Software Development Life cycle
End of project report

Author:	Tom Reed, Matt Whitmore, Dave Clark, Silhab Csoma, Mike Steel, Chris 'Tux' Lloyd, Aleksandra Badyda, Samuel Jackson, Chris Marriott
Config. Ref.:	SE.17.DS.01
Date:	2013-02-07
Version:	1
Status:	draft

Department of Computer Science,
Aberystwyth University,
Aberystwyth,
Ceredigion, SY23 3DB,
U.K.

CONTENTS

1 Introduction

1.1 Purpose of this Document

This is the user interface specification. This document is intended to show how the system will look and feel. It will also show the risk assessment and the Gantt chart.

1.2 Scope

This document will show how the user will be able to interact with the system through use case diagrams. How the system will look through user interface design. It will also list the risks that could possibly arise and what people in the groups primary and secondary roles are. This will be shown in the risk assessment. There will also be a example Gantt chart that will list people in the group, their task and time frame for which it is intended to be completed.

1.3 Objectives

The objective of this document is to design the interface and usability of the system for the monster mash game.
The areas covered by this plan are:

- Overview of system
- Use Cases
- User Interface mock ups
- Gantt Chart
- Risk Analysis

2 Overview

This project requires the group to be able to design and create a web based application that allows the users to socially interact with one another. Monster Mash (The name of the social web application) will allow the user to create an on-line profile and once this is completed, the application will give each new user their own monster, of which they can use to challenge other users of the application to a battle, and depending on the outcome (win or lose) will award you with points and will alter the statistics of your monster. To commence a battle with another user, its required that they are both "Friends" with one other on the web application, like many other social networking sites, this will be done by one user sending a friend request, and the other either accepting or rejecting. Aside from battling each others monsters, the game should be able to include other features such as breeding, so that users can create stronger monsters, and also a high scores page, on which friends can check to see who has the best overall score out of each of them.

3 Deployment

3.1 Overview

We have decided to use a small enterprise style system, which will be hosted on Chris Lloyd's VPS (virtual private server). While this would normally not be hosted on a single server (due to security), it does allow almost all of the benefits such as it being live(non-local), 99.9 percent uptime and self-managed hosting. It also gives us some experience with how to deal with real world solutions that we may come across in our future lines of work.

3.2 Hosting

The hosting has been sorted by Chris Lloyd; he has allowed the group to use his VPS which he has installed Apache http server, MySQL database and a tomcat. Chris Lloyd has also purchased the domain monstermashgame.co.uk to which he has added an A-record to the VPSs IP.

3.3 Apache Tomcat HTTP

Apache Tomcat is used to run the Java environment in a public accessible zone. The servlets will handle the HTTP requests from the Java environment, which sit on Apache tomcat. This will mainly involve sending HTML pages.

3.4 MySQL

MySQL, while not being used at the higher end of the corporate market (held by Oracle), it does allow us to use an enterprise like database system, as it is quite commonly used for small to medium sized business and is often used on small scale web hosting.

3.5 TomCat

Tomcat as a project, is developed by the Apache foundation. Tomcat is widely deployed as a Java application server, all though through the various configurations it can be used to make a mass distributed Java cluster. This is to say that it is possible to run a single Java application across a distributed processing cluster.

3.6 Server Side Conclusion

While there is other software out there that may do a better job, we found that by sticking to the widely used software, implementation and application, we would find a good, well rounded base knowledge in the area of server side web applications.

4 Methodology

Although the structure of the hand in documents leads us to waterfall, we shall aim to do incremental within this. (See fig1)
The advantages are:

- Easier management
- Can get user feedback earlier
- Can avoid crises by be alerted to problems earlier
- Can respond easier to changing user requirements
- Earlier exchange of functionality

The disadvantages are:

- Can be harder to manage, with more steps and crucial decencies and overall progress monitoring
- Version and configuration control is crucial and can be complex

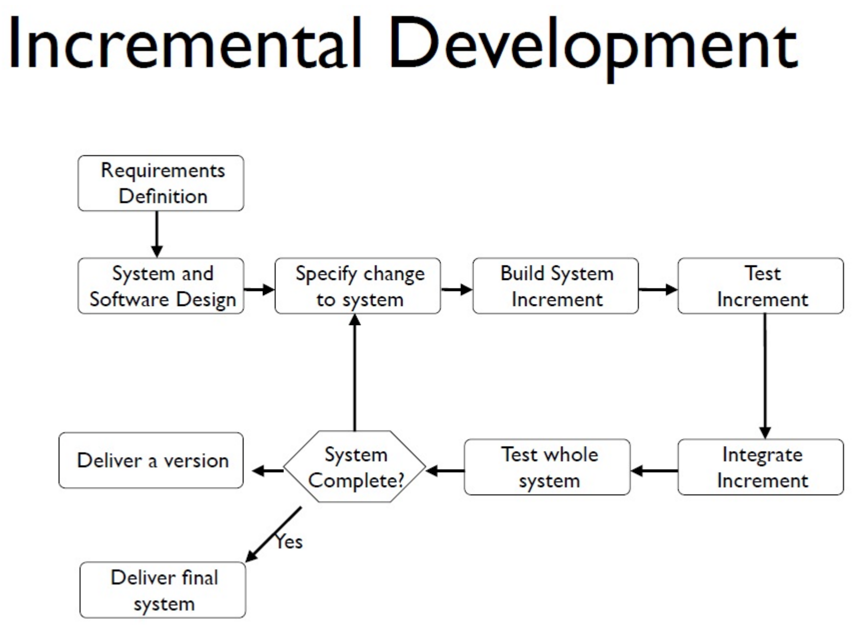
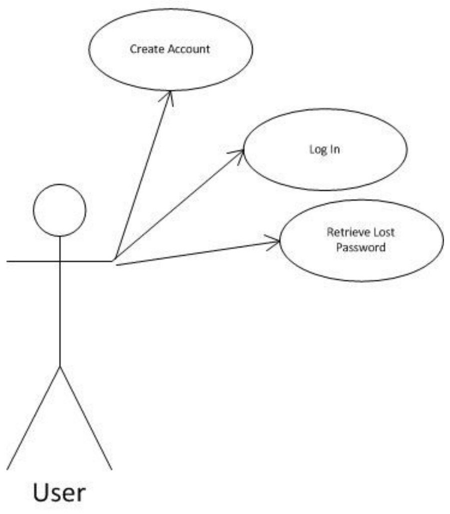

Incremental Development
Requirements Definition
System and Software Design
Specify change to system
Build System Increment
Test Increment
Integrate Increment
Test whole system
System Complete?
Deliver a version
Yes
Deliver final system

Fig1

5 Use Case

This section will highlight the functional requirements that were defined in the requirement pdf.

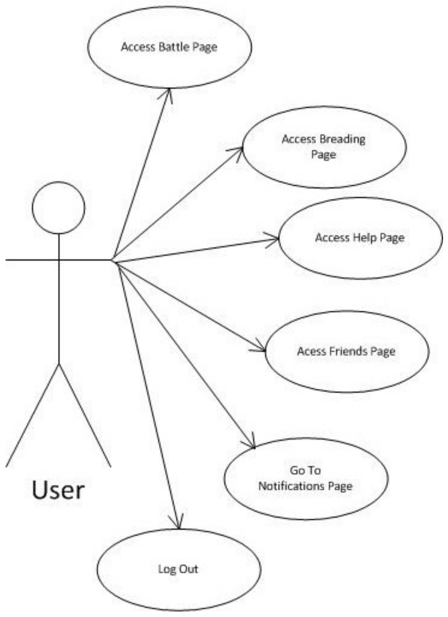
5.1 Welcome Page



Functional Requirements- FR1 - Server-based Authentication FR6 - Client Options FR7 - Start-up of software in browser

This is the screen you first see when accessing the website
Example of usuage: You create an account, you then log in using your details.

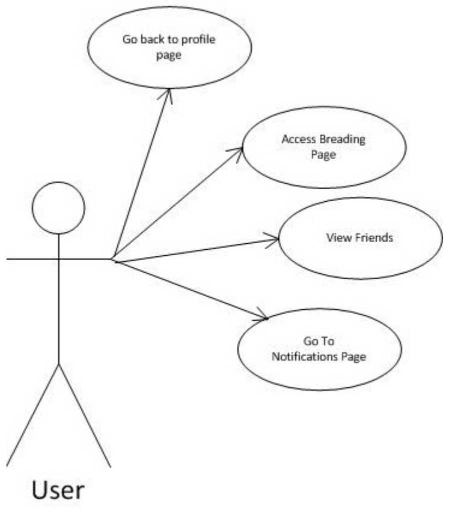
5.2 Main profile page



Functional Requirements- FR8 - Game Display In Browser FR10 - Fight Notifications

From this view you can access all the different pages
Example usuage: You navigate to your friends page

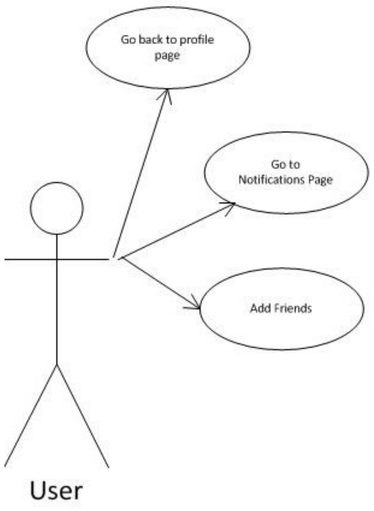
5.3 Friends Page



Functional Requirements- FR2 - Server Friends List FR5 - Server-server communication FR6 - Client Options FR9 - Friend Matching FR11- Friends Rich List

From here you can view your friends or go back navigate to other pages
Example usuage: You view your friends you then decide to navigate to your notifications page

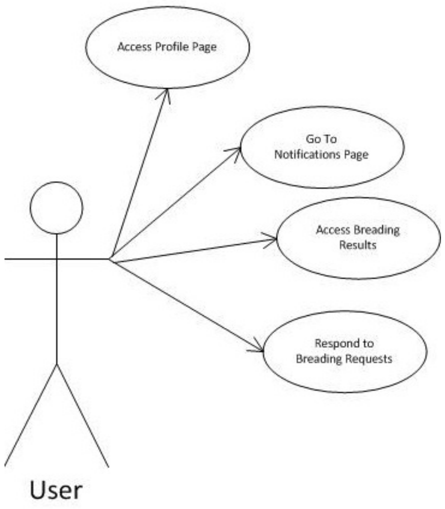
5.4 Add Friends



Functional Requirements- FR5 - Server-server communication FR6 - Client Options FR9 - Friend Matching

From here you can add friends
Example usuage: You add a friend by entering their email address they use with their account.

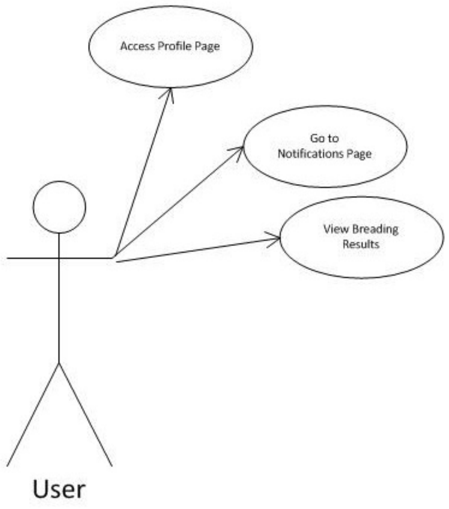
5.5 Breeding



Functional Requirements- FR3 - Server Monster List FR5 - Server-server communication FR6 - Client Options FR8 - Game Display In Browser

From here you can see navigate to the results of any breeding you have done or respond to any breeding requests you might have.
Example request: Yo see a friend has requested a monster to breed with you, you accept.

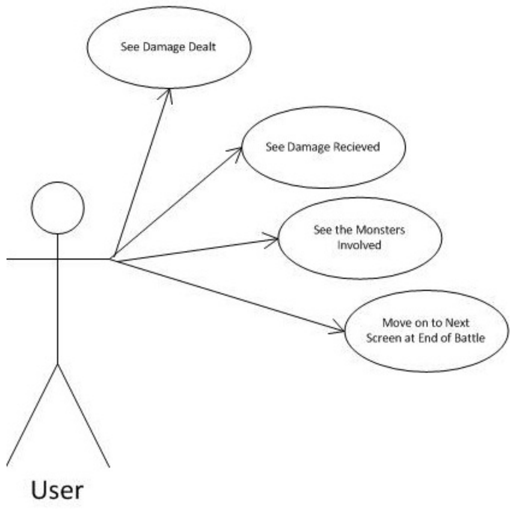
5.6 Breeding Results



Functional Requirements- FR5 - Server-server communication FR6 - Client Options FR8 - Game Display In Browser

From here you view breeding results
Example usuage: you have recently responded to an request you check back to see the results

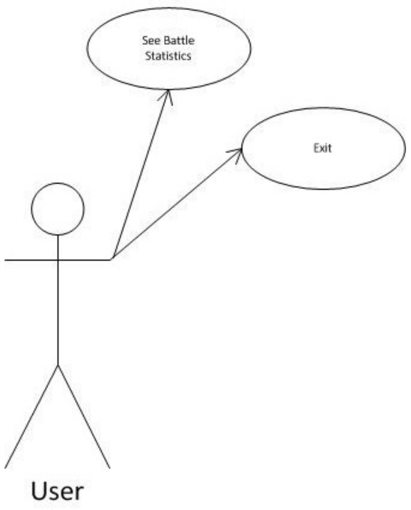
5.7 Battle Screen


See Damage Dealt
See Damage Recieved
See the Monsters Involved
Move on to Next Screen at End of Battle
User

Functional Requirements- FR4 - Server Monster mash management FR5 - Server-server communication FR6 - Client Options FR8 - Game Display In Browser

From here you view information about a battle taking place
Example usuage: you have accepted a battle and now want to see how much damage was dealt.

