Programming Zombies:
A Study of Usability and Collaborative Programming

An Honors Thesis (HONRS 499)

by

Brian M. Bauman

Thesis Advisor
Dr. John C. Dailey, Ph.D.

Ball State University
Muncie, Indiana

May 2010

Expected Date of Graduation

May 2010
Abstract

Much of the current computer science curriculum at Ball State University is based around the best practices of programming - what sorts of algorithms are optimal for performing specific tasks, and why. Less discussed, however, is the broader view of programming, especially programmer interaction and the nature of collaborative software development in general. Similarly, while the current fads and trends of user design are often written about at great lengths, it is much more seldom that the topic of user interfaces is approached from a more universal perspective, in a manner that facilitates the natural evolution of interface tools in any day and age. In my thesis, I document my findings from leading a small group of novice and amateur student programmers through the redesign of a BSU organization's website.

Acknowledgements

I would like to thank the following people for their contributions to my thesis:

- Dr. John Dailey, for advising me during this project, and providing assistance at every stage despite my odd communication schedule.

- Tim McDonald, the webmaster for the Urban Gaming League, for agreeing to allow me guide his developer team through the process of redesigning and producing the UGL Website.

- Kelly Levans, for demanding an understanding of every detail before she would proceed with programming. Her persistence was a valuable asset.

- Colton Callahan, for all his independent work programming and fighting through the inexplicable form bugs.

- Aaron Howard, for appearing out of nowhere midway through the semester and helping us with our graphic design needs.
In the fall of 2007, a Ball State student who had spent a little too much time perusing the internet came to a decision - he would put together a massive, 24/7 game of tag on campus. It would involve people from every major, graduating class, and social clique. After months of planning, lots of advertising, and over four hundred registrations, Andrew Ketrow had done it - the first game of Humans vs. Zombies (HvZ) at Ball State University was up and running.

From the very beginning, the game demonstrated unusual technological needs. During any given game, most in the triple-digit player base don't know the other players. Game play occurs across campus, at all hours of the day and night. Despite the fact that HvZ is an athletic game played entirely outdoors, a surprising amount of online support is necessary for it to function smoothly. Team information must be easily and quickly updatable by the players themselves, and instantly available to the entirety of the player base. Additionally, game administrators need (at minimum) an easy means of sending targeted emails to up-to-date rosters of the various teams.

The first system met these needs, but only barely. The web site ran on a small amount of extra server space begged from one of Ketrow's friends. The code was taken from Goucher College's source code for the same game, and hacked together at the last minute to work with the improvised server. After the first game, I volunteered to rewrite the site from scratch, and my friend Adam Huston petitioned Jo Ann Gora directly for a legitimate server. For the last two years, I've been writing, re-writing, and maintaining the website which has eventually grown into the central hub for the BSU Urban Gaming League (UGL), which coordinates games for several hundred students each semester.

Because of this massive expansion, however, the fundamental nature of the site has changed. Rather than simply remaining the means of controlling a semi-annual game of HvZ,
the UGL website is now the main facility for the daily operations of a campus organization, and the execution of the half-dozen games it runs. The scope of the site has increased drastically, and with it changes must be made to the measures put in place for its maintenance, to counteract several blatant initial flaws.

One major flaw in the site design was a lack of modularity. The original site was intended to support a single game and its players, rather than a complex network of players, administrators, and games. A rudimentary system was quickly cobbled together for the addition of other game modules, a permissions system, and other such necessities, but utilizing them was cumbersome and inefficient, as the site hadn't been constructed with these capabilities at its core. The only way to integrate these features fully was to redevelop the core of the site around them.

Another cause for concern was the absence of complex game control features. When only one programmer is adding all the features for a site that only receives use twice a year, it makes sense to simply manipulate the database directly as need, rather than writing copious amounts of code to facilitate the same manipulation by game administrators. As site use grows, however, it becomes more productive for game administrators to be able to directly control all aspects of their game on their own, and the value of writing such code increases.

A final issue of concern was my eventual graduation. As the sole programmer of the site for the last two years, my sudden departure would be a major setback for the organization. To alleviate this, I determined that it would be prudent to foster the development of replacement programmers to carry on in my stead. I use the plural for several reasons. Firstly, having a team of developers prevents the same problem from arising in the future when the new webmaster must eventually step down. Secondly, the scope of the site is now such that, for the first time, multiple developers is a feasible possibility. Thirdly, the arising complexity of the site demands
a more formal, structured method of developing the software, such that future changes and additions are possible with minimal reprogramming and detailed code analysis. Finally, the candidates to replace me were all relative newcomers to the world of programming, and would be more effective in a group than individually.

