Story Creator Final Report

Brandon Roberts, Nate Olderman, Billy Rathje, DJ
Maguddayao, Kyle Dybdal

December 14, 2013

1 Project Summary

The Story Creator module combines the Sharing Framework, Object Creator, Visual Editor and Animation modules in order to create a holistic view of Edith in a dynamic webpage.

The Story Creator will be directed to a login page that will prompt them for their username and password. If they do not have an account they may create one in the link that says "Create Account." In the case that they have already have an account they will be directed to the main Edith page where they will be able to create a "story." To do this they will click on objects(this makes them glow and puts them into an array for Visual Editor), give the objects animations by manipulating the selected object in the Visual Editor canvas, and create ways to interact with their story by manipulating other objects. By doing this, students will be able to learn how relationships among objects work in programming.

The Story Creator Module designed the Edith main page, login page, and sign up page. A large portion of time spent by the Story Creator module went into the creation of the three major canvas' on the page that were for the Animation, Object, and Visual Editor Modules. These were created and formatted correctly for what each team needed. The Visual Editor and Object Creator module implement a Kinetic canvas while the Animation team utilize an OCanvas. The canvas for Objects was directly implemented in order for objects saved to the Sharing Framework to be displayed in the canvas. The Story Creator module also added the functionality of clicking on the objects to make them glow and put them into an array for the Visual Editor module to implement.

2 Development Procedures

In order to complete Edith the Story Creator module followed the iterative, extreme programming, and the Boehm spiral model. We initially used the Boehm spiral by determining our objectives,identifying risks, testing, and planning our next cycle. We found quickly that this was not the most applicable because of the amount of changing requirements which forced us to throw away old code. It transformed into an iterative model because we

discovered that we had to repeatedly redo our entire module do to changing requirements and specifications from other groups and the project as a whole. As deadlines drew near we found ourselves switching to the extreme programming model in order to solve issues/bugs and to complete large functions of the project as a whole.

We began with the functionality of receiving JSON objects, adding a time to them, and passing them on to other groups. We also added the functionality to set a time to wait for a function (this was later transferred to Animation). Later our requirements changed completely and we began creating the base webpage view for the entire project. Then, we worked on incorporating the Visual Editor canvas into the main page, setup a main canvas, creating dummy buttons for different functionality, and the canvas for viewing objects. From there we worked on design for the login page, main page, and the registration page. We then worked on getting the Ajax calls up and working with the Sharing Framework. We assisted the Object Creator and Sharing Framework modules with saving and loading from the database as well as setting up the save and load for the entire project. We worked on connecting the various elements from the main page with the groups that they corresponded to. This included loading data from the database and displaying it on the object canvas.

Our process was effective at developing functioning software but very ineffective because we created software that was never used or even thrown away. We mostly implemented console log testing and used JSHint. Our program works because the main page displays all canvases as well as allowing the other modules to function correctly on them. The object panel at the bottom of the screen also displays the objects correctly. Console log testing was effective for showing us how the page interprets the information that we were displaying. JSHint was effective at showing us syntax errors in our code. At times console log testing was ineffective at pointing out why our code was not functioning.

Because of how collaborative our process model was, we also made frequent use of pair programming. We did the majority of programming together as a group, and while individuals often wrote sections of code, we would each look at each others code either as it was being written or immediately after. This helped both to catch bugs and to ensure that everybody

understood the entire code base.

We did both formal testing and informal testing throughout the process of putting together the project. Formal testing involved static analysis and unit tests designed by another group. These helped resolve bugs, but our project specifications changed drastically after testing, so we employed frequent informal testing as an adjunct. This formal testing involved printing and manually interrogating the state of all data structures we produced. Because story creator passes data to multiple teams, ensuring that data structures such as the array that holds highlighted objects are working correctly is imperative. We typically investigated the state of these data structures with multiple conditions (often looking closely at boundary conditions like zero items, several items, one items) to ensure that they worked correctly. However, since the majority of our work involved designing the central website, the vast majority of testing involved simply viewing the website to make sure that it displayed properly. There is no direct way to test, for example, whether object images fail to load except through manual investigation. We frequently examined our interface carefully and by hand to ensure it displayed well, looked to the browser debug console for hidden (or sometimes clear) errors, and tested the website in several browsers and configurations (relatively simple because we all use different browser configurations) to ensure that the implementation was robust.

The following responsibilities are based on the components of the final product:

Nate was responsible for how the various canvases were drawn on the Edith main page as well as their relative sizes to the window. He implemented the canvas' resizing capability whenever the webpage window is resized. He also worked on various bug fixes, clean-up of antiquated code, and implementation of many small functionalities (ie. button method calls). He helped implement the functionality of getting objects from the object creation page into the object canvas below the main canvas. He then wrote and implemented the tutorial on the webpage.

Brandon was responsible for helping the Object Creator module and Sharing Framework gain access to the database with javascript. This included writing load and save functions for the main page that sent and retrieved

data. Brandon was also in charge of porting the Visual Editor code and the Object Creator code so that it would function on the main page. Finally Brandon was in charge of loading data from the database in order to draw objects on the object canvas that were clickable.