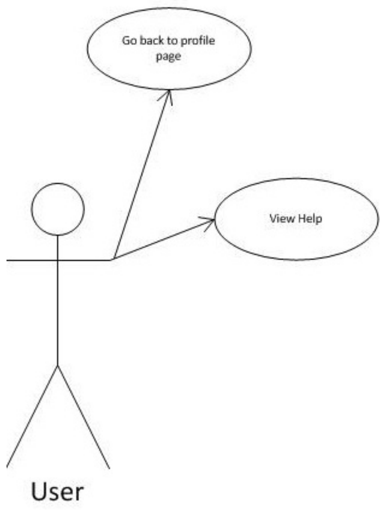
5.8 Battle Results Page



Functional Requirements- FR3 - Server Monster List FR8 - Game display in browser FR10 - Fight Notifications

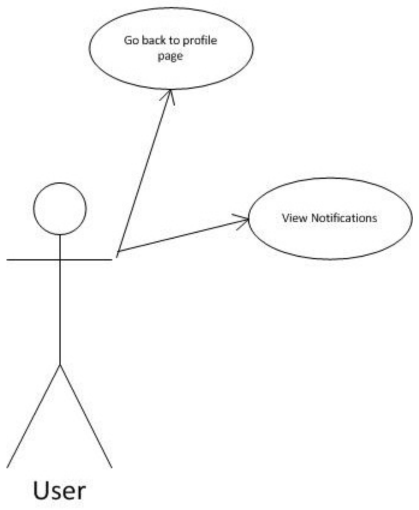
From here you can see all the stats from the battle
Example usuage: you have a battle and wants to see the stats

5.9 Help Page



From here you can view help and see how to play the game
Example usuage: you want to know how breeding works so you navigate to the help page to find out

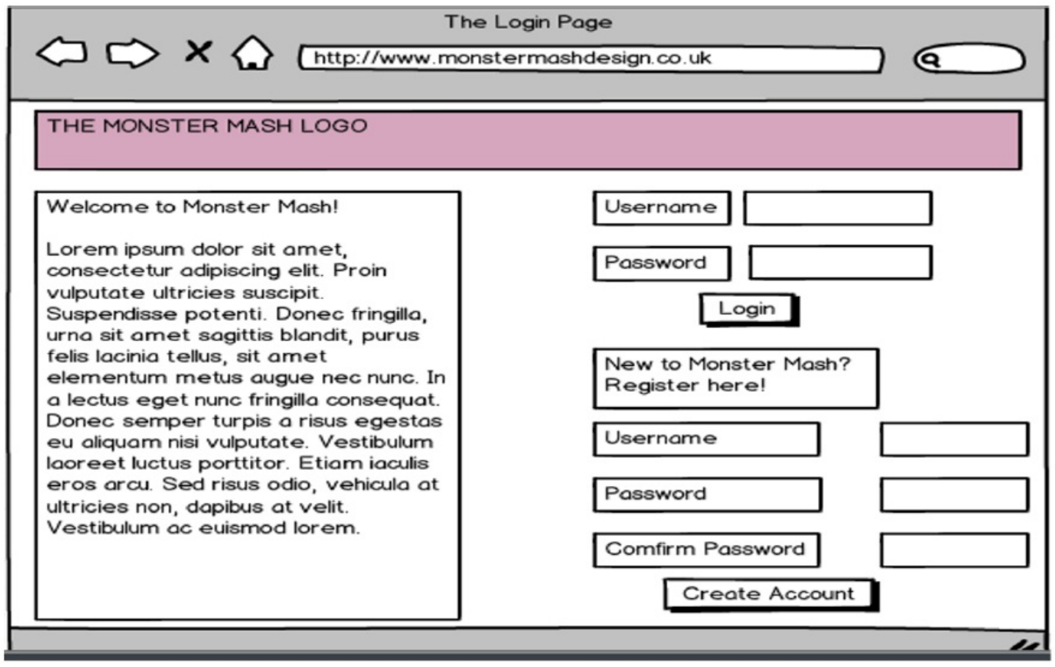
5.10 Notifications Page



From here you can view all notifications you have
Example usuage: you want to see your notifications as your friend sent you a battle requet

6 User Interface Design

6.1 Login Screen(fig2)

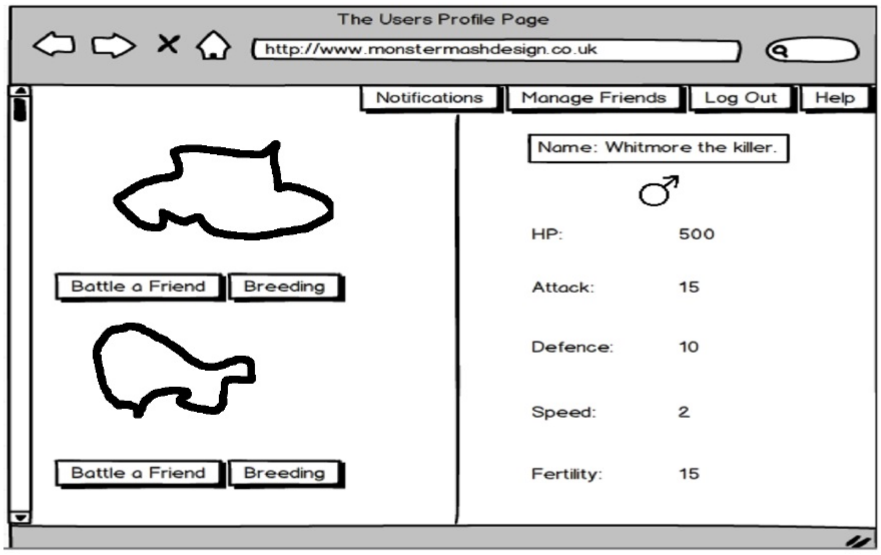


Welcome to Monster Mash!

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin vulputate ultricies suscipit. Suspendisse potenti. Donec fringilla, urna sit amet sagittis blandit, purus felis lacinia tellus, sit amet elementum metus augue nec nunc. In a lectus eget nunc fringilla consequat. Donec semper turpis a risus egestas eu aliquam nisi vulputate. Vestibulum laoreet luctus porttitor. Etiam iaculis eros arcu. Sed risus odio, vehicula at ultricies non, dapibus at velit. Vestibulum ac euismod lorem.

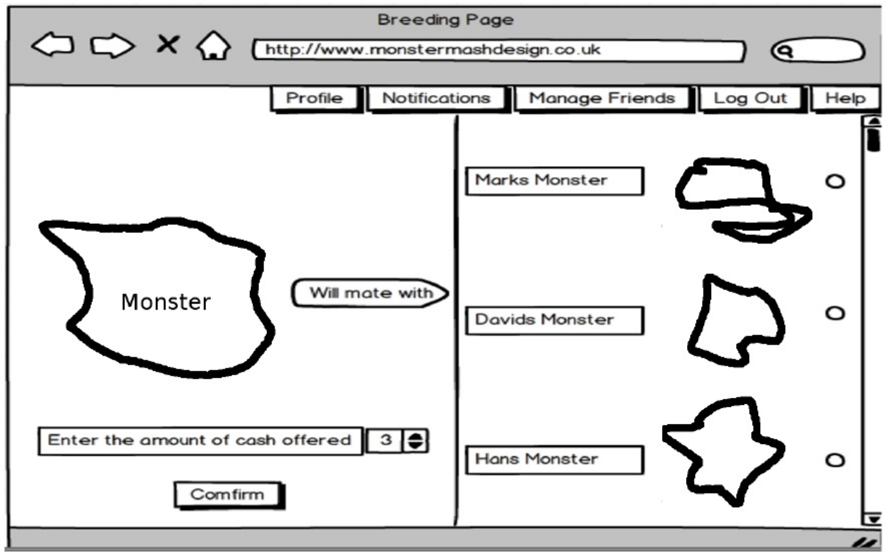
This is the basic design of the page the user will first encounter when they want to either create an account or log in if they already have an account. Both the Login button and the Create Account button (if filled in correctly) will take the user to their profile page. This is the only page in which the user is not signed into their account. Once logged in, the user will be directed to Figure 3.

6.2 Profile page(fig3)



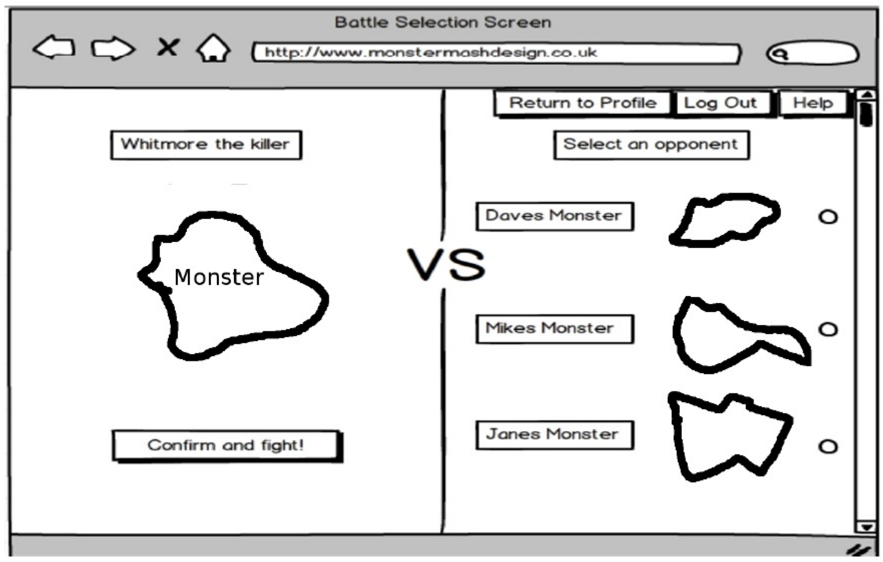
This is the basic design for the profile page of our project, this will be the page the login screen takes you each time you log in. It is from here, you (the user) will be able to select each of your monsters to either battle or breed with a friends monster. There are other features that are accessible from this page, such as the Help page and the Manage Friends page, which can be seen in the top right corner of the screen, however these are also accessible from other pages aside from the Profile page, so we will just focus on the Breeding and Battle a Friend page. On the right hand side of the page, the stats for the monster you have selected will be shown, this will be done by clicking or possibly hovering over the monster with the mouse. You select which monster you want to breed by clicking on the corresponding breeding button. Located underneath the monster you wish to select. Once you have selected your monster you wish to breed. You are taken to figure 4.

6.3 Breeding Page(fig 4)



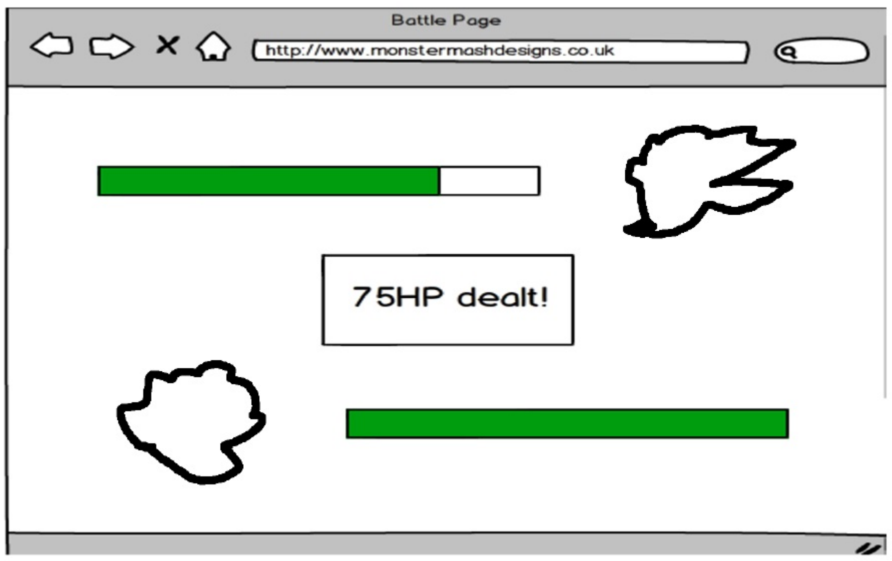
The monster that you selected for breeding from the previous page will be shown on the left hand side of the screen, on the right hand side of the screen are all the monsters that your friends own, the idea here being that you have to select one of your friends monsters to breed with the monster you have, seeing that you are the one who benefits from the breeding process (the one being requested to be bred with does not gain a monster) the user must enter an amount of points, or cash into the box located on the left hand side of the page (the one with the number 3 in it). It is then up to the other player to either accept or reject this offer, if it is accepted, the breeding takes place and the points are transacted from one users account into the others. The player who initiated the breeding process then gains a monster. Returning to Figure 3, if you want to select a monster to battle another friends monster, as with breeding, simply select the corresponding button underneath the monster of your choice, you will then be taken to Figure 5.

