Livephoto Senior Project

Team Livephoto

Members
Brian Wyant
Rohit Garg
Brian Soulliard
Gordon Toth

Project Sponsor
Robert Teese, RIT
Priscilla Laws, Dickinson College
David Jackson, Dickinson College

Faculty Coach
Donald Boyd

Project Overview

The idea behind this project is to enable the creation of interactive video vignettes that can be deployed as a learning tool for physics subjects. The video analysis activities will involve inputting data points onto a video or frame with a cursor. In addition to such vignettes, the project shall support user input in the form of multiple choice answers and will also allow for the collection of usage data as desired. It is somewhat difficult to explain the concept of a vignette, so it may be helpful to view an example of an interactive video vignette on our website at http://livephoto.se.rit.edu/repo/dev/BallToss/index.html

Our scope for this project is to create a framework for these vignettes. Our sponsors have an initial idea of 25 vignettes they would like to create. As such, our goal is to support as much of the necessary functionality as possible. This means the creation of highly configurable widgets; including videos, frames, image players, questions, answers, instructions, tables, graphs, and analysis activities. Each has a variety of configurations; what text is shown, what data to report during collection trials, where resources are located and which to use, and so on. Our project

scope does not include two main tasks. Our goal does not include the creation of the final set of 25 vignettes; that will be part of our sponsors' continuation of this project. As well, tools to enable the creation of vignettes are also out of scope - we instead made a JSON format that is suitable for configuring each vignette.

For this project to be successful, we needed to achieve several main goals. First and foremost, most of the features that our sponsors expect should be included. We believe that about 90% of the future necessary functionality is included. Only a few niche features, that are typically used for one vignette each, have not been included, due to being significantly more complex features to support and code. Also, we needed some level of support for mobile devices. While our mobile UI is not beautiful, it does maximize the screen space available for users to interact with our widgets. As well, we have made sure that no functionality is dependent on hover events, meaning that simple user touches can interact with all widgets. Finally, the framework should be of sufficient quality that future teams can make use of it and not abandon our work. While this has yet to be seen, we believe this to be the case.

Basic Requirements

We started work on the project with a strong understanding of the requirements. Another software engineering team had previously developed a prototype in Flash. This prototype clearly demonstrated much of the desired functionality.

Those requirements not covered by the prototype largely dealt with how the vignettes would be accessed. Ideally, vignettes would be supported by a set of major browsers and mobile devices: the latest versions of Internet Explorer, Chrome, Firefox, Safari for Mac, Opera, iOS, Android and Windows Phone. None of the technologies we analyzed had perfect compatibility, though HTML5, the technology we eventually decided to use, came very close. It is incompatible with older versions of browsers, most notably Internet Explorer, but otherwise works on all devices with only minor differences. It was also the only technology that worked in browsers without any additional downloads necessary.

The vignettes consist of multiple pages, which in turn include multiple widgets. These widgets include text, video, questions, answers, video analysis, tables and graphs.

Text widgets are self-explanatory, and simply display static text.

Videos are also fairly self-explanatory. They have basic controls, including pausing, scrubbing, volume setting, and full screen mode. All videos are close-captioned.

Questions are multiple choice, and may include both text and images. Answers are similar to text widgets, but their contents change based on the answers to earlier questions. Users cannot progress to the next page until all questions on the page have been answered.

Video analysis is one of the most complex widgets, so there are a lot of requirements related to it. Video analysis allows the user to go through several frames of video, clicking on an object each frame. The collected data from each click is stored, and can be used later by graphs and tables. Video analysis widgets can be customized in many ways, depending on what is needed for the vignettes. Video analysis may also involve measuring distances, accepting user input for each frame, tracking multiple objects, and revealing centers of mass.

Tables and graphs typically display the data collected from video analysis, but they can also display preset data. Graphs can also display the results of equations performed on the data sets, and the rate of change of the data. Graphs may include lines of best fit. The lines of best fit can either be calculated automatically, or entered by the user through clicking and dragging the line's control points.

Our requirements are documented in detail in our Requirements document.

Constraints

One major constraint of our project was that the sponsor asked that our solution not require server-side execution for the final product delivered to teachers. That is, any use of such a language, such as PHP, Ruby on Rails, or otherwise, would make it more difficult for individual teachers to deploy this solution, and that would be undesirable. While a SaaS model may be a good alternative to this model, we did not pursue it; we did not and do not feel that our sponsor's request in this regard was unreasonable.

