly the pillar of the project. As the developers and designers of the project, we have the ability to make changes that accommodate the desires and expectations of users. The goal of user-centered design is to make sure the experience of interacting with the application is an enjoyable and friendly one, so it is important not to be remiss in ease of use.

## Direction and Flow

During the last stages of the development process, it was time to add direction to the interface. Without any guidance to the interface, it may be a bit intimidating or directionless for a first-time user. Creating a direction and well-paced flow to an interface can be very tricky. In our case, the difficulty is maintaining a balance between allowing the user freedom to explore and providing the user with direction for the next step.

The simplest step was providing a start screen for the user, where the user is presented with a short text provided by MOPA, briefly introducing the user to the project and prompting them to see if they can find any connections between the photos appearing in the collection to be shown. As such, there is already a goal in mind for the user before they begin. The 'Begin' button provides a clear and distinct starting point for the user. In that way, the answer to the question "what do I start?" becomes clear.

The trickiest part was creating direction for the main interface. The user needs a way to be informed of what to do next, and also of the various features of the interface. I decided to use word bubbles as guidance cues, because they are simple, flexible, movable, and uninvasive to the workflow. I could use them to reference specific interface components, and they could be coupled with a simple gesture animation if referring to a specific gesture. The difficulty was getting the timing right.

Gesture cues are meant to be helpful, but they should not be patronizing or annoying. That is, they should not provide directions for obvious steps, nor should they guide the user to a step they have already learned. An appropriate amount of time needs to pass before the next gesture, to allow the user time to experiment with the current step. The gesture cues should also only appear at appropriate moments. It is important to strike a balance with gesture cues. Below are the key points I kept in mind, put more concisely:

Guidance cues SHOULD

- Be uninvasive to the workflow
- Only appear when necessary
- Have friendly timing and frequency
- Give the user friendly direction during idle states
- Disappear when user has performed an action

Guidance cues SHOULD NOT

- Patronize the user's intelligence
- Be overbearing and annoying
- Provide directions for obvious steps
- Appear at inappropriate times
- Reappear when user has already performed an action

I designed three types of guidance cue bubbles.

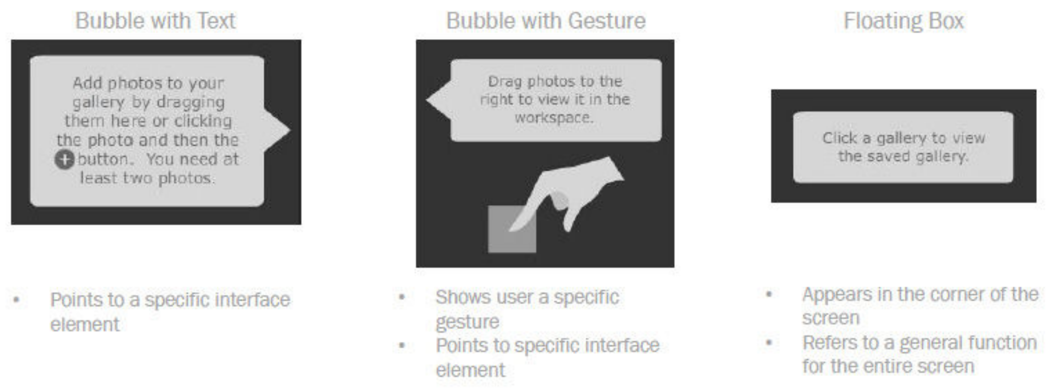
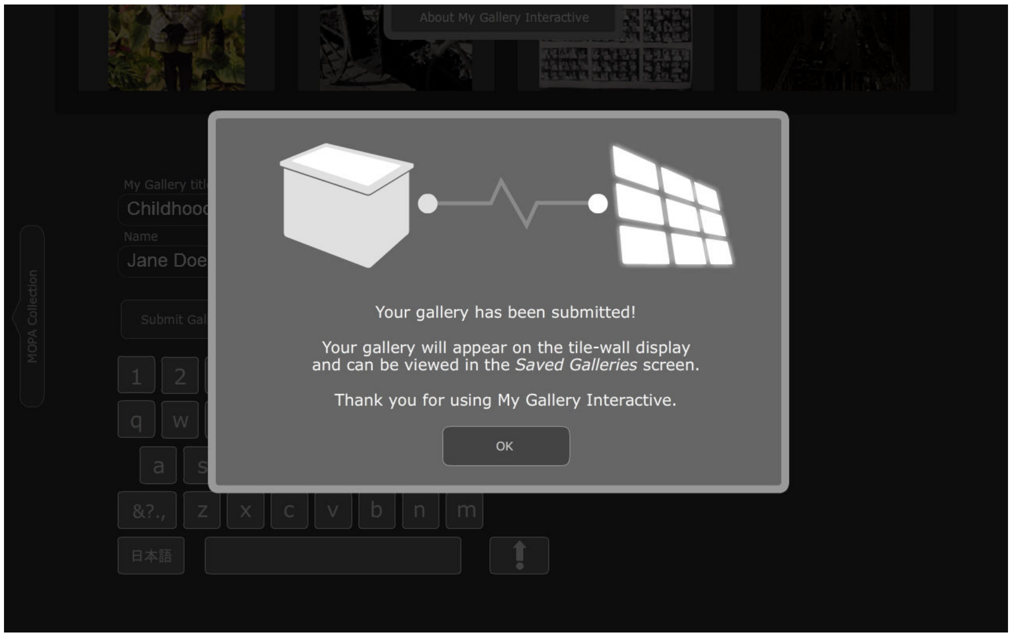