6.4 Battle Selection Screen(fig 5)



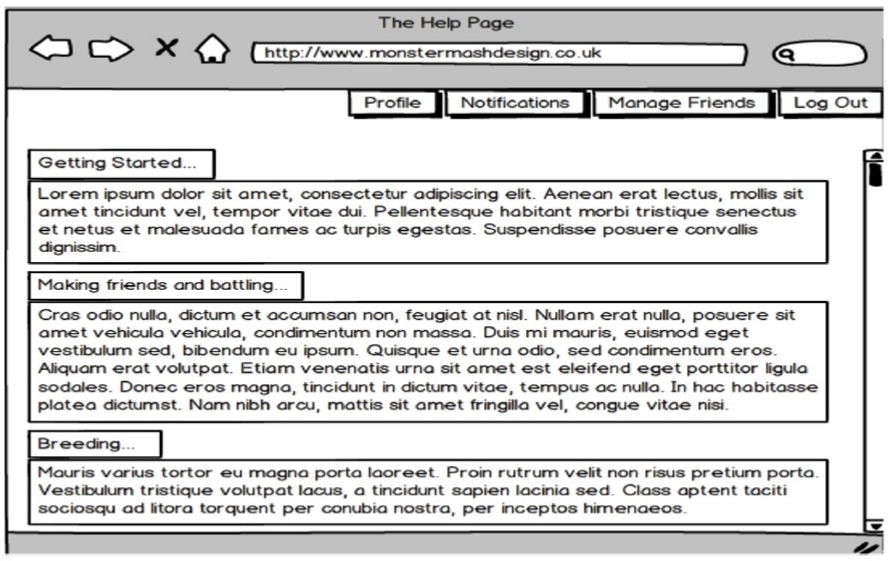
Similar to the breeding page, this requires the user to select one of their friends monsters to challenge, once they have selected, the user simply presses the confirm and fight button, and the request is sent. Both pending battle and breeding requests can be viewed in the Manage Friends page. Incoming requests will appear on your notifications.

6.5 Battle Page(fig 6)



It should be noted that this page will only be viewed by the user accepting the battle request. Once the user has accepted a battle request off their friend, they will be redirected to this page. This is an example of a battle taking place between two monsters. The user (so the one who accepted the battle request) can watch the battle unfold. The text in the middle constantly updates, telling the user how much damage each monster deals to each other. The green bar next to each of the monsters is their health bar, this gradually decreases as the battle progresses. The first monster to lose all of its life it declared the loser. After the battle, the user is directed back to the Profile page (Figure 3), the other user that wasn't present for the fight receives a message in their notifications telling them how the fight unfolded. Moving back to Figure 3, if the Help button, located in the top right hand corner is selected, Figure 7 should appear.

6.6 The Help Page(fig 7)



Getting Started...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean erat lectus, mollis sit amet tincidunt vel, tempor vitae dui. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Suspendisse posuere convallis dignissim.

Making friends and battling...

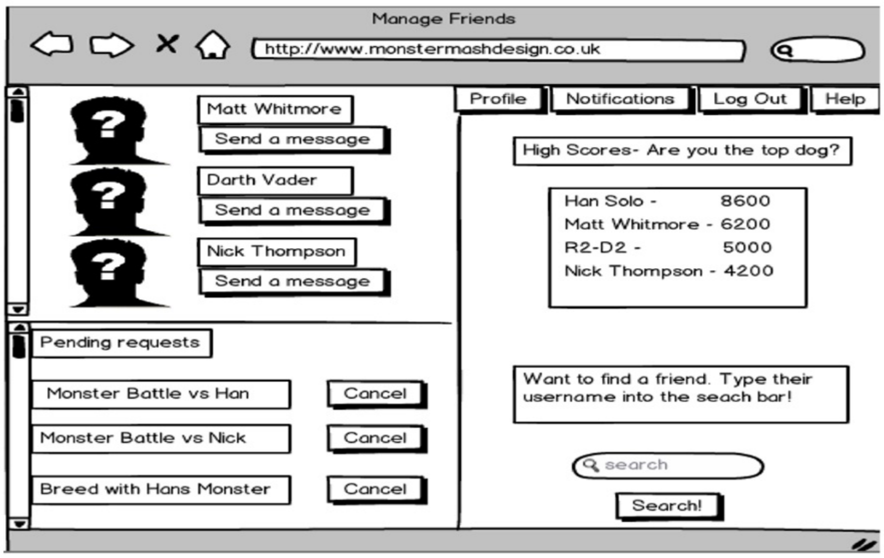
Cras odio nulla, dictum et accumsan non, feugiat at nisl. Nullam erat nulla, posuere sit amet vehicula vehicula, condimentum non massa. Duis mi mauris, euismod eget vestibulum sed, bibendum eu ipsum. Quisque et urna odio, sed condimentum eros. Aliquam erat volutpat. Etiam venenatis urna sit amet est eleifend eget porttitor ligula sodales. Donec eros magna, tincidunt in dictum vitae, tempus ac nulla. In hac habitasse platea dictumst. Nam nibh arcu, mattis sit amet fringilla vel, congue vitae nisi.

Breeding...

Mauris varius tortor eu magna porta laoreet. Proin rutrum velit non risus pretium porta. Vestibulum tristique volutpat lacus, a tincidunt sapien lacinia sed. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.

A simple page in design and implementation. This page will tell the user all they need to know about how to user the application, including things such as how to battle, how to breed and how to add friends.

6.7 Friend Management Page(fig 8)

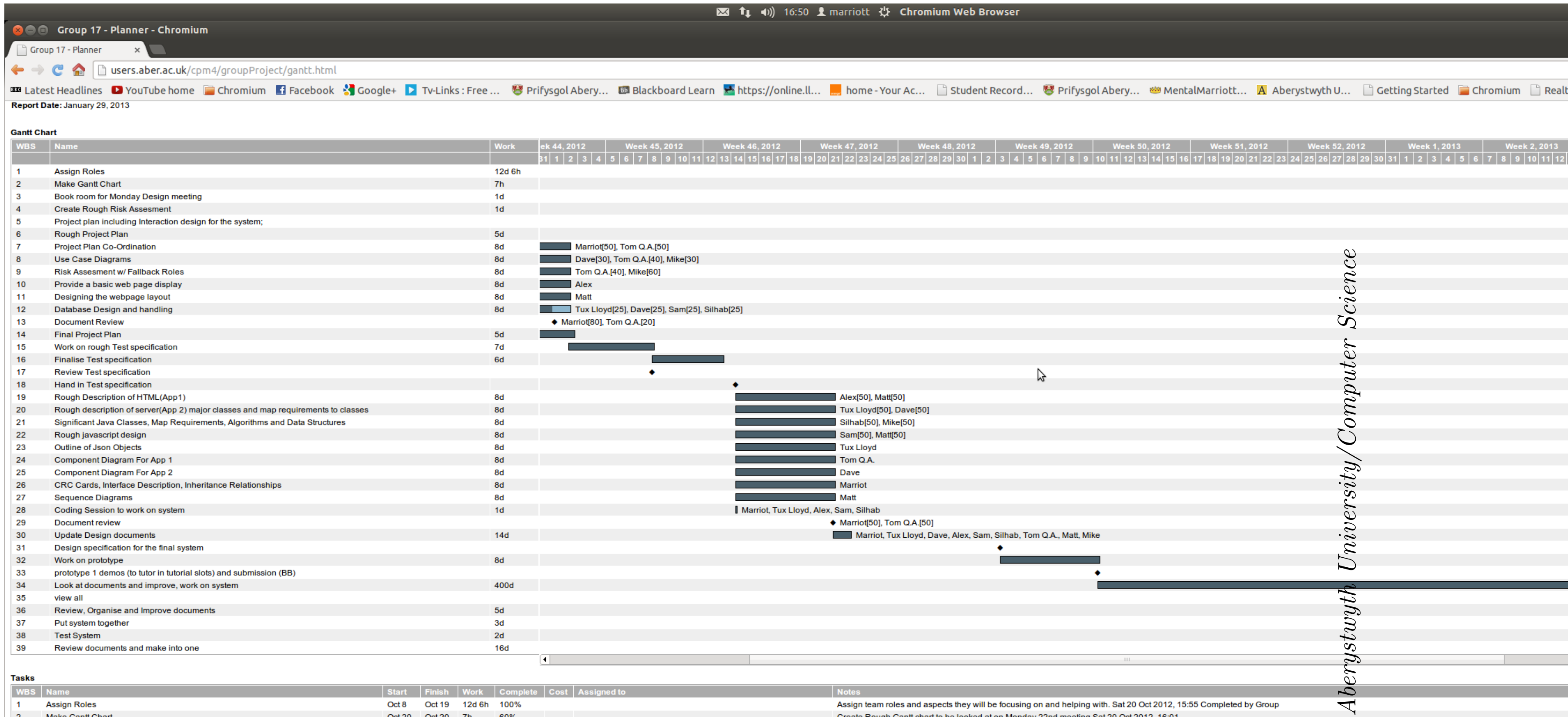


Han Solo -	8600
Matt Whitmore -	6200
R2-D2 -	5000
Nick Thompson -	4200

Want to find a friend. Type their username into the seach bar!

The Manage Friends page can be accessed from most of the web pages, the button that directs to it is located in the top right hand corner of the screen. The Manage Friends page holds many functions. From here you are able to cancel pending requests to either battle or breed with another friends monster, send personal messages to your friends, and new friends to your list and also check who has the top score overall out of all your friends. To log out of Monster Mash, simple click the Log Out button which is found on most of the pages to the top right hand corner. This will take you to back to Figure 1.

7 Gantt chart



16:50 marriott Chromium Web Browser

Group 17 - Planner - Chromium

users.aber.ac.uk/cpm4/groupProject/gantt.html

Report Date: January 29, 2013

Gantt Chart

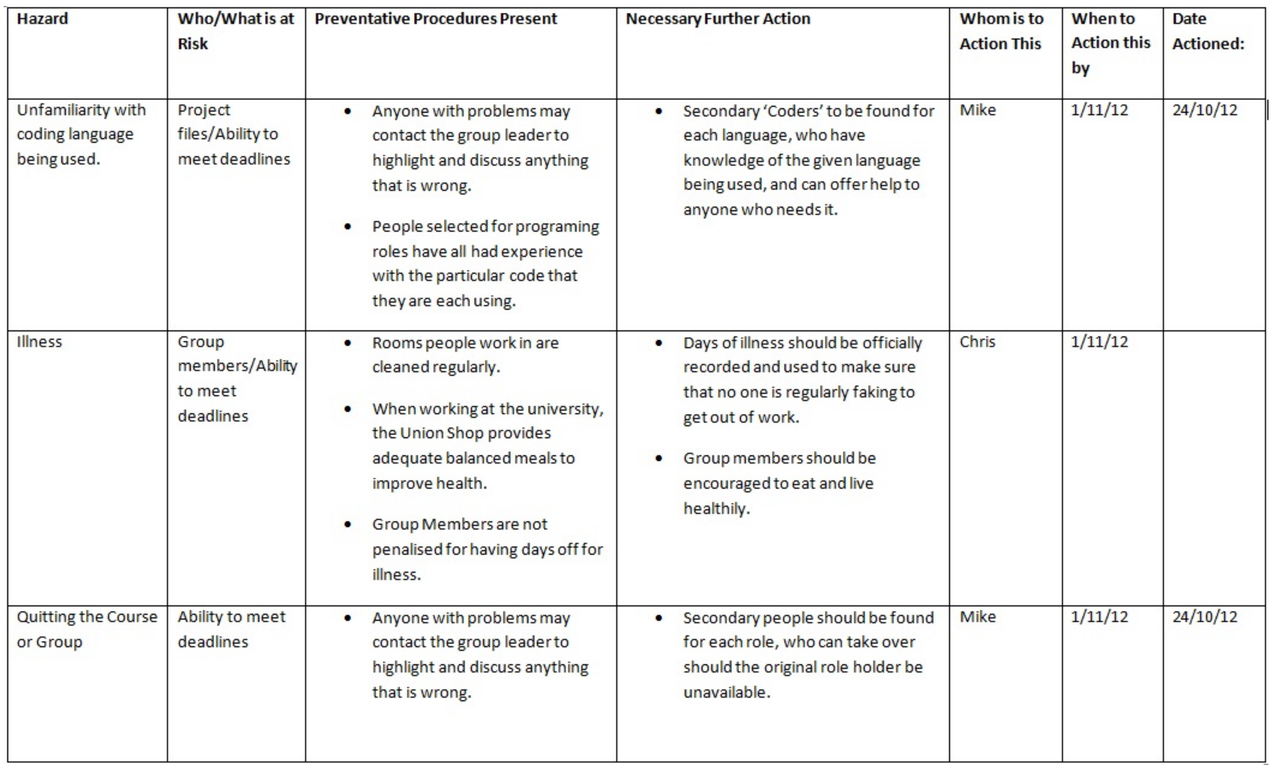
WBS	Name	Work	
1	Assign Roles	12d 6h	
2	Make Gantt Chart	7h	
3	Book room for Monday Design meeting	1d	
4	Create Rough Risk Assesment	1d	
5	Project plan including Interaction design for the system;		
6	Rough Project Plan	5d	
7	Project Plan Co-Ordination	8d	Marriot[50], Tom Q.A.[50]
8	Use Case Diagrams	8d	Dave[30], Tom Q.A.[40], Mike[30]
9	Risk Assesment w/ Fallback Roles	8d	Tom Q.A.[40], Mike[60]
10	Provide a basic web page display	8d	Alex
11	Designing the webpage layout	8d	Matt
12	Database Design and handling	8d	Tux Lloyd[25], Dave[25], Sam[25], Silhab[25]
13	Document Review		Marriot[80], Tom Q.A.[20]
14	Final Project Plan	5d	
15	Work on rough Test specification	7d	
16	Finalise Test specification	6d	
17	Review Test specification		
18	Hand in Test specification		
19	Rough Description of HTML(App1)	8d	Alex[50], Matt[50]
20	Rough description of server(App 2) major classes and map requirements to classes	8d	Tux Lloyd[50], Dave[50]
21	Significant Java Classes, Map Requirements, Algorithms and Data Structures	8d	Silhab[50], Mike[50]
22	Rough javascript design	8d	Sam[50], Matt[50]
23	Outline of Json Objects	8d	Tux Lloyd
24	Component Diagram For App 1	8d	Tom Q.A.
25	Component Diagram For App 2	8d	Dave
26	CRC Cards, Interface Description, Inheritance Relationships	8d	Marriot
27	Sequence Diagrams	8d	Matt
28	Coding Session to work on system	1d	Marriot, Tux Lloyd, Alex, Sam, Silhab
29	Document review		Marriot[50], Tom Q.A.[50]
30	Update Design documents	14d	Marriot, Tux Lloyd, Dave, Alex, Sam, Silhab, Tom Q.A., Matt, Mike
31	Design specification for the final system		
32	Work on prototype	8d	
33	prototype 1 demos (to tutor in tutorial slots) and submission (BB)		
34	Look at documents and improve, work on system	400d	
35	view all		
36	Review, Organise and Improve documents	5d	
37	Put system together	3d	
38	Test System	2d	
39	Review documents and make into one	16d	

