Andres Martin, Matt Mahan, and Matt Rundle
CSE40746 - Advanced Database Projects
Team 3

Table of Contents

Abstract

Our project is an online game, loosely modeled off of the popular Pokemon game series, that features real animals in lieu of fantasy creatures. The different broad types of animals in the animal kingdom (birds, insects, mammals, reptiles, etc.) and the numeric attributes needed for an arena-style game naturally lend themselves to representation in a database. We also use tables in the database to store the animals' assigned attack moves, which are based off the animals' types and real life characteristics, and to store the persistent status effects inflicted by these moves. The game is hosted on a polished, secure website on which users may create and save unique teams of animals with which to battle a computer controlled team.

Project Management

Team Organization, Roles, and Decision Making

We did not assign strict team roles for this project; the small team size of three allowed for relatively easy collaboration between all group members. However, we did periodically assign individual tasks throughout development and did establish specializations within the project as a result. In general, Matt Rundle handled the front-end design and implementation as well as user authentication, Andres Martin was the primary designer and implementer of the game's battle logic, and Matt Mahan designed and implemented the deprecated matchmaking feature and team selection. Some design facets that were collaboratively developed include the database design, the general API schema, and interactions between different parts of the system.

The team's decision making process was relatively informal in structure, but was based off of a large amount of team communication. The team utilized an online chat-room that served as the general forum for the project, and also participated in informal weekly meetings to discuss the status of the project and upcoming project deliverables. All design decisions, save for those that were well within the scope of one person's individual assignment and of no major impact to other areas of the project, were brought forth for team discussion via one of these communication channels. The team also convened for longer, more formal discussions about four times throughout the duration of the project. These meetings served to solidify project direction and allowed for extended discussion necessary to work through the logic of complex and defining design decisions. These meetings also provided the groundwork and direction for individual work by the team members.

Change Management and Version Control

Our project used a git repository on the virtual machine to handle version control. This allowed to us to work simultaneously on the repo without having to worry about clobbering and being able to roll back changes that broke the site. Each one of us had a different technique. Andres Martin kept a repository on his local Linux desktop at home and would push changes in masse to check them. Matt Rundle had a clone of the repository in his project directory on the virtual machine and would push changes from there. Matt Mahan went a step further and made a separate git repository within the same apache server so that he could test changes on a development site.

Code Organization

Our back-end code is organized and handled by a single top level PHP file called "api.php," which we will call the "API Handler" in this document. User interaction on any of the HTML pages results in a POST request sent to this file using JavaScript AJAX calls via the jQuery library. The API Handler then parses the request to determine what functionality is required, and then calls various functions from other PHP files to process the request and return relevant information to the front end for processing and representation.

In the top directory of our project folder is the API Handler, the various HTML pages by which the user will interact with our site, and directories for images, javascript libraries, css, audio files, and documentation. There is also a directory called "api" that contains an "api_handler.ph" file and various other PHP files that are called by the API Handler to process requests. Within this "api" directory is another directory called "battleMechanics," which contains various PHP files that handle battle related logic calculations and various battle test files. We created a directory just for battle mechanics due to the amount of extensive testing that needed to be done.

Verification and Testing

Originally we had a file named “battle_driver.html” within battleMechanics that was the testing grounds for all of our battle calculations before we had a decent visual representation of the game state. From here we tested our starting battle calculations. Similarly, other independent mechanisms, like user login, team selection, and multiplayer, were tested with prototype HTML pages that were later polished and linked to the main home.html page.

Once the project came together and we started connecting pieces we created a global JavaScript debug flag that could be turned off and on. Turning it on would cause all html pages to output, using a pop-up alert, the game state as well as any errors on the PHP side. This allowed to us to make sure everything was as expected or fix any errors in the PHP logic. For final debugging, we turned on this global debug flag and extensively tested all of the website’s core features while examining this debugging output, the examination of which allowed us to find and fix many different errors.

Implementation

Database

The database used was the Oracle SQL database provided to us for the class. The general design of the database can be split generally into two elements: game information and user information.

For game information, we conceptualized our design as a mixture of a traditional animal taxonomy hierarchy and Pokemon mechanics. There were two distinct sets of tables: one set of tables with game data and another set with user data. The game data has 6 tables. *Types* would be our highest level table. In this table animal "types" are listed. Next there's the *Animals* table for the animals themselves, with a foreign key of type. Adjacent to these two tables are *Effects* and *Moves*. If a move has a status effect tied with it then there is a check that it exists in *Effects*. Tieing *Animals* and *Moves* together is *Animal_Moves*. Then finally *Stats* has the attributes/statistics of each individual animal.