Still, this did greatly affect our work. Even as simple a problem as making an index page to all of our vignettes is now infeasible; since JavaScript doesn't have access to the different folders on the server (reliably, at least, since different web servers can disable this functionality entirely), we could not list our vignettes programmatically. Thus, our index page is simply a manually updated page of hyperlinks, updated whenever we felt the need to do so. While a minor issue, this hints at the larger issues that may exist.

For example, everything is organized by meaningful folder locations. A vignette is a named folder in a unique location, pages and widgets are named folders in that vignette folder, templates are a special folder in the root of our deployable folder, and so on. Ultimately, our JavaScript is unable to generically locate and load resources. We start with loading index.json - a special file for each vignette - which subsequently describes the full paths to load for pages, widgets, and their templates.

Had this requirement not been in place, many different architectures may have been (and were) considered. For example, one early setup that we had considered would have involved putting vignette, page, and widget data into a database on our server, actively querying all of this information when we load up a vignette. This would have significantly increased the complexity, and number of moving pieces of, our application. While perhaps it would have worked out long term, we can say that this requirement forced us to be significantly more creative in our work. Within this major constraint that we had, we felt that we made the system work, and reasonably well at that.

Other than this major constraint, there were no other significant constraints. Early on, it was advised that analysis activities make use of <video> elements. Our work found this to be impractical, and the sponsor accepted our suggestion that the use of <img> elements would suffice in its place.

No mentions of implementation-level constraints was made at any time. We were free to use what technologies, standards, etc., as we desired. No specific constraints on packaging our work was made, although in delivering it during weeks 3 and 4 of our second quarter, our sponsor did ask for the work in a specific location so that he would be able to deploy the solution himself for two separate groups of students. Seeing that our repository is very large and couldn't even be delivered over email at the time, this was obviously a reasonable protocol for us to follow.

Development Process

Our team used the spiral methodology to develop our system. Our specific process included identifying weekly tasks prior to our regular meetings with the sponsor, on Tuesday. At this meeting we would discuss our work of the last week, consider design implications, and otherwise discuss issues that were relevant with our sponsor and coach. After concluding issues for which we agreed their presence was necessary, we would then allocate tasks independently of the sponsor and coach. Individuals given tasks would then estimate their own times that each would take at this time.

This plan was developed in the first two weeks of the project. It was discussed with the sponsor and coach, both of whom felt that it was appropriate. When our sponsor expressed interest in the details, which was often, we would explain our decisions and other SE topics with the sponsor.

Our project only identified two primary roles. Brian Wyant took the role of manager early on, while Rohit Garg was identified as the technical lead. During a few weeks of the second quarter, Brian Soulliard took over management details. All tasks were assigned ad hoc on a weekly basis, and no other roles were deemed necessary.

Project Schedule: Planned and Actual

Our team defined two levels of scheduling. Overall, in our project plan, we identified a number of key deliverables to be delivered throughout the project. As well, defined four primary spirals, each focusing on different cohesive feature sets.

Spiral 1, not associated with any release, focused on technology evaluation. During this period, intermingled with various planning activities, we investigated the feasibility of several different platforms for this project. We considered HTML5, Flash, Silverlight, Java, and Quicktime. This spiral went according to plan, as we chose HTML5, and ended one week early.

Spiral 2 was associated with the Layout Release. This spiral was focused on creating a solid framework which would form the basis of our later work, as well as fleshing out a generic layout system. This system would allow ultimately allow us to define slots into which all widgets are placed, and is all configurable by the vignette creator. This release also included a number of basic and high-risk features, all of which were completed; a late spike of work also saw some graphing work getting done before the end of the spiral. This additional work done was our only discrepancy.