Tasks

WBS	Name	Start	Finish	Work	Complete	Cost	Assigned to	Notes
1	Assign Roles	Oct 8	Oct 19	12d 6h	100%			Assign team roles and aspects they will be focusing on and helping with. Sat 20 Oct 2012, 15:55 Completed by Group
2	Make Gantt Chart	Oct 20	Oct 20	7h	60%			Create Rough Gantt chart to be looked at on Monday 22nd meeting Sat 20 Oct 2012, 16:01

This is the Gantt chart that is used for assigning tasks and a time period that those tasks should be finished by. The image below is merely an example as the actual Gantt chart is constantly changing.
Can be seen on-line at http://users.aber.ac.uk/cpm4/groupProject/gantt.html

8 Risk Assessment



Hazard	Who/What is at Risk	Preventative Procedures Present	Necessary Further Action	Whom is to Action This	When to Action this by	Date Actioned:
Unfamiliarity with coding language being used.	Project files/Ability to meet deadlines	• Anyone with problems may contact the group leader to highlight and discuss anything that is wrong. • People selected for programing roles have all had experience with the particular code that they are each using.	• Secondary 'Coders' to be found for each language, who have knowledge of the given language being used, and can offer help to anyone who needs it.	Mike	1/11/12	24/10/12
Illness	Group members/Ability to meet deadlines	• Rooms people work in are cleaned regularly. • When working at the university, the Union Shop provides adequate balanced meals to improve health. • Group Members are not penalised for having days off for illness.	• Days of illness should be officially recorded and used to make sure that no one is regularly faking to get out of work. • Group members should be encouraged to eat and live healthily.	Chris	1/11/12	
Quitting the Course or Group	Ability to meet deadlines	• Anyone with problems may contact the group leader to highlight and discuss anything that is wrong.	• Secondary people should be found for each role, who can take over should the original role holder be unavailable.	Mike	1/11/12	24/10/12

1 Introduction

1.1 Purpose of this Document

By producing this document we shall explain any and all tests to be carried out on our project prior to it being considered finished, all tests shall cover necessary functions of the project and allow us to discover any errors or defects before it is put into use.

1.2 Scope

This document specifies the tests that will be run to discover various bugs and put general strain on the system. It will in detail outline the tests that will be run and the required outcome as well as what will constitute a failed test. This document will also outline what will be done with the test data and how it will be reported.

1.3 Objectives

The objective of this document is to outline the tests that will be run on the system, how they will be structured and how they will be documented and reported.

2 Test Plan

2.1 Test Overview

An HTTP servlet is used to access the data contained within the backend of the server as if it was almost a web page. We will pass parameters between it and get a response. We will be working on the principle that one main servlet will perform different actions, passing through an actions variable and any data required to be processed. There are two methods of accessing this; using get and post requests. The get request will pass the parameters through the URL, the post request through a hidden layer, based on the users input. JavaScript will collate the necessary data, and attach the appropriate action command before sending to the server. All coders will be responsible for testing their code, code will be tested using a set of JUnit tests. As the code has not been written yet it is impossible to stipulate at this time what the tests will need to be. HTML and CSS will be tested using the W3C validators as well as tested in various browsers that support HTML5. For the system testing, we need to test it through the client side as there is no real way access the server side of the system without using JUnit testing which has already been stipulated. So the only way to approach it is to test if the server side does as we expect from the client side.

2.2 Test Specification

Test Ref	**Req. Being Tested**	**Test Content**	**Input**	**Output**	**Pass Criteria**
SE-N17-001	FR1, FR7	Check that a valid user can login to the system.	Input on login page for existing user. User-name: slj11@aber.ac.uk Valid password for this user: mypassword submit button pressed	User authenticated onto the system and redirected to their profile page. New session for this user started on server.	User is authenticated onto the system.
SE-N17-002	FR1, FR7	Check user names that are not emails are not admitted to the system.	Input on login page for existing user. User-name: slj11 Valid password: mypassword	Error message warns user that email isn't valid.	User is not authenticated. Error message shown.
SE-N17-003	FR1, FR7	Check passwords of length less than 8 are too short.	Input on login page for existing user. User-name: slj11@aber.ac.uk Password: mypassw	Error message warns that password is invalid.	User is not authenticated. Error message shown.
SE-N17-004	FR1, FR7	Check that passwords of length greater than 20 are too long.	Input on login page for existing user. User-name: slj11@aber.ac.uk Password: mypasswordqwertylopwq	Error message warns that password is invalid.	User is not authenticated. Error message shown.
SE-N17-005	FR1, FR7	Check a user with a valid email, but not on the system is not authenticated.	Input on login page for existing user. User-name: xxx@aber.ac.uk password: mypassword	Error message warns that the user is not registered on the system.	User is not authenticated. Error message shown.

SE-N17-006	FR1, FR7	Check a user with a valid email but incorrect password is not authenticated.	Input on login page for existing user. User name: slj11@aber.ac.uk password: mywrongpass	Error message warns that the user entered the wrong password.	User is not authenticated. Error message shown.
SE-N17-007	FR1, FR7, FR6	Check that a new user can be registered on the system.	Input on login page for new user form. Username: newuser@aber.ac.uk password: newpassword password check: newpassword	User added to system. Given starting monster and redirected to there profile page.	User is created in the system, has a new monster created and is authenticated and directed to their profile page.
SE-N17-008	FR1, FR7, FR6	Check a new user name that is not an email is not registered	Input on login page for new user. User-name: newuser Valid password: mypassword password check: mypassword	Error message warns user that email isn't valid.	User is not created. Error message shown.
SE-N17-009	FR1, FR7, FR6	Check passwords of length less than 8 are too short.	Input on login page for new user. User-name: newuser@aber.ac.uk Password: mypassw Password check: mypassw	Error message warns that password is invalid.	User is not created. Error message shown.
SE-N17-010	FR1, FR7, FR6	Check that passwords of length greater than 20 are too long.	Input on login page for new user. User-name: newuser@aber.ac.uk Password: mypasswordqwertylopwq Password check: mypasswordqwertylopwq	Error message warns that password is invalid.	User is not created. Error message shown.
SE-N17-	FR1, FR7, FR6	Check a user with a valid	Input on login page for new user.	Error message warns that the	User is not created. Error message

011		email that is already on the system is not created	User-name: slj11@aber.ac.uk password: mypassword password check: mypassword	user is already registered with the system.	shown.
SE-N17-012	FR1, FR7, FR6	Check that if the password that has been typed matches	Input on login page for new user. User-name: newuser@aber.ac.uk Password: mypassword Confirm Password: mypassss	Error message warns that the passwords entered do not match.	User is not created. Error message shown.
SE-N17-013	FR1, FR7	Check that a user can only access parts of the site if they are registered and authenticated.	Attempt to access all other pages of the site other than the login/register page.	Should be redirected back to the login/register page.	Unregistered/Unauthenticated users are redirected back to the login/register page.
SE-N17-014	FR7	Check a user can logout from the system.	Login to the system using a valid registered users details. Then logout using the logout button.	User is directed to the login/register page.	The users session is terminated and they can no longer access secure parts of the site.
SE-N17-015	FR2, FR8	User can access their friends page from their profile.	From the profile click the friends button.	User is taken to a list of their friends.	User is shown a list of their current friends.
SE-N17-016	FR3, FR8	User is shown a list of there current monsters and their stats on the profile page.	On the profile page a list of monsters is shown. Click/highlight a monster to view it's stats.	The monsters stats are shown next to the monster.	Monsters stats are successfully shown the user.
SE-N17-017	FR4, FR8	User can view a list of notifications.	On the profile page click the notifications button.	User should be shown a list of current notifications for battle/breeding/friend requests and fight/breeding	List of battle/breeding/friend requests and fight/breeding results shown.

				results.	
SE-N17-018	FR4, FR8	User can view a list of notifications.	On the profile page click the notifications button when the user has no pending notifications.	User should be shown a “no notifications” message	No notifications message shown.
SE-N17-019	FR4, FR8	For each type of notification (battle, breeding, friend requests and fight/breeding results) check that the list is displayed correctly when the user has only one of these requests and no others.	On the profile page click the notifications button.	User should be shown a list of current notifications for battle/breeding requests and fight/breeding results.	List of battle/breeding requests and fight results shown.
SE-N17-020	FR2, FR6, FR8, FR9	Check that accepting a friend request adds the friend to the users friend list.	Accept a friend notification by clicking accept.	User should be shown a message telling them that the request has been accepted.	Message shown and friend added to the friends list.
SE-N17-021	FR2, FR6, FR8, FR9	Check that the user can send a friend request.	On the friends page, click add friend. Email: test@aber.ac.uk Click send request.	User should be notified with a message that a request has been sent.	Message shown and request should be pending on server.
SE-N17-022	FR2, FR6, FR8, FR9	Check that the user can not add a friend that doesn't exist.	On the friends page click add a friend. Email: noone@aber.ac.uk Click send a request.	User should be notified that the user doesn't exist	Message should be shown. List of friend not changed.
SE-N17-023	FR2, FR6, FR8, FR9	Check that the user can not add a friend with an email address that is not well formed.	On the friends page click add a friend. Email: noone Click send a request.	User should be notified that the email address is invalid.	Message should be shown. List of friend not changed.
SE-N17-024	FR4, FR8	Check that a user can accept a battle	Accept a battle notification by clicking accept.	User should be taken to battle page and a battle	User directed to battle page.

		request.		should start.	
SE-N17-025	FR4, FR8	Check that a user can send a battle request.	On the friends page, click on the button to send a battle request to a friends monster.	User should be notified with a message that a request has been sent.	Message shown and request should be pending on server.
SE-N17-026	FR4, FR8	Check that a user can accept a breeding request.	Accept a breeding notification by clicking accept.	User should be taken to breeding page and breeding should start.	User directed to breeding page.
SE-N17-027	FR6, FR8	Check that user can send a breeding request.	On the friends page, click on the button to send a breeding request to a friends monster.	User should be notified with a message that a request has been sent.	Message shown and request should be pending on server.
SE-N17-028	FR3, FR4, FR10	Check that battles are executed correctly.	Accept a battle request. Users should be redirected to battle page.	Battle should run. The user can then view the results of the battle. Notification should also be sent to opponent to view the outcome.	Battle shown correctly. Both users are able to view the stats of the battle.
SE-N17-029	FR3, FR4, FR6, FR8	Check that breeding monsters is executed correctly.	Accept a breeding request. User should be redirected to breeding results page	Results of breeding should be shown. A notification of the breeding should be sent to friend linking them to the results of the breeding.	Breeding stats shown correctly. Both player are able to access the outcomes of breeding. Request sender gets the children and request acceptor gets money prize.
SE-N17-030	FR3, FR8, FR10	Check that the outcome of a battle can be viewed by the request sender.	After a battle, check that the request sender gets a notification and can view the outcome of the battle.	Notification should be viewable and clicking it will show the outcome of the battle.	Outcome of the battle is viewable by the user.
SE-N17-031	FR3, FR6, FR8	Check that the outcome of breeding can be viewed by the	After a breeding event, check that the request sender gets a notification and can view the	Notification should be viewable and clicking it will	Outcome of the breeding is viewable by the

		request sender.	outcome of the breeding.	show the outcome of the breeding.	user.
SE-N17-032	FR6, FR8, FR9	Check that a friend request that has been sent can be cancelled.	Click cancel on a pending request.	Message shown that the request has been removed.	Request removed from the system and no longer displayed to either of the users.
SE-N17-033	FR6, FR8	Check that a battle request that has been sent can be cancelled.	Click cancel on a pending request.	Message shown that the request has been removed.	Request removed from the system and no longer displayed to either of the users.
SE-N17-034	FR6, FR8	Check that a breeding request that has been sent can be cancelled.	Click cancel on a pending request.	Message shown that the request has been removed.	Request removed from the system and no longer displayed to either of the users.
SE-N17-035	FR8, FR9	Check that a friend request can be rejected.	Click reject on a request	Message shows that the request has been rejected.	The request should be removed from the system, the other user should be informed that their request was rejected.
SE-N17-036	FR8	Check that a battle request can be rejected.	Click reject on a request	Message shows that the request has been rejected.	The request should be removed from the system, the other user should be informed that their request was rejected.
SE-N17-037	FR8	Check that a breeding request can be rejected.	Click reject on a request	Message shows that the request has been rejected.	The request should be removed from the system, the other user should be informed that their request was rejected.
SE-N17-037	FR11	Check that a rich list of the players friends can be viewed.	Go to the friends page.	A list of the richest friends should be showed	Rich list is shown and is accurate.
SE-N17-	FR2, FR6,	Check that accepting a	Accept a friend notification by clicking	User should be shown a message	Message shown and friend added to the