For user information, the database contains relies on one table, called "player," to store user information that is used for login and authentication as well as for the tracking of historical game statistics (wins and losses.)

PHP

Much of the in game logic is done on the back end via PHP scripts. Whenever an API call (a POST request) is made from any of the HTML pages to the the api.php file, the request is parsed and then sent to the API_handler.php file for further processing. API_handler.php will then often call a function in a more specific file (battleMechanics.php if it involves battle, animal_selection_functions.php if it involves team selection) to perform the appropriate calculations and return relevant information. The functions in animal_selection_functions.php involve querying the database to generate lists of available animals or animal teams. The battleMechanics.php script is in

charge of the many calculations (hit chance, damage, critical hit chance, status effects, swapping, death etc.) needed to generate the new game state after user actions, as well as building the battle log.

The project's PHP API implementation also uses the concept of "secrets" to handle both user login and the validation of subsequent API calls. This "secret" is simply a unique, randomly generated string that is returned to the front-end client after they have successfully been authorized and identified, either by logging in or creating a new user. This "secret" is then effectively used in lieu of a username, as it is stored in the back-end database and indicates that the user is properly authenticated. The "secret," stored in local storage by the client, is routinely sent to the server as an attachment to the other types of API calls, as well as through a standalone authentication API call that is performed upon loading each page of our website. If the "secret" attached to any API call is found to be invalid, the API call will refuse to return the requested data and the user will be logged out and sent back to the home screen. This concept was implemented as a basic tamper-proof mechanism; it disallows the sending of malicious API calls by those who know another user's login name and might nefariously try to impact their win/loss ratio or perform other unauthenticated actions.

JavaScript/HTML/CSS

Javascript is largely in charge of setting up the game (main menu/log in/team selection) and actually visually representing the game state. Most of the visual processing on our HTML pages isn't done by PHP on the back-end; rather, the pages make AJAX API calls and the returned data is then represented with the help of jQuery.

On the "battle.html " page, the game state is received via an AJAX API call and is then visually represented in an HTML5 canvas element. The page uses JavaScript and jQuery to parse the gamestate and add components to the canvas.. We also have css styled buttons to take input like moves and swapping.

As mentioned before, the front-end relies fairly heavily on jQuery, a JavaScript library that allows for easy interaction with the DOM. We utilized jQuery for tasks like button click handling and the sending and receiving of AJAX calls to our back-end API.

The look and feel of the site comes largely from Skeleton, a responsive CSS boilerplate project (see references for a link.) This is a very small CSS framework that provides styles for a few CSS elements, including buttons and tables, as well as provides a grid for element placement. Other than that, only basic CSS custom styling was needed.

Code Location

A full copy of all this project's source code is located at:

/afs/nd.edu/coursesp.15/cse/cse40746.01/dropbox/mrundle/project

Conclusion

The Animalmon project experienced many design changes throughout the lifespan of its development, as evidenced by the trail of design documents generated throughout this semester. We faced many challenges, including differences in vision, technological roadblocks, and complicated design choices. However, we viewed these challenges as learning opportunities, and patiently overcame them.

The scope of the project, along with the due dates of design-related documentation, spurned us to work ahead on the project and communicate habitually and frequently. These two traits came to be our greatest strengths. The head-start we established gave us plenty of time to devote to the initially under-estimated amount of work needed to integrate the disparate elements of our project and to perform end-to-end testing and debugging. Our communication helped us learn how to design schemas and project features collaboratively. And both together helped us adapt in the face of design challenges, giving us the flexibility to shift the project design when we rewrote our early inadequate schema, dropped the extraneous JavaScript gaming library, and deprecated the difficult to integrate multiplayer feature.

As a group, we also have learned a great deal about many different technologies. We gained insight on the process of installing and administering to Oracle databases, PHP installs, and Apache web servers. We learned about front-end technology including JavaScript, jQuery, and CSS boilerplates. Most valuably, however, we learned how to connect these technologies and create a functional product as a result of their combination.

In the end, we have implemented and delivered what we believe to be a functional and polished product that represents a solid achievement of the spirit of the goals that we set forth for our team in early February.

References

PHP documention: http://php.net/manual/en/

Pokemon logic: http://bulbapedia.bulbagarden.net/wiki/Main_Page

Oracle SQL: https://docs.oracle.com/cd/E11882_01/server.112/e41084/toc.htm

jQuery: http://api.jquery.com/

HTML: http://www.w3schools.com/tags/

Skeleton, Responsive CSS Boilerplate: http://getskeleton.com/