The webmaster, Tim McDonald, came into the role with very little web work under his belt - his previous experience had mostly centered around several Java programming courses. Colton Callahan joined the project with an introductory knowledge of PHP programming, and a head for algorithms. Kelly Levans brought a few classes of web development experience to the table, along with knowledge of XHTML and CSS. This mix of skills would prove useful, as each member of the team was able to help the others out to some degree.

Team assembled, the task before us was straightforward, if daunting - design a comprehensive core framework for the UGL website, upon which any number of game modules could be built seamlessly, quickly, and efficiently. This core needed to have a user interface intuitive to an array of students with diverse computing experience. The entire project needed to exemplify crystal-clear coding practices - a completely new programmer should be able to read any script, any subroutine - and be able to determine its function and modify it as necessary.

These challenges would be met with a strategy centered around two important facets of programming - usability design and collaborative programming practices. Through the course of this thesis, we will delve into the theory of each of these fields, my experiences in applying them, and the interplay between the two. To provide the proper framework for this analysis, however, it is necessary to outline the principles of each field before addressing specifics.
The User Interface: Site Design

The first step toward the production of a user-friendly site, before the very first line of code is ever written, is a clear set of goals the site is to accomplish. People build sites to do things. It is important that these goals exist independently of any strategies for their implementation - the question of how to accomplish these goals is thus left open-ended, and can be addressed directly and without obfuscation.

One of the critical mistakes designers make is trying to move too quickly - developing a strategy, scope, and even structure in one cohesive step. The goal in such instances is usually to attempt a more natural approach to developing the site. It's certainly a more convenient approach - no ideas need be shelved for later when everything is worked on simultaneously. The drawback, however, is a complete lack of focus. Compromises must be made in any design process, but this method tends to favor ideas that were conceived earlier than others, rather than selecting and implementing the best ideas. Ultimately, the method emphasizes the wrong things. While it addresses every aspect of design at once, it fails to prioritize the most crucial part of the design strategy - the goals for the site. One could say, therefore, that this method is strategically focused, rather than goal focused.

Others commit a more egregious sin - they immediately start writing code or designing interfaces, with the mindset that they will simply develop what is necessary as need for it arrives. This method seems attractive because it creates tangible results as quickly as possible. Like the above technique, however, this method places far too much emphasis on the wrong part of the process - the implementation. Developers who practice this method very frequently find themselves forced to reconcile disparate elements of design as new needs are developed. Eventually, such reconciliation might even cause production to halt altogether, while the team attempts to salvage the work they have done using the same design processes that would have
prevented their problems in the first place. It is far better to simply move through most of the
design process before implementation even begins (though allowing adjustments after the fact).

In UGL's case, we had a list of common complaints users made while using our site. One
big problem was that new users to the site had no idea of what they were supposed to do. Most
of them had been referred to the site by a game flyer or a recommendation by a friend. Neither
situation provides very much context, so most are looking for some sort of introduction to the
organization, an brief outline of what the site offers, or easy direction to the game of their choice.
Another problem was that the process of managing games was overcomplicated, even for
veterans who knew how the system worked. Thus, our goal was to develop a site that was
simple to use, and streamlined to allow users to accomplish their goals with a minimum of
hassle. Our metric would be somewhat subjective, but still fair - we wanted a sizeable majority
(60%+) of veterans to feel that the new site allowed them to find information and accomplish
tasks more efficiently, and we wanted most (50%) of the new users to consider their experience
on the site a positive one.

Once the ultimate goals of a site have been determined, it is necessary to analyze the
means by which these goals can be accomplished. This analysis is usually performed in two
major steps. The first step segments the target user base into meaningful divisions, and
establishes the needs of each division - a process known as user segmentation. The second step
determines which of the users' needs fall within the purview of the site, and which will be met in
other ways (or not met at all) - otherwise known as the scope of the site. A website that meets all
its users' needs that fall within its scope is inherently successful.