038	FR8, FR9, FR5	friend request from a friend on another server adds the friend to the users friend list.	accept.	telling them that the request has been accepted.	friends list.
SE-N17-039	FR2, FR6, FR8, FR9, FR5	Check that the user can send a friend request to a friend on another server.	On the friends page, click add friend. Enter email of friend on another server Click send request.	User should be notified with a message that a request has been sent.	Message shown and request should be pending on server.
SE-N17-040	FR2, FR6, FR8, FR9, FR5	Check that the user can not add a friend on another server that doesn't exist.	On the friends page click add a friend. Enter email of a friend that does not exist on another server. Click send a request.	User should be notified that the user doesn't exist	Message should be shown. List of friend not changed.
SE-N17-041	FR2, FR6, FR8, FR9, FR5	Check that the user can not add a friend with an email address that is not well formed.	On the friends page click add a friend. Email: noone Click send a request.	User should be notified that the email address is invalid.	Message should be shown. List of friend not changed.
SE-N17-042	FR4, FR8, FR5	Check that a user can accept a battle request from a friend on another server.	Accept a battle notification by clicking accept.	User should be taken to battle page and a battle should start.	User directed to battle page.
SE-N17-043	FR4, FR8, FR5	Check that a user can send a battle request to a friend on another server.	On the friends page, click on the button to send a battle request to a friends monster.	User should be notified with a message that a request has been sent.	Message shown and request should be pending on server.
SE-N17-044	FR4, FR8, FR5	Check that a user can accept a breeding request from a friend on another server .	Accept a breeding notification by clicking accept.	User should be taken to breeding page and breeding should start.	User directed to breeding page.
SE-N17-	FR6, FR8,	Check that user can send a breeding	On the friends page, click on the button to send a breeding request	User should be notified with a message that a	Message shown and request should be

045	FR5	request to a friend on another server.	to a friends monster.	request has been sent.	pending on server.
SE-N17-046	FR3, FR4, FR10, FR5	Check that battles are executed correctly with a friend on another server .	Accept a battle request. Users should be redirected to battle page.	Battle should run. The user can then view the results of the battle. Notification should also be sent to opponent to view the outcome.	Battle shown correctly. Both users are able to view the stats of the battle.
SE-N17-047	FR3, FR4, FR6, FR8, FR5	Check that breeding monsters is executed correctly with a friend on another server.	Accept a breeding request. User should be redirected to breeding results page	Results of breeding should be shown. A notification of the breeding should be sent to friend linking them to the results of the breeding.	Breeding stats shown correctly. Both player are able to access the outcomes of breeding. Request sender gets the children and request acceptor gets money prize.
SE-N17-048	FR3, FR8, FR10, FR5	Check that the outcome of a battle with a friend on another server can be viewed by the request sender.	After a battle, check that the request sender gets a notification and can view the outcome of the battle.	Notification should be viewable and clicking it will show the outcome of the battle.	Outcome of the battle is viewable by the user.
SE-N17-049	FR3, FR6, FR8, FR5	Check that the outcome of breeding with a friend on another server can be viewed by the request sender.	After a breeding event, check that the request sender gets a notification and can view the outcome of the breeding.	Notification should be viewable and clicking it will show the outcome of the breeding.	Outcome of the breeding is viewable by the user.
SE-N17-050	FR6, FR8, FR9, FR5	Check that a friend request that has been sent to a friend on another server can be cancelled.	Click cancel on a pending request.	Message shown that the request has been removed.	Request removed from the system and no longer displayed to either of the users.
SE-N17-	FR6,	Check that a battle request	Click cancel on a	Message shown that the request	Request removed from the system

051	FR8, FR5	that has been sent to a friend on another server can be cancelled.	pending request.	has been removed.	and no longer displayed to either of the users.
SE-N17-052	FR6, FR8, FR5	Check that a breeding request that has been sent to a friend on another server can be cancelled.	Click cancel on a pending request.	Message shown that the request has been removed.	Request removed from the system and no longer displayed to either of the users.
SE-N17-053	FR8, FR9, FR5	Check that a friend request from a friend on another server can be rejected.	Click reject on a request	Message shows that the request has been rejected.	The request should be removed from the system, the other user should be informed that their request was rejected.
SE-N17-054	FR8, FR5	Check that a battle request from a friend on another server can be rejected.	Click reject on a request	Message shows that the request has been rejected.	The request should be removed from the system, the other user should be informed that their request was rejected.
SE-N17-055	FR8, FR5	Check that a breeding request from a friend on another server can be rejected.	Click reject on a request	Message shows that the request has been rejected.	The request should be removed from the system, the other user should be informed that their request was rejected.
SE-N17-056	N/a	Clicking on the help button.	Clicking on the help button takes you the informative help page.	User shown the help page.	Help page shown.

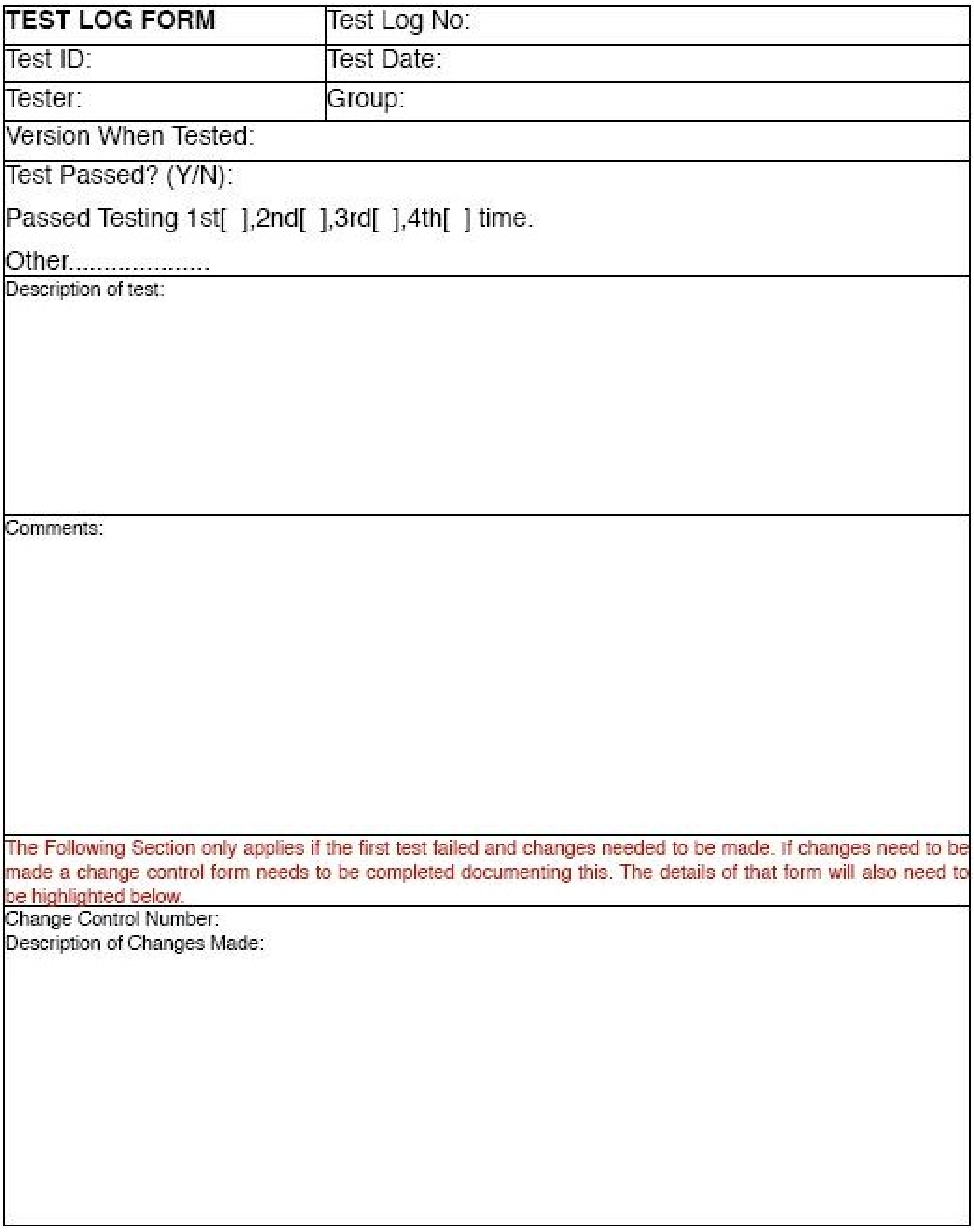
2.3 Test Result Reporting

Test results will be recorded in a test log form (included on the next page). These documents will be stored in the groups Github repository under the folder Test Data, there will be two sub folders with in this directory called Module Tests and System Tests. The corresponding test documents will be stored into those two directories.

2.4 Configurations

Using Github we can branch off from the master copy of the project for testing and working on, doing this will insure that everyone is using the same version of the project. This is to make sure that everyone runs their tests on the same copy of the project insuring consistency. Branching off from the master copy will also allow for version roll back. Meaning that if a mistake is made and the system no longer works. We can return to a stable version when we know it works.

3 Test Log Form



TEST LOG FORM	Test Log No:
Test ID:	Test Date:
Tester:	Group:
Version When Tested:	
Test Passed? (Y/N): Passed Testing 1st[ ],2nd[ ],3rd[ ],4th[ ] time. Other...................	
Description of test:	
Comments:	
The Following Section only applies if the first test failed and changes needed to be made. If changes need to be made a change control form needs to be completed documenting this. The details of that form will also need to be highlighted below.	
Change Control Number: Description of Changes Made:	

1 Design Plan

1.1 Purpose of this Section

This should provide everything designers, programmers and testers need to know to use the facilities provided by a module. It should include an outline for all parts of the system. The information provide should be enough to get a basic understanding of the system.

1.2 Scope

An outline for each class and its containing methods. Explanation of how the server works and significant algorithms. Outline of how the system is navigated and operated through sequence diagrams.

1.3 Objectives

The objective of this document is to give an overview of the system that is being produced.
The areas covered by this plan are:

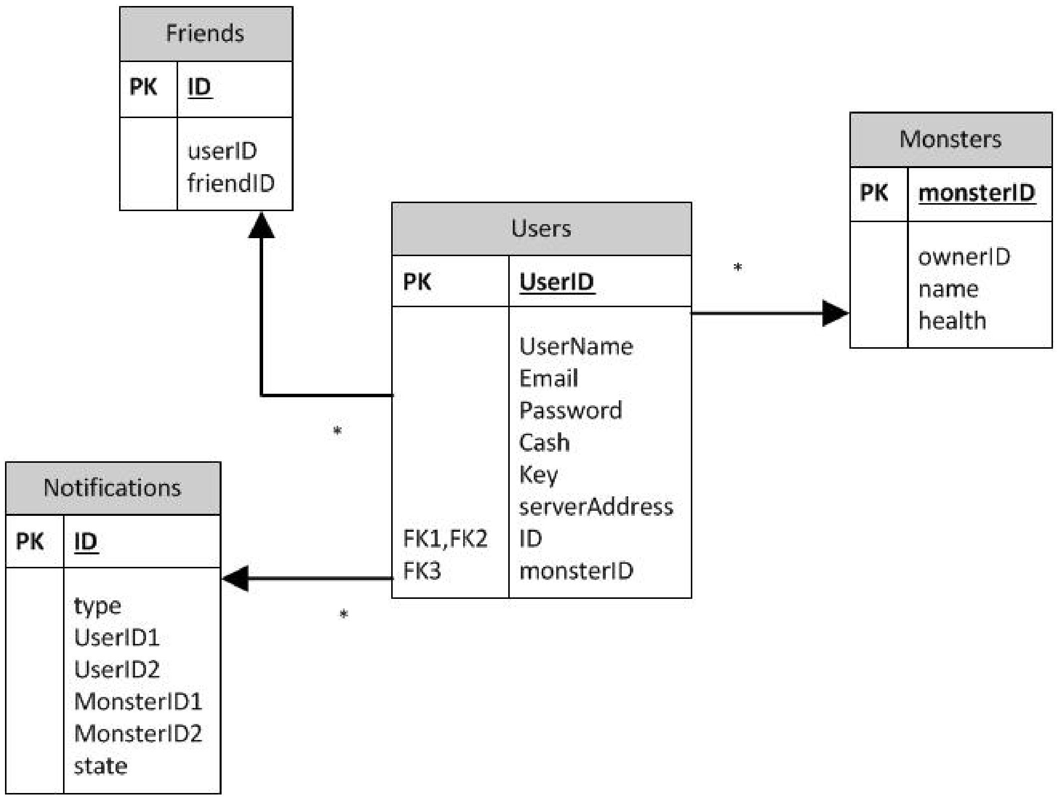
- Java Classes explanations
- Sequence diagrams
- Relational Database diagram
- JSON Table
- Algorithms

2 General Functionality

2.1 Description

An HTTP servlet is used to access the data contained within the backend of the server as if it was almost a web page. We will pass parameters between it and get a response. We will be working on the principle that one main servlet will perform different actions, passing through an actions variable and any data required to be processed. There are two methods of accessing this; using get and post requests. The get request will pass the parameters through the URL, the post request through a hidden layer, based on the users input. JavaScript will collate the necessary data, and attach the appropriate action command before sending to the server.

2.2 Relational Database diagram



2.3 Relational Database diagram explanation

This is a relational database diagram for our database, we have used 4 tables to store information about the User, the Users table to store information directly related e.g. user name etc. We then have the Monsters table in which will have information about the users monsters, the friends table to store the users friends and the notifications table to store information about any notification the user might have e.g. a user who wants to battle their monster with you.

3 Algorithms

The algorithm for the aging process ind strengths as follows:
=(10+2.7*A)*EXP((A*(-0.09)))
The A is the number of days that have passed. Upon birth it is at 100% health, after 7 days it rises to 150%. After 21 days(3 weeks) it's back to 100% and at 84 days(12 weeks) the monster shall be at 0%(Dead).