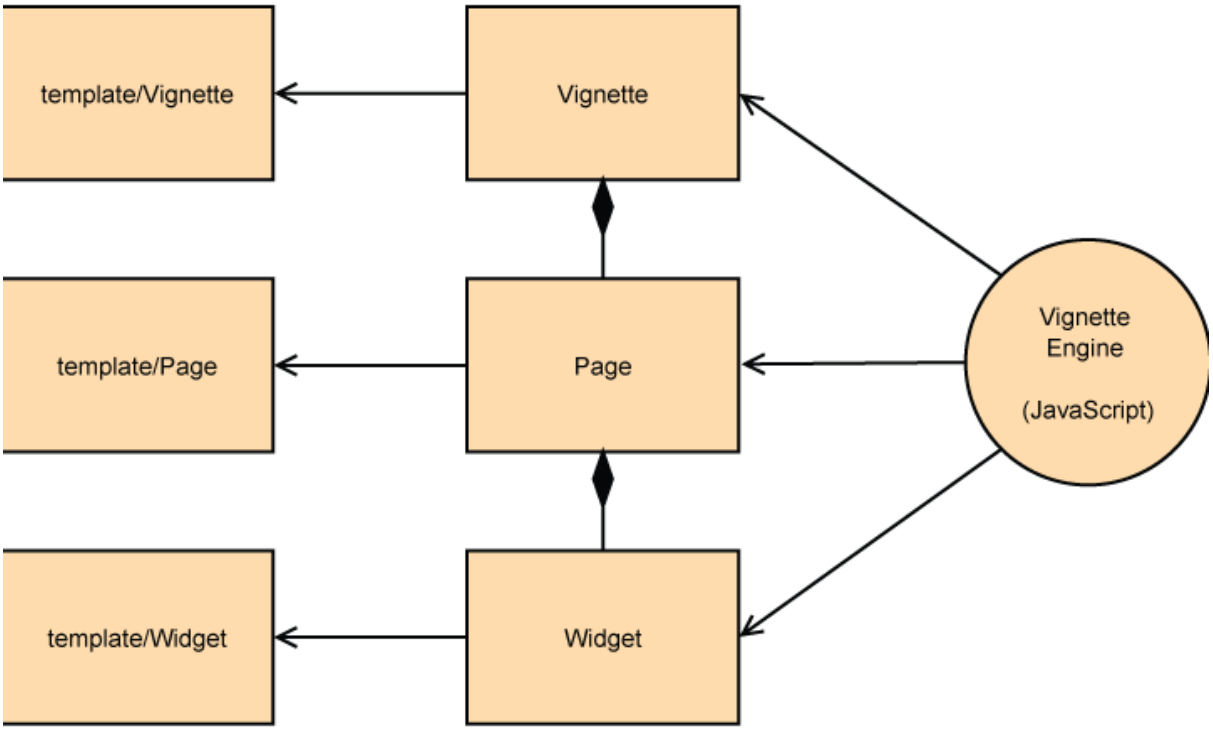
Spiral 3 was associated with the Pages, Graphs, and Mobile Release. This spiral was focused on enabling pagination of our vignettes, giving mobile devices a more screen real estate friendly interface, and otherwise expanding our feature set. Additionally, we took it upon ourselves to

prepare the Projectile Motion Vignette for weeks 3 and 4 of this quarter. This vignette is a real, 7-page vignette, meant to teach students the difference between horizontal and vertical motion. Ultimately, this work led to a higher level of polish, but hampered some of our ability to do feature work. Features left out include sophisticated graphing capabilities.

Spiral 4 was associated with the Feature Expansion Release. Less specified than the other two major releases of the project, this spiral was left open-ended to deal with any late requests that our sponsor may have had. During this time, focus was placed on expanding graphing features, where technically feasible to do so with jqPlot. Additionally, several widgets were rewritten to better honor a Physics Professor's understanding of our work, and several vignettes were made solely to provide an example for future teams using our framework.

Overall, our actual spirals closely adhered to those laid out in our planning. The primary discrepancies include which specific features were included in any given release; short of not completing some stretch features, most spirals would only exchange a few with other spirals. Stretch features were technically sophisticated features only meant for inclusion in one or two vignettes, which our sponsor indicated could be left out with little harm to vignette effectiveness.

System Design



The LivePhoto framework needed to be easy to deploy. The teacher deploying these vignettes would not necessarily have the technical expertise needed to setup server technologies, such as PHP or .NET. This meant that the framework needed to be entirely client side and deploy-able directly to a web server from a single folder.

The LivePhoto architecture is broken up into three tiers of components: vignettes, pages, and widgets.

Vignettes are made up of pages, which in turn are made up of widgets. Pages may be configured to have one of a few layouts for placing widgets. Each widget may be one of nine different types, such as Video, Analysis, Table, Graph, and Question, but are not limited to just these.

A vignette is loaded from its root directory containing an index file. This index file loads in initial jQuery dependencies and calls the Loader script. The Loader script in turn starts up the Vignette Engine. The Vignette Engine loads in other necessary libraries, both custom and third party, before loading in the actual vignette content. The content is loaded recursively going through the vignette, pages, and widgets. When these are loaded, the Vignette Engine looks for template and post-processing files for each. Template files are HTML files embedded with JavaScript and resolved with Underscore.js. Post-processing scripts are run to add dynamic content to these components. Both the templates and the post-processing scripts are configurable with JSON files which hold the information for the structure, behavior, and appearance of the vignette, each page, and each widget. Assets, such as videos or images, are put in Resource directories, referenced in JSON, and pulled in through templates or during post-processing.