A few weeks into the semester, the UGL development team went through the user
segmentation process. We isolated six major groups, and created a profile for each one -
including their needs, desires, and frustrations. Additionally, we spoke directly to several individuals from each group to obtain third-party input into the matter. The following user needs were determined:

<table>
<thead>
<tr>
<th>User Needs Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single-Game Newbies</strong></td>
</tr>
<tr>
<td>• Easier registration</td>
</tr>
<tr>
<td>• Easier access to their game</td>
</tr>
<tr>
<td>• Password recovery</td>
</tr>
<tr>
<td>• Read-only game access</td>
</tr>
<tr>
<td>• Full mailing lists</td>
</tr>
<tr>
<td><strong>Game Administrators</strong></td>
</tr>
<tr>
<td>• Full game controls</td>
</tr>
<tr>
<td>• Waiver control</td>
</tr>
<tr>
<td>• Game mailing lists</td>
</tr>
<tr>
<td>• Partial game controls</td>
</tr>
<tr>
<td>• Read-only full-game viewing</td>
</tr>
<tr>
<td><strong>UGL Addicts</strong></td>
</tr>
<tr>
<td>• Simultaneous registration for multiple games</td>
</tr>
<tr>
<td>• Website status readout</td>
</tr>
</tbody>
</table>

This simple table provided the framework for all the development that followed. Once we had our users' needs fairly well laid out, our next task was to consider the implications of each need, as well as the best way to accomplish each need effectively. Some needs were fairly trivial to meet - simply including a new control or adding a feature to an existing interface often worked. Password recovery, game controls, and the website status readout all were fairly straightforward extensions of extant code. Other needs, however, required a complete shift in site layout, as well as a new navigation scheme. The desire new players expressed to be able to quickly move to their game while ignoring the rest of the site, resulted in the hotlinks design along the top of the main page. Administrators' desire for more comprehensive controls resulted in a complete overhaul of the permissions system!

Arriving at the above criteria collaboratively was a multi-step process. The initial brainstorming session only produced about half the list of goals. The other half came about through a structured composite sketching exercise. I began by drawing a large rectangle on a
chalkboard, representing the 1000x600 pixel area we were working with. Within that area, we started sketching the new features we wanted to add and the old features we wanted to keep - discussing, editing, and adding to each others' contributions. We determined that the old site wasted precious real estate on a full-width logo, while a tighter configuration granted the previously haphazardly placed login panel a comfortable, intuitive home in the upper right corner. We debated the merits and drawbacks of placing navigation on the left, right, or top of the page, and finally settled on a persistent top-bar global navigation system, with localized navigation appearing on the left in each of the subsections.

Once the key elements had been placed, we looked back through our user needs analysis and attempted to flesh out additional features that would improve the users' experience beyond their stated wants. Our thoughts immediately jumped to a separate complaint we had been receiving on our forum - our game announcements module was bloated, effectively forcing all the forum content below the fold of the page. By making use of a large right-hand column in the home page, we could transfer the announcements to the front page of the site (a more apropos location for them) and clean up our forum in one fell swoop. As we iterated through structure designs, we would run them by UGL members for feedback and suggestions for improvement. See Appendix A for site design iterations.

In the future, we will design skins around the structure, and employ the same user testing process to ensure that our members will find the changes helpful and attractive, whilst simultaneously broadening the universal appeal of the organization. Unfortunately, the rest of the semester was spent designing and building the supporting software for the core framework, so further development of the site design beyond the structure could not take place. Building the core was a challenge in its own right, however, as we would soon discover.
Under the Hood: Collaborative Programming

Determining the approach one will take toward a programming project depends on a large number of parameters, but the resulting conclusions are few. Generally, project styles fall into one of three categories: code-n-fix, waterfall, or agile. The first of these approaches is extremely informal - the programmer(s) simply begin writing code and fix problems as they crop up. It works well for quick prototyping, or development of small and unimportant systems. The first few iterations of the UGL site were written in this style. While it made sense at the time (I was only one programmer, after all), the system it produced was not robust enough to last more than a few years.

The waterfall approach is precisely the opposite of code-n-fix; a great deal of time is spent planning and structuring the development process, then executing it in strict order. This approach is often embraced by large corporations with a lot of personnel who aren't in constant communication, as a centralized, formalized development schema keeps everyone on the same page. Because of its rigid structure, however, it doesn't accommodate problems or desired changes very well.

Given the small size of our group, the overall lack of inexperience, and the voluntary nature of the project, I opted to use primarily agile software development methods, rather than waterfall or code-n-fix methods. Agile methods benefit from maintaining only a limited formalized development process, allowing the programmers to switch focus regularly. To work effectively, however, it is essential for the members of an agile programming group to remain in regular communication. With this in mind, I attempted to focus on four major goals for the team.