Bubble with Text
Add photos to your gallery by dragging them here or clicking the photo and then the + button. You need at least two photos.
Points to a specific interface element
Bubble with Gesture
Drag photos to the right to view it in the workspace.
Shows user a specific gesture
Points to specific interface element
Floating Box
Click a gallery to view the saved gallery.
Appears in the corner of the screen
Refers to a general function for the entire screen

Figure 14. Three different types of guidance cue bubbles.

Lastly, it was important that the user knows when they have reached an ending point. It is very disconcerting as a user when an interface does not provide a clear end. Furthermore, at that end point, the user needs a sense of accomplishment and completion. For our interface, I provided the user with a short animation to show their gallery has been submitted to the tile-wall and has been saved into the system, and the gallery is immediately displayed on the tile-wall. The user is also thanked for using the application, and given an OK button that restarts another session for the next user.


Your gallery has been submitted!

Your gallery will appear on the tile-wall display and can be viewed in the *Saved Galleries* screen.

Thank you for using My Gallery Interactive.

Figure 15. Confirmation that a gallery has been submitted. It also marks the end of a session.

By adding an introductory screen, subtle guidance cues, and a closing screen, the flow of the experience becomes much more clear cut. The experience has a clear beginning, the direction of the workflow is clear, and the ending provides the user with closure and a sense of contribution.

### Visual Feedback

Originally when a user dragged a photograph into My Gallery, there was no visual feedback. MOPA's suggestion was to add some kind of visual indication during dragging that the photo will be dropped into My Gallery if released at that point. I added a subtle glow to My Gallery when the photo hits My Gallery's hit area. From a visual perspective, the addition made a large difference. It became clear that by dragging the photo above My Gallery, there was some kind of interaction between the photo and My Gallery.

The necessity of visual feedback became clear to me at that point. When a user interacts with an interface and manipulates visual elements on screen, it is important that the user gets visual feedback for their actions. Without visual feedback that is appropriate and intuitive, a user will very quickly assume something does not function or give up altogether. As an example, when a user touches a thumbnail, the thumbnail becomes transparent and expands. Something as simple as visual feedback lets the user know their action was registered.

### Copyrighting

I added the branding assets into the interface during the final stages of the project. It was important for me to be careful and sensitive with branding issues, because oftentimes organizations are sensitive about the appearance of their graphical assets. Logos oftentimes have limitations on coloring, spacing, sizing, and orientation. Also, it was important to add a credit line that acknowledges all of the proper collaborating and contributing parties involved in the project.