The code for the framework is held in three folders. Src contains custom JavaScript classes written for this project providing global functionality. Templates contains the template files and post-processing scripts for individual components. Vignettes contains individual vignette data including all the JSON configuration files used to populate the templates.

New vignettes, pages, and widgets can easily be added to the system. By simply creating the necessary JSON data, new vignettes can be added by dropping a folder into the Vignette directory. New twidgets can be added by providing the HTML templates and post-processing scripts.

Unit tests are contained in the Test directory and are written for QUnit. The test directory contains the necessary libraries, test environment, and QUnit results page for displaying the results of testing every most classes in the system.

Process and Product Metrics

From very early in the project's development, the team's metrics focused on how efficiently meetings were conducted. We tracked how closely we adhered to our meeting schedules, and how many off-topic digressions occurred per meeting (affectionately called "Reichs"). If either of these metrics was especially high, we would re-evaluate how meetings were conducted, but they stayed fairly stable.

To keep metric tracking simple, we relied on Redmine, our project management tool, to track the majority of our simple metrics. These include the number of total and outstanding bugs, the number of assigned and unassigned issues, and the estimated time remaining for assigned tasks. To ensure the code's quality, we tracked the number of JavaScript problems detected by JSLint.

Product State at Time of Delivery

Currently the project has met about almost all of the original planned requirements. The scope of the project included a list of possible features that would be good to have investigated by the end of the project, but were not entirely necessary. As the project went on, new feature ideas were brought up by both the team and the sponsor. These features were evaluated on their relevancy, importance, and ease of implementation and determined if they were worth the time and effort and added if they were. The final delivery includes the entire Mercurial repository containing research prototypes, the completed projectile motion vignette, example/testing vignettes, widget templates, and several experimental branches included to aid future developers in furthering the system.

Testing is not as complete as the team would have liked. There are unit tests contained in a separate folder, /test.

In addition to the code for the system, the server currently hosting the project is being handed over to the summer development team that is continuing the project. This includes Jenkins which pushes a nightly build, runs JavaScript code through JSLint, and compiles the JSDoc reference files. Unit tests are not as complete as the team would have liked but provide a basis for future tests to be developed.

Project Reflection

Overall, many things went well during this project. First and foremost, the sponsor was happy and satisfied; both with our work as it went along, and now, at the conclusion of the project. In terms of functionality and meeting the requirements, we feel that we have achieved success. So while we may now, with hindsight, make changes, these are all in a spirit of continual improvement, and not complete overhaul.

Specifically, a few technical decisions ultimately panned out. While we had many problems with video codecs, once in place, our usage of MediaElementJS allowed us to not focus on the UI of videos. Our usage of images, in place of videos, to back analysis activities was well-deserved, and saved us significant time in resolving timeseeking issues. Our architecture allows for future work to be dropped in by later developers with relative ease. Finally, our overall level of code quality, we feel, is fairly high. It is our hope that future developers, despite not being well-versed in JavaScript, should not have too many problems continuing our work.

What didn't work very well, as alluded to above, was our lack of focus on testing. Early on, we knew that testing our extensive use of JavaScript would be difficult. To this point, we engaged in some early work, after the conclusion of our second spiral (the first non-research spiral), towards automated testing. However, during spiral 3, our focus became redirected towards getting a vignette used in the classroom. In this pursuit, we dropped and stopped supporting even the small backbone of tests we had created before. And, during spiral 4, we decided to push documentation and testing work to the end of the spiral. This decision, mixed with a general sense of senioritis, leaves our testing and documentation in a worse state than would otherwise be desired. It's more than nothing, but less than desirable.

In the future, we might make a few changes. First, in starting our work, some earlier sophisticated designs would have been useful. In general, we skipped forward straight to

developing features fairly early. While it is possible we may not have recognized a good design without first doing some work, this did mean that we had functionality at this early stage that had to be remade from scratch. Further, documentation and testing work should not have been second citizens. Due to the impacts of senioritis, pushing this work to such a late stage ended up being a very bad decision. This work should have been better integrated with prior work products, and considered a first citizen of this project. While everyone had sense at various points of our project that our work would need thorough documentation, and excellent code to be accepted upon our conclusion, this sense appears to have evaporated during our final two weeks. If this pride were maintained, perhaps the state of this work would be different.