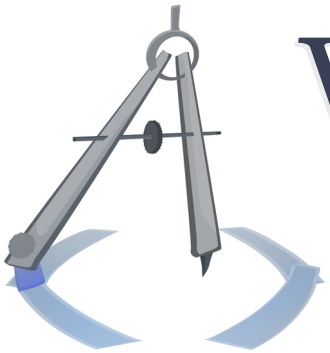


Website Architecture

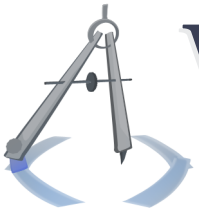
Project Specification

Project 2

The specification for Project 2, which tests proficiency with writing PHP, handling form data, and managing sessions.

Michael Serritella

Summer 2012



Website Architecture

Project Specification | Project 2

Project Goals

This project tests your proficiency in writing PHP, using sessions, receiving form data, and handling & writing HTTP headers.

The project's simulated task is a little contrived, since most sites with server-side programming would have a database and this does not use a database; there are a few tasks in this project that might more typically or mor natrually be addressed with a database. But aside from that, this gives a helpful and realistic example of typical, smaller-sized PHP projects, such as the processing of strings and building of array structures. The use of sessions is a realistic but simple use case, and the receiving of form data is also realisitc but small (since validation can be so tedious in real life, depending on how thoroughly helpful your UI is).

General Description

The project consists of a small website with about half a dozen pages. The user logs in by giving his/her name, explores a dungeon, such as in a typical RPG, and may pick up items or make notations along the way. The dungeon is traversed by arriving in new rooms and then choosing directions such as left/forward/right/back. Upon arriving in a new room, the user may see some randomly-placed items, claim them, and describe them; the described items would then go in an Items page.

Most of the presentation and client-side work have been done for you, so that your primary tasks are managing session information, form validation, and PHP data management in general. The provided code uses MiniArc, though you are not required to use it in any of the code that you write.

Provided Materials

All provided materials are on the course website in a file called "CIS4930_WAM_2012Su_Project2_Provided.zip". This includes:

- PHP files, called "Project2_DungeonQuest.inc", "Project2_DungeonQuest.php", "Project2_DungeonQuest_Items.php", "Project2_DungeonQuest_Login.php", "Project2_DungeonQuest_Logout.php", and "Project2_DungeonQuest_Dungeon.php".
- The CSS file, in the folder "includes", called "Project2_DungeonQuest.css"
- The JavaScript file, in the folder "includes", called "Project2_DungeonQuest.js"

- Its sundry images, in the folder "images"

Requirements

This section details the components of your project along with their requirements, followed by general requirements which apply to the whole project.

The navigation menu

Accessible on each page shall be a navigation menu. It should have different contents, depending on whether the user is logged in. If the user is not logged in, a "Log in" option should be available. If the user is logged in, options "Dungeon", "Items", and "Log out" should be available. The "Dungeon" item shall go to the most recent room the user has visited.

Logging in & out

The user must be able to log in & out, meaning that the user provides a username during login and that name is used to acknolwedge the user (e.g. "Welcome, Greg88") until the user chooses to log out (there is no authentication against a list of usernames & passwords, though this is not much harder and we shall see concerns of password management later in the course). There will be a page dedicated to logging in and a page dedicated to logging out. Each of these pages shall present a prompt to the user. If the user goes to a secondary page for form-data processing, the location and presentation of that page are your choice.

While logged in, each page of the site shall greet the user with his or her username.

The Items page

The Items page shows a list of items the user has claimed from the dungeon during the current session, along with the name of the room (if the user has named it) and user-provided descriptions of the items. In particular, this should be a list of entries with the following data for each item:

- A name for the item; user-provided
- The image of the item as seen in the dungeon, possibly scaled smaller (client-side scaling OK & preferred)
- The name of the room in which the item was claimed, or "(Unknown)"
- A categorization in {Weapon, Gear, Commodity, Trinket}; user-provided
- The estimated value of the item in USD; user-provided
- The number of minutes that have passed between claiming the item and the current page load time (e.g. "15m")

Exploring the dungeon

The main screen, of course, is the exploration of the dungeon. You can refer to the Diagrams section for a visual aid, although you do not have to make sure any particular details from the diagram are represented in your submission, since this project is not about presentation. The vast majority of the look and feel of the dungeon (and the site in general) is given in the provided files.

Navigating

The user may choose to proceed down the left tunnel, the right tunnel, the tunnel ahead, or the tunnel going backwards, which goes to the previous room. You may assume the graph of dungeon nodes is not connected at all and so no rooms happen to be next to each other, etc.; the only relationships you need to assume exist are the ones between the current room and the chosen room.