First, it is critical that each member of the team be operating under a consistent set of general programming principles. While there is room for personal coding styles and variance between each programmer's output, it is ultimately necessary for every programmer to be able to
quickly and easily modify others' code as necessary. Thus, the overarching structure of site
elements needed to be determined early in the project cycle, and adhered to throughout.
Functions should be easy to locate, named intuitively, and structured similarly. Database
structures should remain consistent as they develop. A developer should never have to ask
where required information is - they should already know intuitively by how the system is laid
out. See Appendix B for an outline of our site structure.

Second, it is critical for developers to know what is going on in other aspects of the
project. To this end, our functions went through a strict vetting process, for which each member
of the team was required to read through and understand every function before it could be added
to the official API. While an easy means of referencing functions was still necessary (see
below), this step gave the developers a basic idea of what features were available to them from
the other programmers. Additionally, what each developer was working on was frequently
mentioned, especially if the functionality would affect the development of other portions of code.

Third, developers should be quickly able to assess the characteristics of program
segments they are not personally involved with by means of adequate documentation. In the
group's specific case, we adopted a very specific documentation method for our vetted functions
that allowed one to easily discern what outputs the function would provide, and what inputs were
necessary to obtain the desired output. In this manner, the actual code of the function was
irrelevant - so long as a given function worked as described, any of our developers could interact
with it easily and with trust. See Appendix C for an example of our documentation.

Fourth, in any collaborative exercise it is imperative that proper versioning control be
maintained, lest someone's work be overwritten, reverted, or otherwise lost. Since most of the
coding was occurring in Adobe Dreamweaver, we opted to use the built-in checkout system.
After a day explaining the system and how to use it, we had no incidents of anyone losing work due to synchronization issues.

With these four goals in place, development proceeded fairly smoothly, with new principles of programming and database management explained as developers encountered them. Our vetting process helped this considerably - each developer was required to explain his own code. If Kelly wrote a function, she would have to explain to each programmer looking over her work what was going on and why, as well as answer any questions they might have. This ensured, once her functions were added to the API, that she understood them inside and out, and that every other developer had at least a passing understanding of what she had written. In this way, the developers were able to build off each others' success, and produce effective, reusable code.

Unfortunately, the need to teach many programming principles over the course of a semester whilst also producing useful code slowed down production significantly, and we were not able to complete the programming of the entire UGL core site by the end of the semester. With the improvements the development team has made in that time, however, I am confident that we will be even more effective next semester, as we finish up core site development and move on to game module programming.
Bibliography


Levans, Kelly, Tim McDonald and Aaron Howard. Semester Closing Interview Brian M. Bauman. 26 April 2010.


Appendix A - Design Iterations

UGL Website

LOGO

Welcome Message...

Register/Login

Newsletter

Forum Roster Rules Etc...

Figure 1 - Initial Design - Home Page
Figure 3 - Second Design - Home Page
Announcements

4-Square
There are currently no announcements for this event.

DeathMagus @ 2010-03-03 02:31:17.

Assassins
There are currently no announcements for this event.

DeathMagus @ 2010-04-03 02:31:17.

Capture the Flag
There are currently no announcements for this event.

DeathMagus @ 2010-03-03 02:31:17.

Dagorhir
There are currently no announcements for this event.

DeathMagus @ 2010-03-03 02:31:17.

Humans vs. Zombies
There are currently no announcements.
Appendix B - Site Structure

As every website does, ours begins with a root source folder. Within that folder are 4 key folders: *functions*, *games*, *images*, and *style*. *Functions* contains the top-level PHP include files that are necessary to run the site core. These are made up of *inc_default.php*, *inc_games.php*, *inc_sql.php*, *inc_structural.php*, and *inc_user.php*. *Inc_default.php* is the base include - every page on the entire site can draw from the functions contained inside. This file also controls additional includes, and allows pages to access more specialized functions, if necessary.

*Inc_games.php* contains game-related functions that are utilized by multiple games. Items such as code name generators and game data processors are included in this file. *Inc_sql.php* is the means of accessing our databases. The functions inside allow programmers to quickly and easily query the database for information, send it updates, and more. *Inc_structural.php* contains functions that build portions of the site such as tables, announcements, and navigation. Centralizing these build tasks allows us to propagate site-wide changes very quickly. Finally, *inc_user.php* contains all the functions that manipulate user accounts on the website. This includes registration, profile control, and password reset functions.

The games folder is the attachment point for every game module we build. Each game is self-contained, with all the HTML, CSS, and PHP necessary to function (assuming it's attached to the site core). This is accomplished through the use of *inc_default.php*, which checks which folder the requesting page is in, and automatically includes all necessary extensions. A similar process occurs with the CSS, allowing the site core to expand with the modules to allow extensive customization, without allowing undue access to unneeded components of the system.