Kyle was responsible for much of the design of the website. This process started with making a website mockup, and then designing a logo and color scheme. Lots of work was done with HTML and CSS, putting together the general layout of the page. The goal of the design was to be minimalist and out of the way, keeping as clean of a workspace as possible. He also created the login and registration page.

Billy was responsible for designing, styling, and implementing the object browser panel. He wrote the code for displaying objects in a grid, adding objects individually and from an array, and for sizing/positioning objects. He also worked on CSS styling for the object panel. He implemented a Kinetic JS canvas so that the panel could display objects. He also implemented the "glow" or highlighting functionality which highlights objects and places highlighted objects in an array that the Visual Editor accesses through an accessor method. He worked on some of the data retrieval code (particularly for displaying objects designed in the Object Creation panel) as well as server setup/testing. He also contributed to some of the page layout/design.

3 Requirements Evaluation

3.1 Functional Requirements

- "Navigate Story Files" - The story teller will have the ability to start a new story file, save a story, and close a story. This is a helpful requirement because it allows the entire story to be able to function. This enables the story teller to begin their story and come back to it later. We were not able to complete the "close a story" option. The current story is set as a project and is loaded from the sharing framework. The option to close a story was not completely necessary in the end because the user still has the option to make a new story and begin it.

- "Start a new story file" - The story teller will select an option to create a new story. The option to start a new story is not available. This requirement was not defined well because the "starting a new story

4

file" does not specify if the old one is saved or if there is an option to use pieces from the previous story.

- "Save a story" - The story teller selects the option to save the story. A representation of the story and its objects, scripts, and settings is sent to the Sharing Framework. This was a very helpful requirement because it allows a previous story to be loaded onto the screen after being saved. It was very effective because it was a great way for ajax to be used in order to retrieve necessary information from the database.

- "Close a story" - The user selects an option to close the story. The story is removed from the story editor and animations, scripts, and objects associated with the story are cleared from their respective editors. The ability to close a story is not an option. This requirement seemed unnecessary because the story teller can load a separate story without closing an old one. At the end of the process closing a story did not seem very important.

- "Create an object" - Storyteller will go to the Object Creator screen. Storyteller will select an option to create an object. Storyteller will use the mouse to drag the object. Storyteller will choose where to place the object. Storyteller will drop the object in that place. The ability to create objects and add them to the scene is fundamental to making edith a functioning visual programming system. As a result, this requirement was very important, although it would have been helpful to split it into two requirements, one for making objects and one for adding objects, since these both ended up being complicated functions designed by different teams. The final implementation worked such that an object was created in the object creation window designed by the Object team. When finished, this was uploaded to the server and added to the object browser through an ajax call to the server from the main page designed by the Story Creation team. Adding objects to the canvas is done programmatically within the code made by the Visual Editor team. In the visual editor panel is a function addObject() which adds an object to the animation window. As a result, there was no need to use a drag and drop interface to move objects to the canvas. The Visual Editing team decided on this option in order to allow objects to be added as part of an animation; with the drag-and-drop approach,

objects would have had to be present at the start of the animation to appear in the story.

- "Remove an object" - selects an option to delete objects. User selects the object to delete. The object disappears from the scene. This function is not present in the final implementation. It was originally intended to be used with the previous functional requirement so that if an object was dragged onto the animation canvas it could be removed from the canvas. Since objects are no longer dragged onto the canvas, there is no need for a remove feature. This functional requirement would have been useful had we kept the old implementation, but it is no longer necessary. If we had broken up the previous functional requirement into two functional requirements, we might have been able to decrease the ambiguity surrounding what it means to "remove" an object. We do not need a feature to remove objects from the animation canvas, but removing objects from the object browser might have been a nice feature. If we had decoupled the previous requirement into create an object and add an object to scene, we might have thought to have added functional requirements for remove object and remove object from scene. "Remove object" may have been a nice feature for the final implementation.

- "Move an object" - User selects an option to move objects. User drags the object from its original location to a new one. (This is not an animation) Since the ability to drag and drop objects (see last two requirements) is no longer part of the final implementation, there was no need to implement the "move an object" functional requirement. It would have been used to position objects using drag-and-drop on the animation canvas. Instead, this functionality is handled via parameters in the Visual Editor's addObject() function. This approach is similar, and is probably better for teaching users early on how to program using function parameters. The old requirement would have been useful, however, had we kept the drag-and-drop approach.

- "Develop or edit a script" - Storyteller interacts with the visual editor to develop a new script. Storyteller selects a specific action which is an individual script (All of the scripts interact a main loop). The ability to develop or edit a script is central to the final version of Edith. Creating an animation centers around using the visual editor to develop scripts

that animate objects as well as a main method that calls these scripts. This requirement was met. The Visual Editing team developed a visual programming interface for writing functions, loops, conditionals, and program logic which can be combined together into scripts. The Visual Editor assigns scripts to objects that are highlighted in the object panel designed by the Story Creation team; if no object is highlighted, the script is not used for any particular object (like a global/scene wide function) or is part of the main method. The implementation of this requirement changed somewhat over time. Initially the Story Creation team was going to assign time values to functions made in the Visual Editor, but timing is so essential to visual editor functions and scripts that Visual Editor incorporated timing parameters into the Visual Editor interface.