The user can choose to take a tunnel by clicking on an arrow image. The arrow image should have a link to a resource with some GET data which indicates the choice of room. This GET variable which indicates the choice of room should be of finite length - say, ten characters or fewer - though the depth of the dungeon is conceptually infinite.

Dungeon rooms may have some very basic visual characteristics, such as the color of the walls, which distincuish them from one another. The wall color is already variable with the provided code. If a user traverses to another dungeon room and then back (perhaps going several steps forward and

then several steps back), the room should appear the same (except for its items; see ITems). So you must keep track of some data for each individual room. The appearance of the room may be randomized at first, but it should stay consistent if the room is revisited.

Form submissions in general

Some of the tasks in this project involve form data or GET variables. You choose the destination/processing pages of those forms and how they interact with the rest of the site (e.g. whether you use the same page for the dungeon as the form handler (a "postback" pattern), whether or how the form-handling page redirects to another page, etc.). There is not one correct configuration which you must figure out; several configurations would work for this project and you can choose.

Naming the room

The user may name each dungeon room. The room must maintain the same name across visits and its name can only be set once. Names are not required to be unique (this will not be tested; so you may rely on them to be unique for this project, but in real life, don't make any foolish assumptions in your data structures, such as expecting user-provided data to be your unique IDs).

The naming process is managed by some JavaScript and an HTML form; in the provided code, the form is incomplete and has no handler script yet. The room name must be of alphanumeric ASCII and whitespace characters. The name must contain at least one alphabetic character. The name must not start or end with whitespace, though it should not fail validation if it does; it should simply be fixed so that future representations of the name do not have this problem. Similarly, any whitespace except for the simple space (" ") shall be converted to a single space, without failing validation.

After submitting the form, the user shall see a simple message of success or failure. The failure message does not need to include details. After the success or failure message, the user should be able to return to the dungeon room via a prompt (i.e. not just the "Dungeon" link in the navigation bar), or it may happen automatically a dialog box (already made) and an HTML form (not made).

Items

Users may find items in the dungeon rooms. The items shall be represented by images, randomly placed in coordinates on the screen and layered below all other UI elements, such as the arrows used for navigation. There may be any number of items on the screen, including zero. The set of items in each room is randomized upon each visit. Specifically: If a user fails form validation when claiming

an item, the same set of items is not required to appear when the page reloads.

The images of items shall come from a directory; the set of possible image items shall be whatever the set of images in the directory when the page loads. The paths to this directory shall be represented by PHP constants: `DUNGEON_ITEM_IMAGE_DIRECTORY_FILESYSTEMPATH` and `DUNGEON_ITEM_IMAGE_DIRECTORY_SITERELATIVEPATH`.

Upon clicking the items, the user may fill out a form and claim them for his or her collection. After the claiming process, the same dungeon room is reloaded.

The claiming process is managed by a dialog box (already made) and an HTML form (not made). In the dialog box, the user can set the name of the room if it is not set already. If the validation fails for any part of the form, the room name shall not be set. The rest of the form pertains to the item and all fields are required. The user must give:

- A name for the item, using only ASCII alphanumeric characters, whitespace, and the symbols "#", ".", and "&". The name must contain at least one alphabetic character. The name must not start or end with whitespace, though it should not fail validation if it does; it should simply be fixed so that future representations of the name do not have this problem. Similarly, any whitespace except for the simple space (" ") shall be converted to a single space, without failing validation.
- A categorization of the item in {Weapon, Gear, Commodity, Trinket}, input in any sensible way.
- A nonnegative value for the item in United States Dollars, input in any sensible way.

After submitting the form, the user shall see a simple message of success or failure. The failure message does not need to include details of which field was incorrect. After the success or failure message, the user should be able to return to the dungeon room via a prompt (i.e. not just the "Dungeon" link in the navigation bar), or it may happen automatically.

Custom error pages

Requests for nonexistant resources - 404 errors - should receive responses generated by a custom PHP script. Your PHP script should make the HTTP response have the typical characteristics of a 404. The content of the page shown to the user should refer to the resource that does not exist, such as by saying "The requested resource - /DungeonQuest/blah - was not found".

Nicey-nice URIs

The URIs of the pages should be nicey-nice, such as "/DungeonQuest", "/DungeonQuest/Logout", etc. In particular, they should not show file extensions. Note that a request with a URI which maps to a nonexistant file should still receive a 404 response.

General requirements

These requirements apply to the project overall.

Use of libraries

You may use MiniArc but no other PHP library, since it is hard (or unreasonably annoying) to judge how each library may have contributed to the project requirements; at least MiniArc is a known quantity. You are not required to use MiniArc in your code, however.