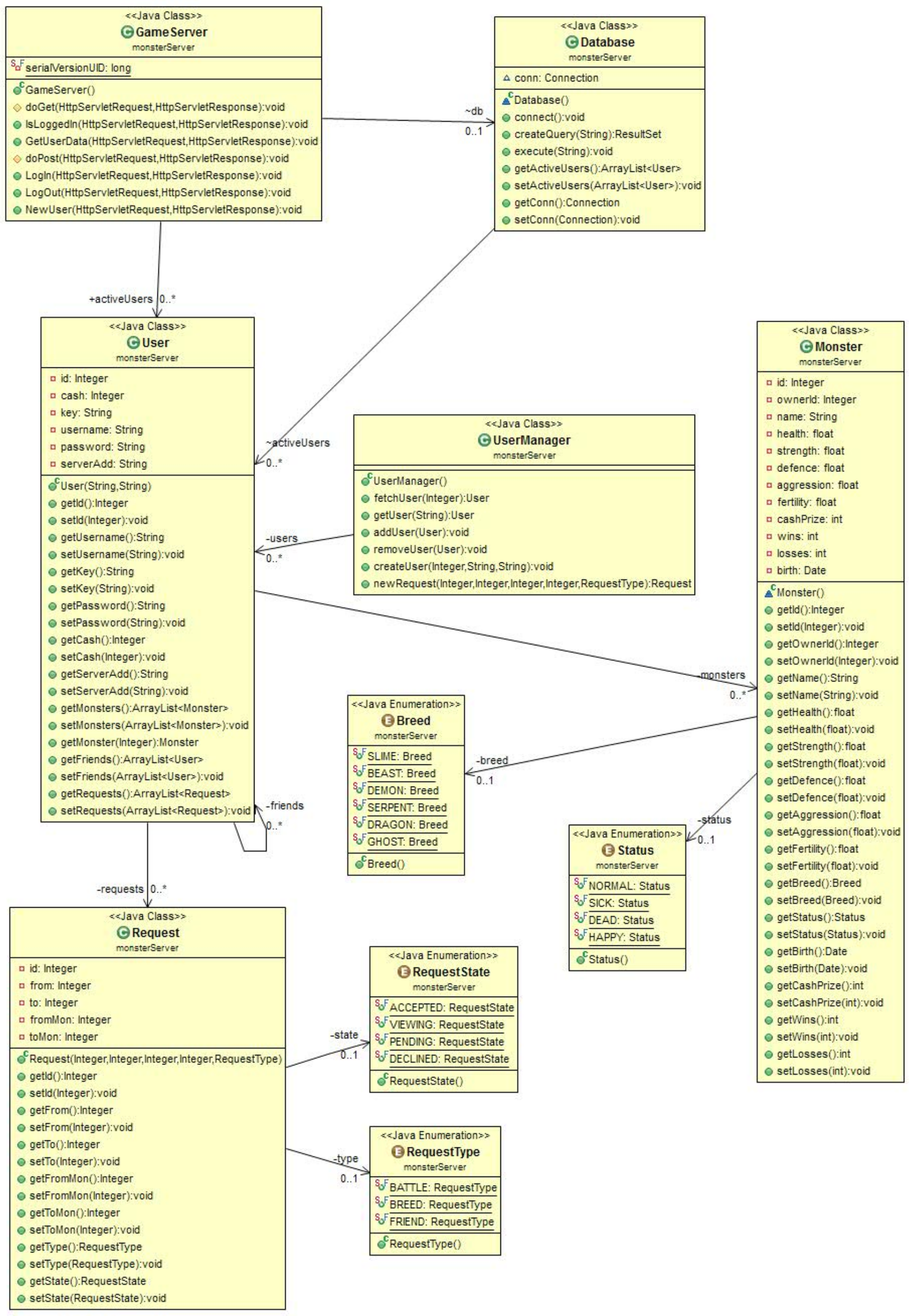
4 JSON Table

Page	Event	Description	Action	Data	Response
Profile	On Load	Request a list of the users monsters	getMonsters	N/a	N/a
Battle	On Load	Request a list of users monsters and a list of the users friends	getMonsters,getFriends	N/a	N/a
Battle	On clicking a friend	Request a list of a friends monsters	getFriendsMonsters	friendId : Int	ID representing a friend.
Battle	On clicking battle	Create a new battle request	newBattleRequest	userMonsterID:Int, friendId:Int, monsterId:int	ID of the selected monster, ID of the friend we are battling with, ID of our friends monster
Breed	On Load	Request a list of users monsters and a list of the users friends	getMonsters,getFriends	N/a	N/a
Breed	On clicking a friend	Request a list of friends monsters	getFriendsMonsters	friendId:Int	ID representing a friend
Breed	On clicking Breed	newBreedRequest	userMonsterID:Int, friendId:Int, monsterId:Int	ID of the selected monster, ID of the friend we are breeding with, ID of our friends monster	N/a
Friends	On Load	Request a list of friends	getFriends	N/a	N/a
Friends	On Load	Request a list of pending friends	getAllNotifications	N/a	N/a
Friends	Accept Friend Click	Accept a pending friend request	acceptRequest id: Int	ID representing a friend	N/a
Friends	Decline Friend Click	Decline a pending friend request	declineRequest id:Int	ID representing a friend	N/a
Friends	Add Friend Click	Send a request to connect to another user as a friend	addFriend	username: email	The users email address
Notifications Menu	On Load	Request a list of all notifications for the current user	getAllNotifications	N/a	”Notifications”: [”Type”:”BATTLE” , ”ID”:”1” , ”From”:”email” , ”Type”:” BREED ” , ”ID”:”2” , ”From”:”email” , ”Type”:” FRIEND ” , ”ID”:”3” , ”From”:”email” ,] Type can be BATTLE BREED or FRIEND
Notifications Menu	Click accept request	Accept the notification	acceptRequest	id :Int	ID of the notification
Notifications Menu	Click decline request	Decline the notification	declineRequest	Id :Int	ID of the notification

4.1 JSON Explanation

JSON (also known as JavaScript Object Notation) is a form of data interchange that is designed to be "human readable". Derived from the JavaScript scripting language, it shows simple data structures and associative arrays, which they call "objects". This table shows us the various data interchanges that take place within the design of our project. JSON is used as an alternative to XML.

5 Class Diagram



6 Significant Data Classes

6.1 Java Data Classes

There are four main classes within the Java designed to be implemented within the project, these are Monster, User, User-Manager, and Request.

6.2 User-Manager class

The User-Manager class handles many different tasks related to how each of the User profiles are created and organised. This class has creates the ability to add and remove a user from a database, this can be seen in the "addUser" and "removeUser" methods located within the class. This class can also perform other tasks that are important to the functionality of the system, such as the ability to fetch particular User data, which can either be the Users ID, or the Users name (as depicted by a String). Without this class, it would be impossible to create a usable log in page, as the system would not be able to get (fetch) the data that is stored in the database to validate their log in. This class would be absolutely necessary for a functioning registration form for the new users, as without it, no Users could be added to the system.

6.3 User class

The User class handles the more specific information appropriate each user that is registered to the system. This includes information such as the Username and password of each of the registered Users. But this is not all this class holds. The User Class also holds details that are necessary for the management of the user within the game. This includes a method to get how many monsters the User owns, and another method which calculates how much cash the User owns. This class also contains information on which Server the User is registered to, without this method within this class, the game would simply not work, as the online features, such as the battling, mating and friend requests would not know where to look for the other users also wanting to play Monster Mash.

6.4 Monster class

The Monster class will handle all of the functionality regarding monsters that the registered Users may own. This can include many basic things, such as how many battles the monster has won or lost, which User owns the monster, the name of the monster and its birthday. However, other more complex stats of the monster will be included within the MonsterStats class, this includes personality attributes, such as how aggressive each particular monster is, and also includes the battle statistics (the health, attack power and the defensive power of each monster). This MonsterStats class will also contain the fertility rate of each monster, which will be important in the breeding stage of the game. The idea behind splitting the information between each of the classes, is for added simplicity when coding, it would have been a shame not to include extra features such as the age of each of the monsters, as these extra stats add greater depth to the game.

6.5 Request class

The Request class, as the name would suggest, handles the various requests that each of the registered Users will send to each other while playing the game, these include the options to be able to fight and breed with another Users monsters, and also the ability to become friends with another registered User. Aside from these requests, this class also deals with the notification feature we are implementing within our game, this class will be able to tell the User from these notifications what sort of request another User has sent to them, who has sent it, and the ability to respond to the request.

6.6 Functional Requirements

Monster	FR3, FR4, FR10
User	FR1, FR6, FR7, FR8, F11
User-Manager	FR1, FR2, FR3, FR7
Request	FR6, FR9

Table 1: This table shows the functional of Java classes

7 Breed Class

The Breed class is a Private class that deals with the breeding process of the particular User monsters that are in question. The name of this class is Breed.class. There is an ENUM and it is also the only Public Method located within this class, this ENUM has no parameters. The Breed class is basically a set list of monster types that will be able to breed. We used ENUM so that you can only select a type from that particular list.

7.1 Public Methods

The only public method this class has is ENUM Breed and has no parameters. This is a list of set monster types that the monster will be of type. Used ENUM so can only select a type from that list.

8 Monster Class

The Monster class is named Monster.class and is a public class. This class contains getters and setters for the monsters attributes. These methods can set and get attributes such as the ID of the monster, the owners (the Users) ID, the name of the monster, the stats of the monster (which include attributes such as the attack, the defence and the aggression of the monster), the breed of the monster, the status of the monster, the cash prize that would be attained by defeating the particular monster and finally the wins and losses each monster has achieved. These methods will set the value for each of these, and will be able to return the value for the monster.

8.1 Public Methods

This class contains getter and setters for monster attributes. The getters and setters are for id, ownerId, name, stats, breed, status, birth, cashPrize, wins and losses. They set the value and return the value for the monster.

9 MySQLDatabase Class

The MySQLDatabase class is called MySQLDatabase.class and is a public class. The MySQLDatabase doesn't contain any public methods. The MySQLDatabase class's goal is to manage the data in our relational database system. This class will store all the information that is important to each particular User. This includes the Username and Password of each of the Users, the monsters they own, and what friends they are with. Without this class the game would simply break, as there would be nowhere to store and retrieve the relevant information for the User.

9.1 Public Methods

10 Request

The Request class is called Request.class and is a public class.

10.1 Public Methods

public User getFrom() - This will return from which is of type User. public void setFrom(User from) - This will set the User from and take User from as a parameter.

public User getTo() - This will return to which is of type User. public void setTo(User to) - This will set the User to value and take User to as a parameter. public Monster getFromMon() - This will return fromMon which is of type Monster. public void setFromMon(Monster fromMon) - This will set fromMon and takes Monster fromMon as a parameter. public Monster getToMon() - This will return a toMon which is of type Monster. public void setToMon(Monster toMon) - This will set toMon and takes Monster toMon as a parameter. public RequestType getType() - This will return type which is of type request. public void setType(RequestType type) - This will set type and take RequestType type as a parameter. These will be used to determine whether it is pending, accepted and declined. public RequestState getState() - This will return state which is of type RequestState. public void setState(RequestState state) - This will set state which is of type RequestState and takes RequestState state as a parameter.

11 Request state

The request state class is called RequestState.class and is a public ENUM class.

11.1 Public Methods

The only method in this class is enum RequestState. This is used to allow 3 states to be identified of which are ACCEPTED, PENDING and DECLINED. These will be used for requests such as battle and breed.

12 UserManager

The User-Manager class handles many different tasks related to how each of the User profiles are created and organised. This class has creates the ability to add and remove a user from a database, this can be seen in the "addUser" and "removeUser" methods located within the class. This class can also perform other tasks that are important to the functionality of the system, such as the ability to fetch particular User data, which can either be the Users ID, or the Users name (as depicted by a String). Without this class, it would be impossible to create a usable log in page, as the system would not be able to get (fetch) the data that is stored in the database to validate their log in. This class would be absolutely necessary for a functioning registration form for the new users, as without it, no Users could be added to the system.

12.1 Public Methods

public UserManager() - public User fetchUser(Integer id) public User getUser(String name) public void addUser(User user) public void removeUser(User user) public void createUser(Integer id, String username, String email, String password)

13 User

The User class handles the more specific information appropriate each user that is registered to the system. This includes information such as the Username and password of each of the registered Users. But this is not all this class holds. The User Class also holds details that are necessary for the management of the user within the game. This includes a method to get how many monsters the User owns, and another method which calculates how much cash the User owns. This class also contains information on which Server the User is registered to, without this method within this class, the game would simply not work, as the online features, such as the battling, mating and friend requests would not know where to look for the other users also wanting to play Monster Mash.

13.1 Public Methods

public Integer getId() - This will return id which is of type Integer. public void setId(Integer id) - This will set id and takes Integer id as a parameter. public String getUsername() - This will return username which is of type String. public void setUsername(String username) - This will set username and takes String username as a parameter. public Integer getKey() - This will return key and is of type Integer. public void setKey(Integer key) - This will set key and takes Ineger key as a parameter. public String getEmail() - This will return email and is of type String. public void setEmail(String email) - This will set email and takes String email as a parameter. public String getPassword() - This returns password and is of type String. public void setPassword(String password) - This will set the password and takes String password as a parameter. public Integer getCash() - This will return cash and is of type Integer. public void setCash(Integer cash) - This sets cash and takes Integer cash as a parameter. public String getServerAdd() - This will return serverAdd and is of type String. public void setServerAdd(String serverAdd) - This sets serverAdd and takes String serverAdd as a parameter. public ArrayList<Monster>getMonsters() - This will return a list of monsters of type ArrayList<Monster>. public void setMonsters(ArrayList<Monster>monsters) - This sets monsters and takes ArrayList<Monsters>monsters as a parameter. public ArrayList<User>getFriends() - This will return a list of friends of type ArrayList<User>. public void setFriends(ArrayList<User>friends) - This sets friends and takes ArrayList<User>friends as a parameter. public ArrayList<Request>getRequests()

- This will get a list of requests of type ArrayList<Request>. public void setRequests(ArrayList<Request>requests) - This will set requests and takes ArrayList<Request>requests as a parameter.

14 Status class

The Status class is called Status.java and is a public enum class.