At this step of the project, I made sure to communicate with the proper parties, making sure I had confirmation with the wording of a particular verbiage, or make sure the layout of a logo was approved. Handling the branding assets of a project are very important, to avoid legal issues later on.

## Conclusion

The experience working through this project has taught me plenty of skills that extend beyond this summer internship, the most valuable thing being that I have a much better understanding of *how* I work. This is the most responsibility I have ever had on a project. It is also the largest scale project I have ever worked on.

First, I am a visual thinker. I consider myself more of a designer. Though a significant amount of time was spent drawing designs for the interface, most of my time during this project was spent writing lines of code. Even while programming, I found myself scribbling notes and diagrams in order to help organize my thoughts. Not only did it help me think about algorithms visually, it helped me offload information onto paper to consult later.

Second, I understand the importance of pacing when problem solving. Especially in something as involved as programming and writing algorithms, it is very easy to get so caught up in a specific problem that you lose sight of the larger picture. I found myself spending too much time trying to solve a specific problem, then stepping back and realizing that the problem is not critical to the project as a whole. For issues that were critical, I found that when I took a break from the problem by putting it aside, the solution became much easier when I returned to it later with a fresh perspective.

Third, I learned the benefits of iterative design, or what some call agile programming. The idea of iterative design is to focus on getting small elements of a larger project functioning, then focus on the details later. By having a framework that at least functions, the developer can get immediate feedback from users. This avoids the oft painful problem of building a near-finished product that has not been user tested, only to realize that many foundational components need to be rebuilt. I had never had this much control over such a large scale project. Oftentimes this became a battle between the inner detail oriented designer and the practical programmer. I found myself needing to look past the fact that something was not operating optimally or did not ‘look good’, and push forward to get things functioning. Especially when a deadline is approaching, sometimes it is even better to implement a quick fix that at least works, then worry about the efficiency or design of it at a later time. What became useful was building a list of tasks, and then prioritize them. Every time a task was addressed, I could cross it off the list. This kept things in perspective and prevented me from getting too caught up in specific problems.

The culmination of our hard work and PRIME experience was at the Knowledge Capital Trial 2011 exhibition, which NICT attended presenting 20 works, ours among them. NICT's booth was a collaborative effort by people from a multi-disciplinary background, coming together as VisLab, and headed by Shimojo-sensei. Attending the exhibition in Osaka was definitely the highlight of the internship. Being able to stand alongside some very big name companies, be visited by very important figures in the Japanese technology field, and be in the presence of some of top researches in their fields was truly an honor. Seeing users use our application and be genuinely interested in the technology involved was exceptionally rewarding. The entire experience was a user-test in itself. Inevitably, there were some bumps along the road, where a glitch or two would occur, and we would have to restart the application. Those things are inevitable. However, as my first application ever developed at this scale, and as my first exhibition, I would say it was definitely a huge success. We have several ideas for improvements on the application prior to the next exhibition: Tokyo Photo 2011.

In addition to the three main categories above, I believe the experience has shown me how powerful programming can be as a skill. I must be honest and say I do not see my future in a position where I am writing as much code regularly as I was during this internship—I much prefer design the interface. Despite that fact, I have a greater appreciation for programming, and I can certainly see myself using it as a skill in the future. I have also grown much more comfortable working in a foreign environment. I would absolutely love to work in the technology field in an international setting. Japan is certainly at the top of that list.

## Works Cited

Griffith, Christopher. *Real-World Flash Game Development: How to Follow the Best Practices AND Keep Your Sanity*. Focal Press, 2009. Print.

Spuy, Rex. *Foundation Game Design with Flash*. Apress, 2009. Print.

Open Exhibits. Ideum, November 2011. Web. June 2011. < http://openexhibits.org/>

Parsons, Jonathan. *AS Gamer*. AS Gamer, February 2009. Web. June 2011. <http://asgamer.com>