In Apache, you may use any module. Be aware that they might not be installed in your environment by default. For grading purposes, you can assume that the installation or availability of a library will not be a problem. You may want to search for how to install a module, however; it's not that big a deal.

Value-added content

For fun or education, any benign content may be added to that which is required - except that which is required to not exist - and it would not incur a penalty.

In particular, you may use any set of images for treasure items; the included ones are just suggestions. You should probably use PNG and GIF images, since they support transparency and they would look a little nicer on the dungeon background.

Use of the HTPPS and surrounding architecture

This project uses the HTPPS in order to render the markup of the pages. The class/derivation structure of the HTPPS in this case is exactly what could be done for a real site. Read, or at least peruse, the code and comments of the specialized pretty printer `DungeonQuestPP`. You do not have to use this; you could just call `print()` anywhere or you could do whatever. But you should probably try to go with the flow, as it will teach you some software engineering principles in a hands-on manner.

For example, try swapping out the (proxied) pretty printer or changing the scribe, such as adding a carbon-copy scribe that writes to a file or a string; notice how the application code doesn't have to change whatsoever; notice how cool this is; etc.

Getting Started

This project is generally more linear than Project 1, in that you only have to write in one or two languages without asynchronous runtime interdependnecies (such as exist in HTML, CSS, JS). So it is "just a program". However, it may seem that there are several ways to get started, and you may be stuck. That is very realistic. Despite that, here are a few tips for getting started; you may want to attempt these as the first few tasks.

1. Convert the given URIs to be nicey-nice
2. Custom error pages
3. Logging in & out
4. Going to various dungeon rooms and persisting some data about the rooms, such as their names or visual characteristics, so that when a user goes back and forth between rooms, the rooms maintain this information and appear to have the same names as before, etc. The names can be arbitrarily set by you at this point.
5. Anything/everything else
6. Validation, probably

Submission

This project has a preliminary due date of **Saturday, July 28, 2012, at 11:59:59 PM EDT**. In order to retain the privilege of resubmitting the project and improving your score, you must submit at least some of your own work by this time.

You may submit your work in one of three ways; choose the first option which is available at the time you wish to submit:

1. A form on the CIS4930.com course website
2. A Blackboard site for the course
3. Email; **mas04m@my.fsu.edu**

The project must be contained within a folder called "Project2_*[Surname]*", where *[Surname]* is your surname, using only alphabetic characters, without spaces, with the first letter of each word in your surname capitalized within the representation. That folder must be contained within a .zip file, such that the folder is the top-level element in the zip file. The zip file must be named "Project2_*[Surname]*.zip".