14.1 Public methods

This class is an ENUM class and defines a set of 4 statuses that monsters can have. NORMAL, SICK, DEAD and HAPPY.

15 RequestType class

The Request class, as the name would suggest, handles the various requests that each of the registered Users will send to each other while playing the game, these include the options to be able to fight and breed with another Users monsters, and also the ability to become friends with another registered User. Aside from these requests, this class also deals with the notification feature we are implementing within our game, this class will be able to tell the User from these notifications what sort of request another User has sent to them, who has sent it, and the ability to respond to the request.

15.1 Public methods

This class is an ENUM class and defines a set of 3 types for request. BATTLE, BREED and FRIEND.

16 Data Storage

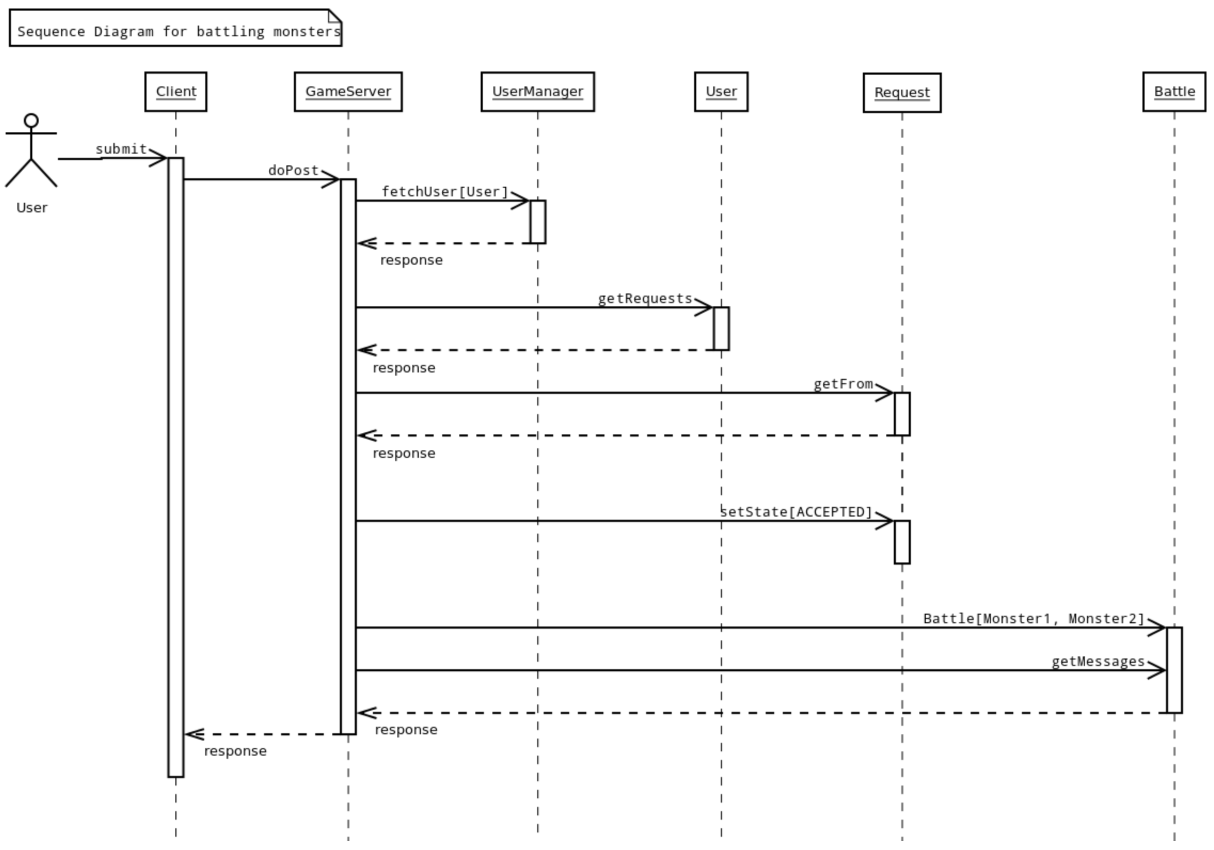
Within the java programing the data is stored in a number of ways, a major way that data will be handled is through enums; requests will be handled through this and will be stored as either battle, breed, or friend, and the states of these requests will also be stored as enums, being either accepted, viewing, pending, or declined. Also stored as enums will be the statuses of monsters, thusly each monster status will be handled as normal, sick, dead, or happy. Breed considers the different types of monster that we plan to be breeding with each other, and this therefore will be stored as an enum with the value of slime, beast, demon, dragon, serpent, or ghost.

Array Lists will also be used, but only, as far as designed, in private instances, so there will be no public instances that need to be explained in class diagrams. Otherwise, usernames, passwords, and the like will be handled in private strings and variable types as are appropriate.

17 Sequence Diagrams

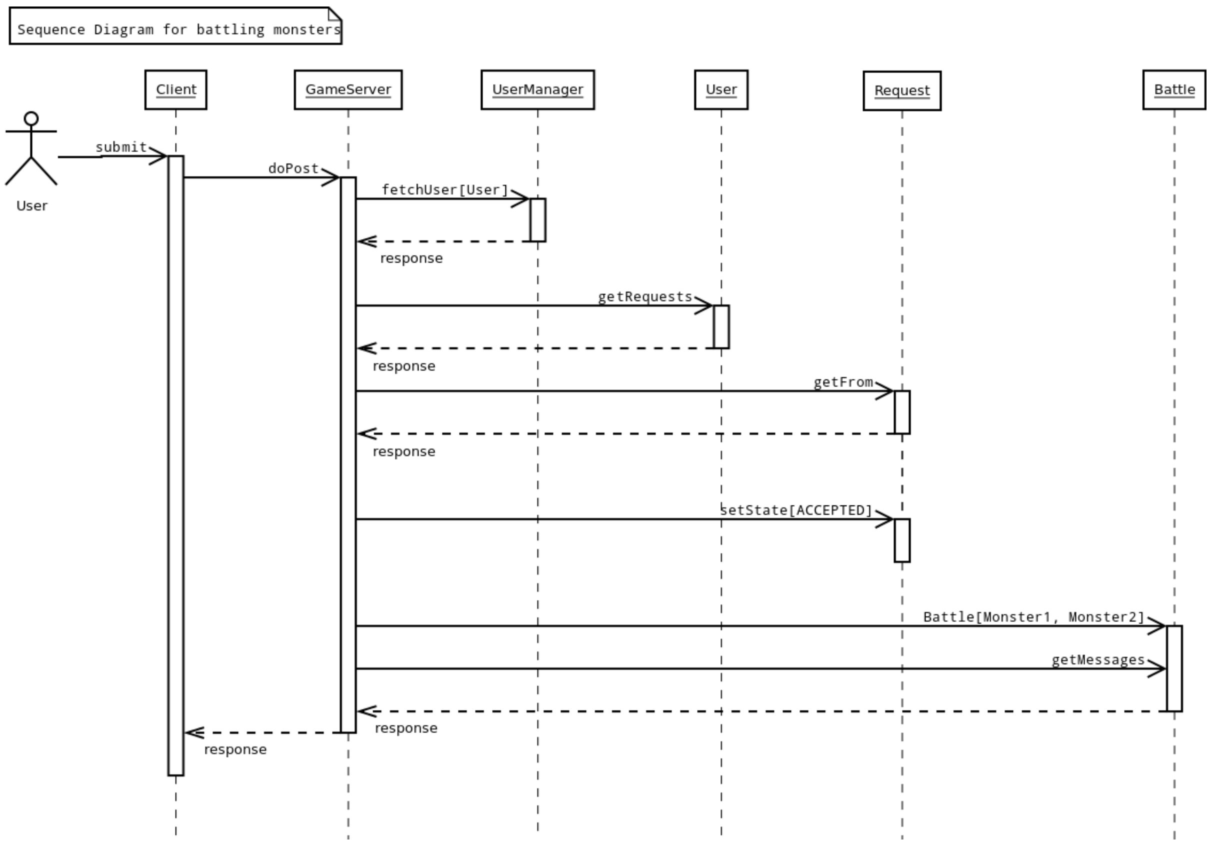
17.1 Accept Battle

This sequence diagram shows a user accepting a battle request that has been sent to them by a friend. The diagram shows the users response being sent to the server using the doPost method. The request then runs through the relevant classes in order to gain the information required.



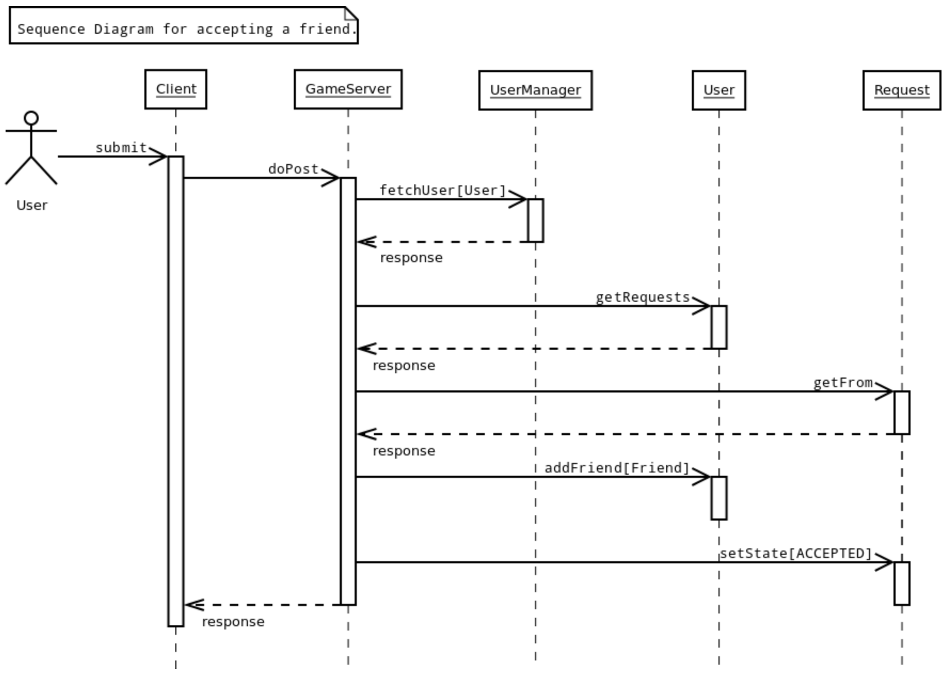
17.2 Accept Breed

This sequence diagram shows a user accepting a breed request that has been sent to them by a friend. The diagram shows the users response being sent to the server using the doPost method. The request then runs through the relevant classes in order to gain the information required.



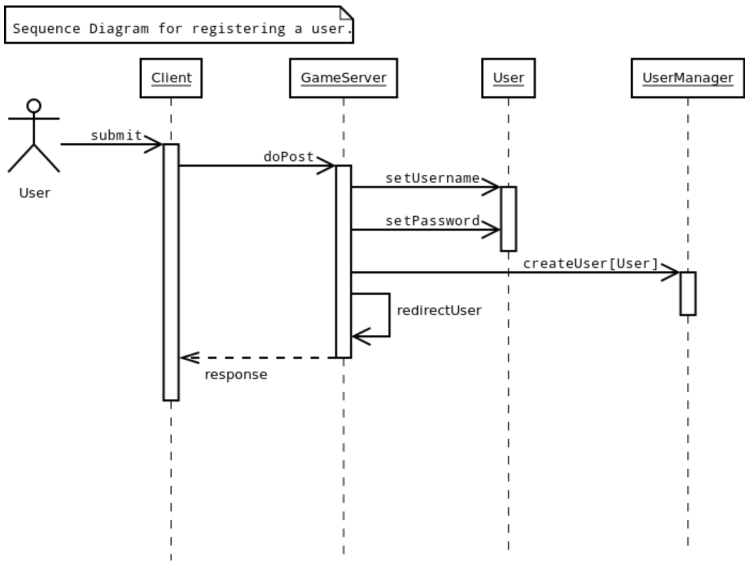
17.3 Add Friend

This sequence diagram shows a user adding a friend. The users request is sent to the server using the doPost method. The request then runs through the relevant classes in order to gain the information required. A request is then sent to the corresponding user using the addRequest method and User class.



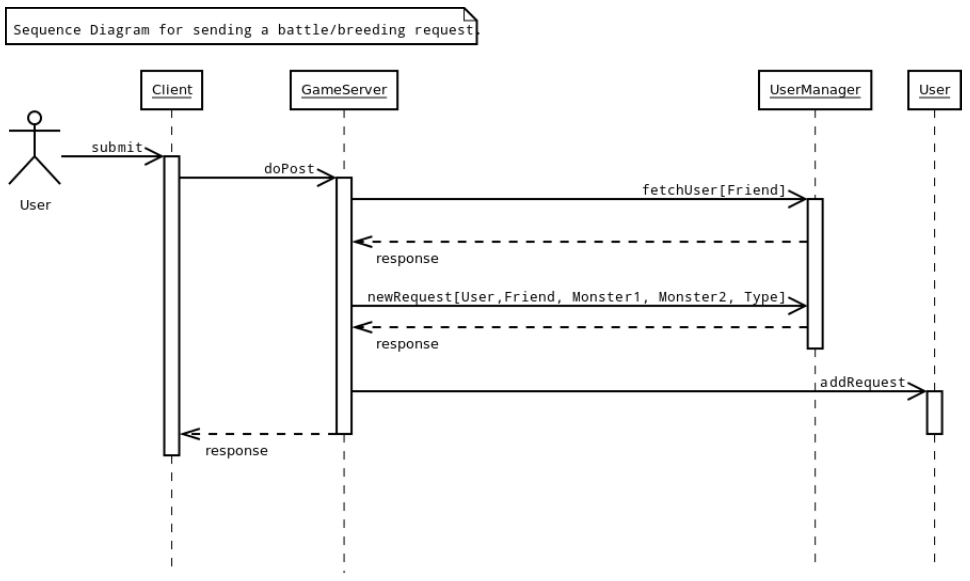
17.4 Register User

This sequence diagram shows a user submitting their password and username to create a monstermash account. The diagram shows the users password and username being sent to the server using the doPost method. The request then runs through the relevant classes in order to process the users details and create the new account.