The other two folders, *images* and *style*, are simply repositories for all the images and core CSS used on the site.
Appendix C - Documentation

Since it is critical for our developers to readily use each others' work in their own code, it was necessary for each developer to provide clear and concise documentation of his functions.

For sake of example, the following is the core authorization function - by Tim McDonald:

```php
function authorize($id, $clearance){
    //Checks if $id's ugl_level is greater than $clearance, if ugl_status equals $clearance, or if ugl_status equals a value in $clearance[]
    //Returns 1 if access is granted and 0 if access is denied
    if (is_array($clearance)){
        //if yes iterate array through Authorize($id, $clearance[] )
        foreach($clearance as $userStatus){
            $tally += authorize($id, $userStatus);
        }
    }
    return $tally;

    //Check if $clearance is a string
    }else if (is_string($clearance)){
        $sql = "SELECT ugl_status
                FROM ugl_users
                WHERE ugl_id = '$id'
                LIMIT 1";
        $userData = mysql_fetch_array(uglifyQuery($sql));
        if($userData['ugl_status'] == $clearance){
            return 1;
        }else{
            return 0;
        }

    //Check if $clearance is a number
    }else if(is_numeric($clearance)){
        if($id ==0){
            return 0;
        } else{
            $sql = "SELECT ugl_level
                FROM ugl_users
                WHERE ugl_id = '$id'
                LIMIT 1";
            $userData = mysql_fetch_array(uglifyQuery($sql));
            // if number is less than or equal to $clearance allow access
            if($userData['ugl_level'] <= $clearance){
                return 1;
            }else{
                return 0;
            }
        }
    }else{
        //If nothing matches the page dies
        die('Error: Illegal authorization function');
    }
}
The first form of documentation we make use of is ample commenting. Comment lines are denoted by the double slashes in front, and they represent notes to other programmers (or ourselves) about what precisely the code is doing on the line below the comment. For instance, line 4 (//$clearance == array?) is a comment asking if clearance is an array. Sure enough, the line below is code to check if the variable $clearance is an array: if(is_array($clearance)){}. In this manner, it is possible to follow the programming of the site loosely without any knowledge of programming whatsoever.

Far more useful, however, is the holistic documentation located at the top of the function. Our top-level documentation consists of three parts - a description of the function, a description of the results, and the current status of the vetting process. The description of the function explains in straightforward terms what each function does, as well as the information it uses to operate. In our example, the function "Checks if $id's ugl_level is greater than $clearance, if ugl_status equals $clearance, or if ugl_status equals a value in $clearance[]". The use of variable names in the comments serves as a quick orientation to the function, so the programmer can easily see what information needs to be sent to the function and what the information needs to be labeled as. A programmer seeing the above line knows that they need to send the function the ID number of a user, and a clearance value that is either a number, a string of characters, or a list of character strings.

The second comment line demonstrates the sort of value that will be returned from the function if it completes. In our example, the function will return the number "1" if the user is allowed to view the page, and a "0" if the user is not. This line, combined with the one before it, is a very powerful tool. Together, the comment lines tell a programmer what he needs to send to the function, and what he will get back. This means that (so long as the function works
correctly) the programmer doesn't even need to read the rest of the function to use it effectively.

He would be just as capable if this was all he saw:

```c
//Checks if $id's ugl_level is greater than $clearance, if ugl_status equals
//clearance, or if ugl_status equals a value in $clearance[
//Returns 1 if access is granted and 0 if access is denied
function authorize($id, $clearance){
    ...
}
```

Thus, effective documentation makes code simpler to follow, and easier to use.

The third part of the top-level documentation consists of each developer's initials, indicating that he has read the function and understands it. Once every developer has signed off on the code, the signatures are removed, and the function is added to the official API.
Appendix D - Developer Interviews

At the close of the semester, I interviewed each of the developers about their experiences on the team. Some were verbose, and others terse - all had valuable insights into the strengths of the process we used and how it might be improved.

Aaron Howard:
Do you think the group programming approach we utilized was effective?
A.H: I believe so. Everyone was well updated and received help when needed.

What was your favorite part of it?
A.H: I had fun designing the buttons. I was allowed to let my creativity wander.

What was the biggest weakness of the approach we used?
A.H: I felt the group was a bit undersized. At times we would spend over 3 hours a week.

What aspect of the user interface we designed is your favorite?
A.H: I like how the design is fairly simple.

What is the weakest aspect of the user interface?
A.H: It could be a bit more user friendly. Perhaps a mini-tutorial for new users.

If you could change one aspect of the development process, what would it be?
A.H: Perhaps recruit a few more people into the development or a make a schedule.

If you could change one aspect of the user interface, what would it be?
A.H: As I said before, a mini-tutorial

Kelly Levans:
Do you think the group programming approach we utilized was effective?
K.L: For the most part, yes. We all learned and worked at different paces, so it was good for us to work on separate stuff with Brian helping us each as we needed it. It was also probably fairly efficient for the website to be working on a number of things at once. However, Brian did say that we should have had more group programming, and working together a bit more occasionally might have been a good thing, pooling knowledge and all that. But it is kind of hard for three people to work on one thing together when you can really only do it at one computer. The brainstorming we did at the beginning for what we wanted the site to have, with Brian writing stuff down on the board, was a really good approach. One thing I would have liked - in the beginning, Brian had us each play around with html and CSS and bring in our work as little assignments. While this is a good idea for an intro, I think it might have been more effective if Brian had given us a worksheet, so to speak, specifying a few things he definitely wanted us to
know and work with. That way we would have had some pointers/ideas of things to play with, rather than just being tossed into a sea of code and told to choose what we felt like.

What was your favorite part of it?
K.L.: As much time as I spent trying to wrap my brain around stuff that seemed like quantum physics to me, and as much as I probably drove Brian up the wall demanding to know where the returned data that we didn't need might be displayed, I kind of liked mucking about with the function codes, trying to understand the different commands and how they all worked together in the (to me) shadowy world of the virtual. There are still a kajillion things that I could learn, and I have trouble remembering the stuff I do know, but I feel like I've gain a bit of insight into this vast sea of code and learned a little bit of how computers talk to each other. I also liked working with the HTML and CSS near the end, with a little PHP, because that I understood more. But the functions were fun.

What was the biggest weakness of the approach we used?
K.L.: While our working on separate pieces was efficient, I guess it would have been nice if we'd have had a few like, discussions every week or so about what we were learning and working on. We did talk a little bit about our work, but not really what we were doing specifically. I think discussing what we were learning about code and bringing up things that the others might not be learning yet because they were working on different stuff would have been helpful in general as to knowing about various aspects of coding. Then I could be like, "Hey, I don't understand the stuff Colton's using. How does it work, and how does it work differently than what I'm working with?"

What aspect of the user interface we designed is your favorite?
K.L.: I think probably one of my favorite parts is the password reset function. I worked on that a lot - with Brian basically writing half the code over my shoulder - and it was a good learning experience. I felt kind of accomplished when this huge chunk of code was finally written, and I knew it was a pretty useful piece. Now if somebody forgets their password, they can easily fix it themselves with a few clicks of a button - instead of having to email the webmaster and wait until he reads his email to be able to get back onto the site.

What is the weakest aspect of the user interface?
K.L.: Hmm... We've got a pretty good thing going here, and we're trying to make it easily accessible so the new users don't look at the site and go "How the **** does this work?". I guess I would be a little concerned about having just horizontal nav on the main page, and then bringing in a left nav for the other pages without the buttons for the separate games staying around. That might be a little bit confusing to the new users at first, but since there will be a "games" link on the left nav, they'll probably figure it out pretty quickly.

If you could change one aspect of the development process, what would it be?
K.L.: I guess I would like those update meetings I talked about in my answer to question 3. Like, every Monday we could just have a ten minute discussion of the code we were using and what we'd learned about it, and then everyone could know what one person had learned. Then we'd all know it.
If you could change one aspect of the user interface, what would it be?
K.L: I like it, but I think I would just leave the game buttons up in the horizontal div all the time. That would be simple to me, but it might take up a lot of room, so maybe it's for the best.

Tim McDonald:
Do you think the group programming approach we utilized was effective?
T.M: The group approach was effective. We were able to [do] a lot of work.

What was your favorite part of it?
T.M: My favorite part was designing the CSS for the site.

What was the biggest weakness of the approach we used?
T.M: Not everyone knows every aspect of the site.

What aspect of the user interface we designed is your favorite?
T.M: The security access

What is the weakest aspect of the user interface?
T.M: The stuff we have not done yet

If you could change one aspect of the development process, what would it be?
T.M: Have two people working on the same project at the same time.

If you could change one aspect of the user interface, what would it be?
T.M: (No response)