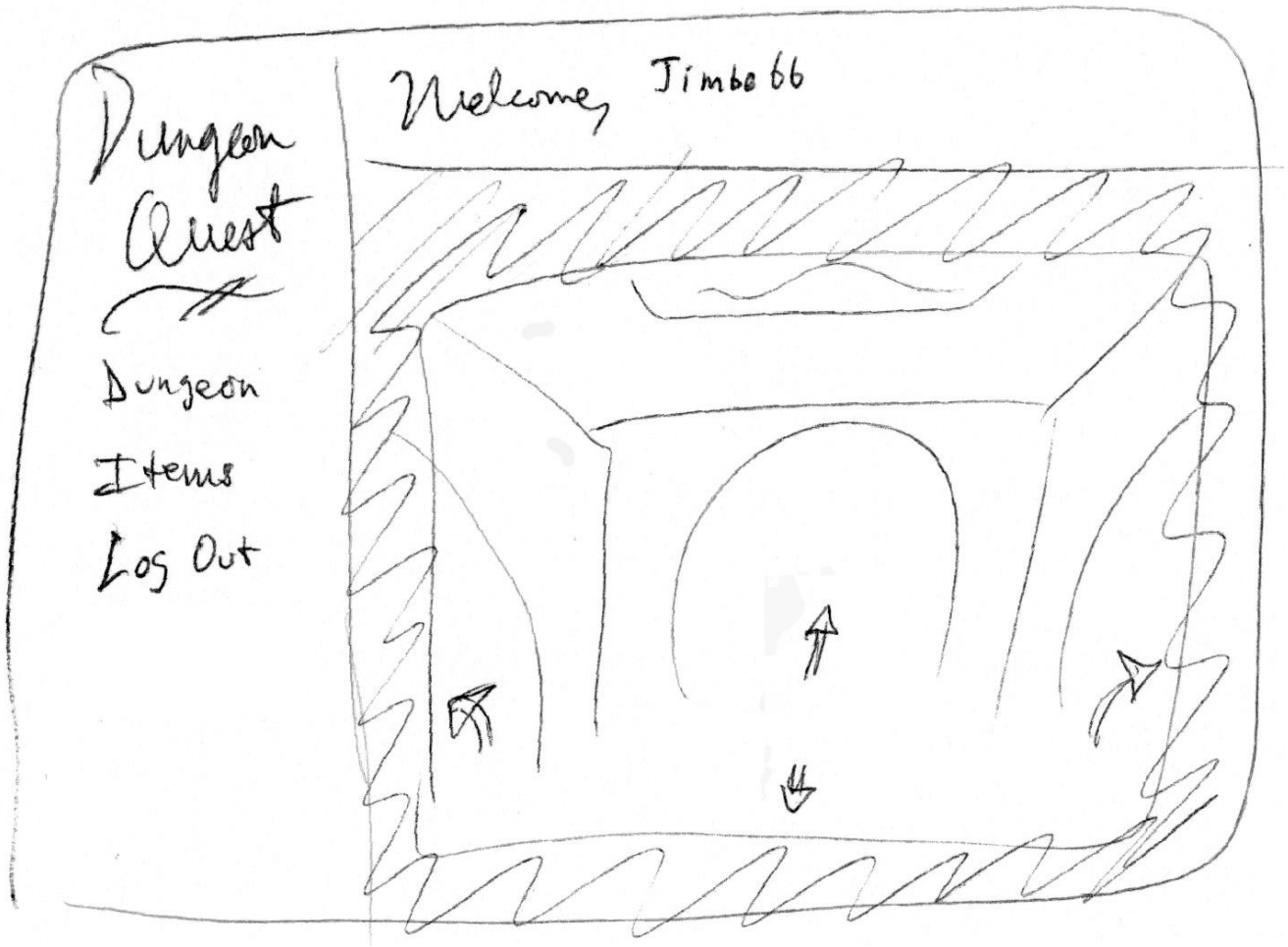
In Linux, you can achieve this by navigating to the parent of your project directory and executing the command: **zip -r Project2_*[Surname]*.zip Project2_*[Surname]***

Future Work

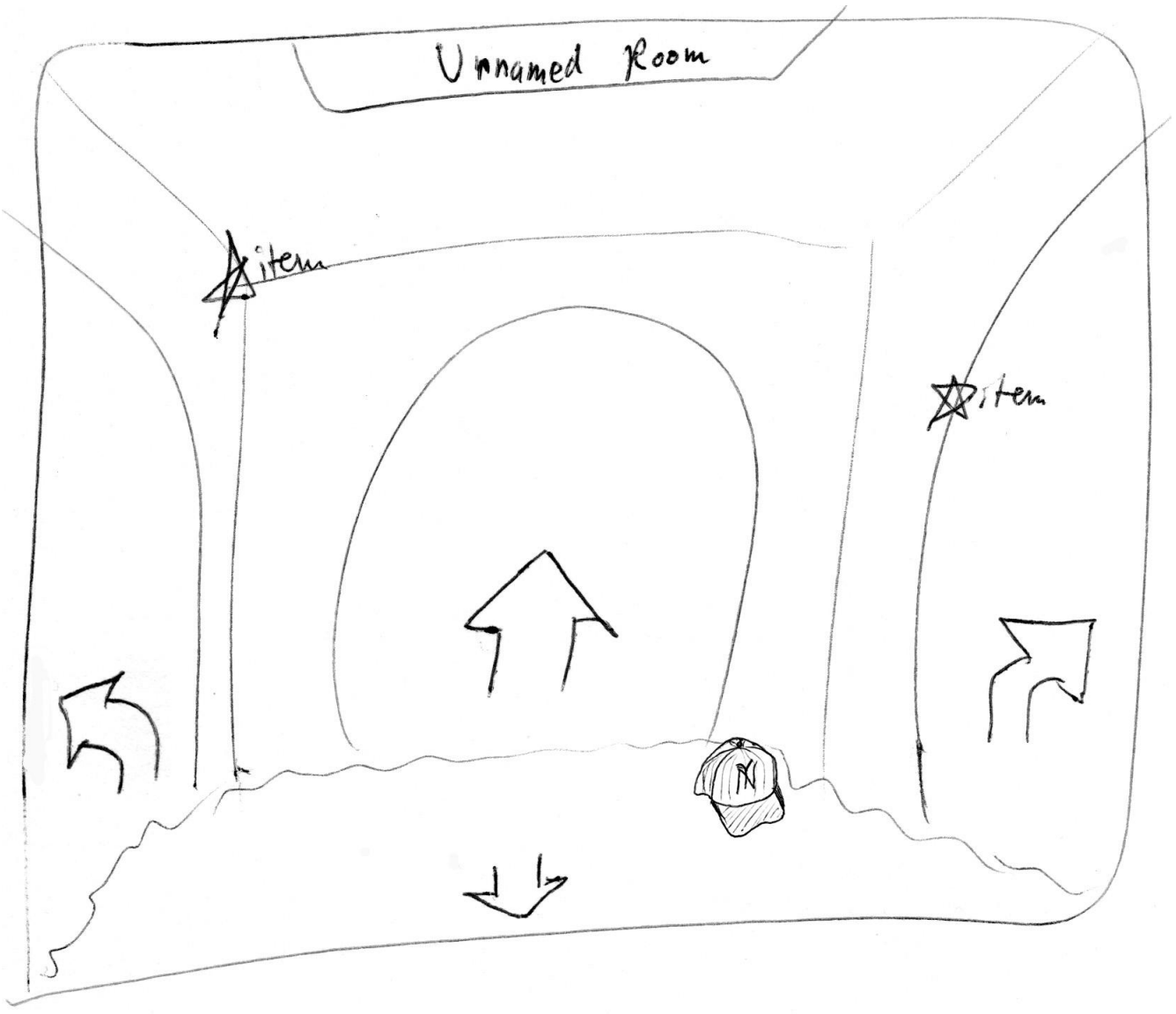
This project has a pretty good potential for expansion. Here are some ideas for your future work which are not required for this submission.