- "Delete a script" - User interacts with the visual editor to delete a script. In the final version of Edith there is no ability to delete a full script since there is no interface displaying full scripts. However, this functional requirement can be achieved; a user could interact with teh visual editor to delete all the code for a single object or for an entire story, then save the story to record the deletions. There are options to delete functions and other program logic from the Visual Editor panel. This is done simply by dragging function/logic/programming blocks out of the right half of the panel. The functional requirement was useful because the functionality it proposes is present in the final version of Edith. However, it may have been better in retrospect to have used a finer level of granularity in descrbing the functionality and discussed deleting individual logic/object/functions.

- "Save an object or script" - The user wants to save an object or script for reuse. A representation of the object or script is saved. We were able to save objects for reuse by having js functions "grab" the object exported by the Object Creator module. We did not implement saving of scripts individually; scripts must be produced on the fly for each new project. However, this appears to be something that would fall under Visual Editor's responsibilities rather than our own.

- "Animate an object" - User selects a script from the visual editor. User drags or otherwise attaches the script to an object in the scene. This requirement changed over time due to changes in the interaction

with and responsibilities of the Visual Editor module. Animation was achieved by having standard functions that apply to all objects that take the target object as a parameter. The Storyteller can highlight multiple objects for a function in the visual editor to act upon.

- "Play story" - Storyteller selects a "play scene" option. The story plays for the storyteller in the animation viewer at the beginning.

- "Save an object or script" - The user wants to save an object or script for reuse. A representation of the object or script is saved.

- "Animate an object" - User selects a script from the visual editor. User drags or otherwise attaches the script to an object in the scene.

- "Play story" - Storyteller selects a "play scene" option. The story plays for the storyteller in the animation viewer at the beginning. ¿¿¿¿¿¿¿ 70c27ea1704a32f284fcbec5659c86eb326ab640

- "Share a story" - User selects a "publish story option." The story is converted to a representation that the sharing framework can read. The story representation is sent to the sharing framework so that it can share the scene. This was achieved through the use of ajax communicating with the PHP on the server set up by the sharing team. The project is saved as a JSON object, which can then be loaded again.

3.2 Non-Functional Requirements

- "Ease of Use" - The number of errors made by experienced users may differ significantly from those made by first-time or novice user. An experienced user should make no more than 3-5 errors per hour. While there is some complexity to the story editor, an experienced user should not have trouble knowing which options to choose to add objects or play the scene. We never performed any studies to quantify the number of errors made when using Edith, although this could be future work for the project. Judging from anecdotal evidence, it appeared that the system was relatively easy to use (with some instruction from the various teams) during integration demos. Since most of the teams were using parts of Edith designed relatively recently, they could be considered "novice users." Their general ability to learn and use the

system readily suggests that Edith is relatively simple for novices to learn. In the sense that this nonfunctional requirement is specific, it is a useful requirement, but in the sense that testing it involves procedures that we never did, it may have been too difficult a requirement to adhere to effectively.

4 System Design and Architecture

5 Individual Reflections

5.1 Brandon Roberts

Edith was a challenge from start to finish. I believe that the most voluminous issue was directly organizational. When we began this project we believed that we had an idea of what each team was doing, but in fact we had no clue what "Story Creator" actually meant. We had a description of our module that included things like "allows the user to specify animations and other events" and "utilize the Visual Editor to specify actions performed by objects provided by the Object Creator." This was completely fine to begin with and made sense but unfortunately the project did not end up functioning in this way. This meant that time and time again the Story Creator module had to modify/completely scrap large portions of code simply because it would not function correctly with other groups. One large issue we had tied back to the fact that "This module is the least able to function as a stand-alone system." There were times when we learned that the Object Creator module was changing how they save objects, Animation was changing how they interacted with the sprites, Visual Editor changed how they displayed and moved functions, and the Sharing Framework changed how the tables were set up in the database and what we needed to do in order to retrieve information. This all comes together to show that it is difficult to be the final piece of the puzzle.

I also encountered many issues with retrieving objects(sprites) from the database and displaying them inside the object panel. I will take some very useful database debugging and a much better knowledge of Ajax with me. I have also learned a great deal about databases and php as well as javascript. These three things together will be very useful in the future because of how prevalent they are in the field of web development and computer science.

9

In order to complete the project I originally attempted to implement the use cases in my requirements document. That quickly became outdated so I started to use the "adapter" design patter to try and piece together the other groups software. If restarting on this project I would attempt to have a long session with other groups to go through what they planned on doing and how we could put it together. A large problem was that many pieces could not be put together in the way that individual groups had originally designed. If I were to continue on this project I would add functionality to actually work.

5.2 Nate Olderman

5.3 Kyle Dybdal

Edith was a lesson in communication, patience, and flexibility. How we initially assumed the final project to work ended up being very different from what the website ended up being.

6 Glossary and References

- Script: A small program made in the visual editor that controls the object or scene event to be displayed.