- Keep track of how many times the user has taken each of the presented paths; e.g. "You have gone left from here 60% of the time."
- Allow for the user to use multiple tabs or windows and make multiple simultaneous ventures (you probably didn't think about it, but your current solution probably breaks in this case).
- Show some kind of indicator of overall dungeon depth or a map/tree of the previous choices
- More helpful/courteous UI in form validation, such as showing them their previous values and letting them resubmit if they give erroneous data

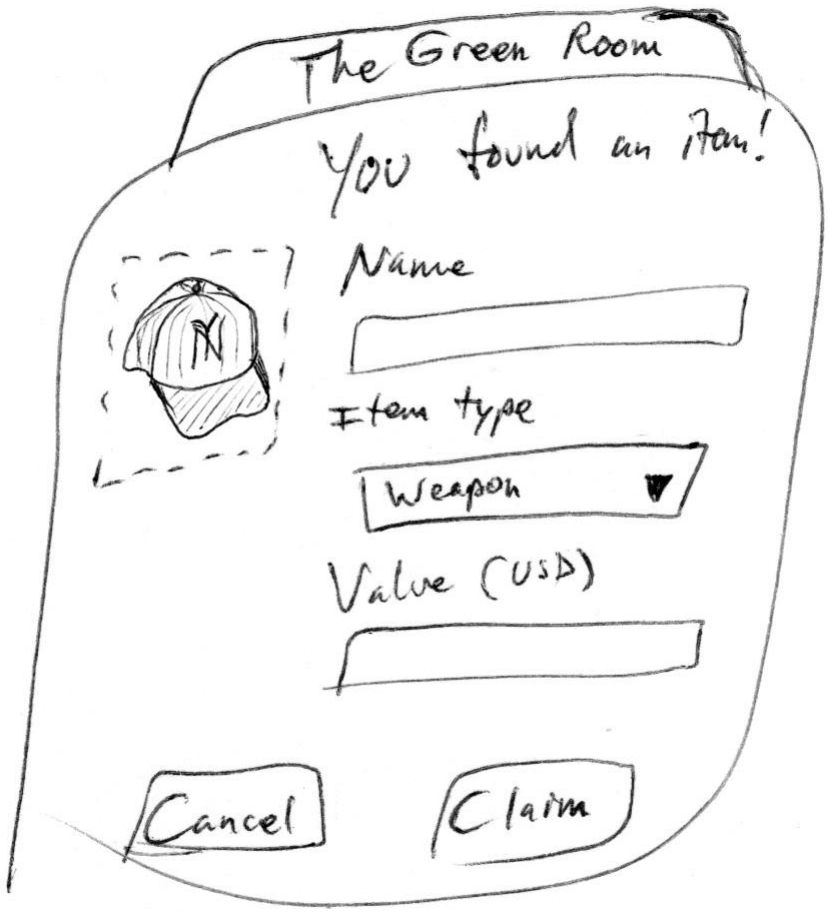
Diagrams



The overall layout. You may zoom in for small details.



The dungeon view. You may zoom in for small details.



The dialog box for claiming items. You may zoom in for small details.