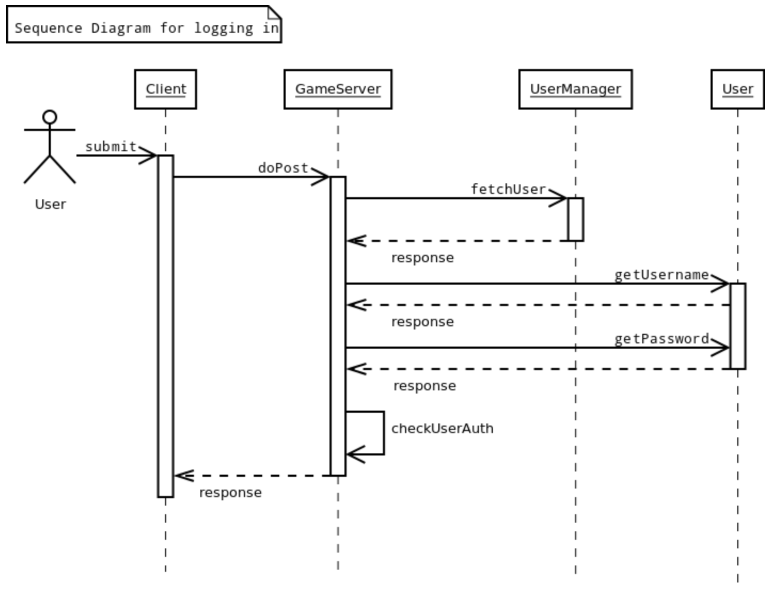
17.5 Send Battle/Breed Request

This sequence diagram shows a user sending a breed or battle request to a friend. The diagram shows the users request being sent to the server using the doPost method. A request is then sent to the corresponding user using the addRequest method and User class.

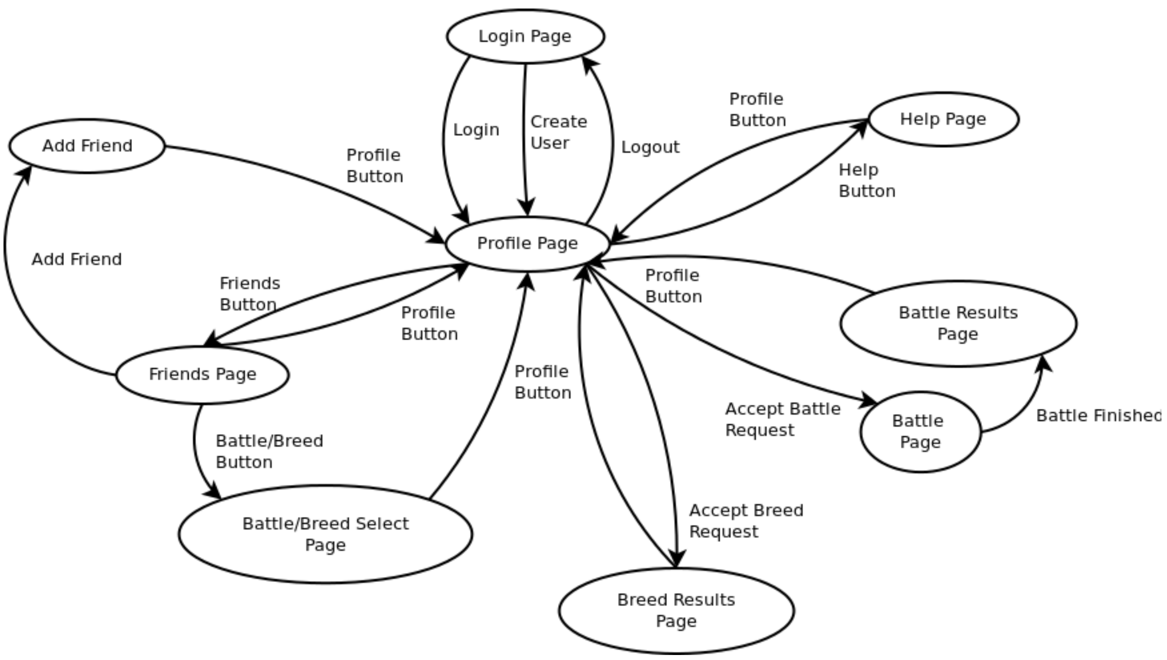

Sequence Diagram for sending a battle/breeding request
Client
GameServer
UserManager
User
User
submit
doPost
fetchUser[Friend]
response
newRequest[User,Friend, Monster1, Monster2, Type]
response
addRequest
response

17.6 User Log In

This sequence diagram shows a user logging in to monstermash. The user submits their login information and it is posted to the server using the doPost method. The request then runs the the relevant classes, checking the login details. A response is generated if the login details are invalid or the user is logged in if they details are valid.



18 State Diagram



18.1 State Diagram Description

As you can see, the diagram above shows the design structure for how we want to be able to navigate each of our web pages. I will go through each connection individually, and explain the reasoning behind each.

18.1.1 Login Page to Profile Page

This is the first page that the user will be able to see when going onto our website, the page will require the user to enter their username and password into the allocated areas, once this is done, the user will be taken to the profile page, this is because the profile page is regarded as the users most important page, as this is where the user can decide what s/he plans to do.

18.1.2 Profile page to Various pages

The reason we had the profile page link to most of the pages on the website, was simply because this page acts almost as the home page of our structure, and it makes it easier for the user to have just one page in which they can select most of their options. If we separated and spread out each of the links to various pages to other pages in the structure, this would likely confuse and frustrate the user, by having them all in the same place, this avoids most issues about navigation that may arise.

18.1.3 Profile page to Help page

It is most likely that if the user had a problem, they would not be on any page other than the profile page, as this would be the starting point for the user, for this reason, we linked the Help page only too and from the profile page.

18.1.4 Profile page to Battle page

As the Battle page one of the most important pages (Alongside Friends) it was important that we put the link to this page in an obvious place, and there is no place more obvious than on your Profile page, as (as was explained before) this page works as the home page of this website, a page that is constantly referred to.

18.1.5 Battle page to Battle results page

After you have accepted an offer of a fight, the user will be taken to an area called the Battle page, in which you can watch the battle between your monster and the opponent that challenged you, after the battle has been completed you will be taken to the Battle results page, we structured this so that if you are the user that sent out the battle request, you will automatically be taken to the Battle Results page, as (as we have explained previously) the user sending out battle requests does not get to view the battle, but just sees the results. After the users have seen the results of their battle(s), they have the option to return to the Profile page, so that they can continue to do other tasks within the game. Profile page to Friends page:

The link to this page is very important, it is from the Friends page you will be able to add and accept who you want to be friends with in monster mash, who you would like to battle with, and whos monsters you would like to breed with. Without this page, the game simply would not work. So for this reason, we linked this page to the Profile page, similar to the reason with the Battle page, this was done so the user can find this page easily, as the Profile page will be the most visited page on the website.

18.1.6 Friends page to Various pages

From this page, it is possible to do three separate things, you will be able to manage your friends, select which monsters you wish to breed with, and select which monsters you wish to battle. We decided to put these options on this page and NOT on the Profile page, because you would begin to run the risk of cluttering the Profile page, as important these tasks are to the game, we concluded that it would be much more neat and less confusing to put these options within a separate Friends page, so the user wouldnt be drowned with options on the Profile page.

18.1.7 Friends page to Battle/Breeding page

Originally, we were going to have the Battle and Breeding pages separate here, so the user would be able to select which one they wanted. It was only later on that we realised that both these tasks could be accomplished on the same page, so having them on the same page was unnecessary. Once the user has selected whether they want to breed or battle certain monsters, the user will be directed back to the Profile page, this is a running theme in our structuring, as this is where everything on the website stems from.

18.1.8 Friends page to Add Friend page

It would seem fairly obvious that from the Friends page, we would be able to select which friends we would want to add, so from here, we added an Add Friends page, in which the user can type the recipients email or username, to send a friend request. Once the user has finished adding the friends they wish, they can then link back to the Profile page, to continue with other tasks they may wish to do.

18.1.9 Profile page to Breeding Results page

This link is very similar to the Battle page explained earlier, the only difference being that it does not matter who was the recipient of the offer or not, they would both be linked straight to a breeding results page. Once the user has looked at the results, they can then link back to the Profile page.

18.1.10 Profile page to Logout

For a small while, we did consider having a logout page on every page of the website, but we theorised that this may cause problems that would be difficult to fix, for example, what would happen if you logged out in the middle of a battle on the Battle page. For this reason, to avoid confusion, we only put the logout page on the Profile page, as this guarantees that the user is not in the middle of a task.

1 Introduction

1.1 Purpose of this Document

This document is produced to aid an effective and sensible maintenance of the system produced previously as part of the System Development Life Cycle project, called Monster Mash. Herein are described all key parts of that system, the natures of them, and their usefulness to each other, including how and where data is stored, and why parts of the program are designed as they appear.
This document should be read and understood by anyone unfamiliar with the system before and changes are made by that person to any part.

2 System Details for Maintenance

2.1 Program Description

The Monster Mash Game is a web based program that, using a server side database and java methods, allows a user to create an online account and manage a list of monsters therein. A system of money allows the buying and selling and monsters, while breeding and battling allow users to interact and compete with each other.

2.2 Program Structure

The program has three main levels of programming; the java, which handles the database and the main methods to manipulate that data for use in the game; the HTML and CSS, which creates the web page to display the program in, and is the entire user interface of the program; and lastly there is the javascript, which manages the interchange of data between the java controlling the database and the HTML and CSS controlling the user interface.

2.2.1 Java

The java is structured in a number of related classes that handle the data being requested from the database by the user, there are a number of different classes that each create a separate individual part of the game, including battling, breeding, and selling.

2.2.1.1 Battle.java

Battle manages all battle related methods, it holds the following:
doBattle, which returns an arraylist of strings that are the victors of battles, is

a short public method that calls the attack method and handles the entire battle round by round.
The next method is the private method declareWinner, used by the doBattle method to create an arraylist that holds a SQL statement to update the database. It updates the status of the winners and losers, and handles things such as setting health of the loser to 0, and adding victory money to the winner.
The final method in Battle.java is the attack method, which is a void method and uses variables atkMon and defMon to work out the amount of damage recieved by a monster from the given attacking monster's level and the defending monster's level.

2.2.1.2 Breed.java

Breed.java is an enum class and contains the options SLIME, BEAST, DEMON, SERPENT, DRAGON, and GHOST.

2.2.1.3 Breeding.java

Breeding.java handles the breeding of monsters and allows the creation of new monsters from the attributes of two parents.
There is only one method in this, which is doBreed, this takes the two users and their two monsters which are being bred, and from these calculates a new monster with new attributes based on the two previous monsters' attributes, a random amount is then added to this to give a chance of the new monster being more powerful than the previous two. An SQL statement is then generated and returned to the database to store the new monster.

2.2.1.4 Database.java

The Database class manages the connection with the database, which allows the SQL statements generated by other classes within the program to be sent to the database.
The first method is the connect method; this creates the connection with the database.
createQuery is the next method and executes a query that another method of the java has generated previously, this then returns a ResultSet.
execute follows this method and takes a string, which is used to prepare an update statement for the database.
getConn is the penultimate method in this class and is used to get a connection variable.
setConn is the final mathod and opposite of the former, and is used to set a connection variable.

2.2.1.5 GameServer.java

This class handles the management of connections between the server of the client, and actions that the client does that the server has to respond to.
The first method is called doGet and takes an Http Servlet request and response. This then works out whether the user is logged in, or whether user data has been requested by the user, depending on how the method is called.
The second is called IsLoggedIn, and takes a Http Servlet request and response, the same as the former method. It works out whether a user is logged in by finding out if their session has a username stored, if it is then the user is logged in, otherwise the user is not logged in.
Then there is GetUserData, which takes a HTTP Servlet request and response before printing the relevant articles returned to it and sending them on to the method from which it was previously called.
doPost is the next method and this again uses a Servlet request and response to get the kind of server interaction requested by the user; it is filled with case statements, which pick between a number of different options depending on the request sent to the server.
LogIn is the next method that allows a user to log into the server and manages a session with that user.
LogOut follows this and ends a session before setting a suer as logged out.
NewUser follows LogOut and is the method that handles the registration of new users, it creates the SQL queries to register them in the database and creates a new account with user name and password, as well as a new monster.
addFriend comes next and handles sending a friend request to another user for the to accept or decline.
newBattleRequest follows this and generates a new notification for a given user that a battle has been requested with them.
newBreedRequest is next and creates a new request for a given user to be accepted or declined.
newBuyRequest follows and handles a user buying another user's monster, it generates the SQL to handle money changes and monster transfers.
getMonsters comes next and gets monsters and their attributes.
getFriends gets a list of friends and their details.
getFriendsMonsters gets a list of friends and their monsters.
changeName allows a user to change the name of a given one of their monsters.
getAllResults gets the currently existing results of all battles that included the user who is requesting the data, and generates an SQL query to get this data.
getAllRequest is used on the notification page and is used to get all notifications that a user has.
acceptRequest is used to manage the acceptance of a request by a user who has received one.
acceptBattleRequest is used to accept a battle request by a user who has received one, and initiates a battle using Battle.java.

acceptFriendRequest is used to accept a friend request from the notification page and generates the SQL necessary to add that friend into the database.
declineRequest is used to remove a request without accepting it, and generates the necessary SQL for such an action.
setBreedCost is used on the auction page to set the breeding cost of a monster, and generates the SQL for that process.
setBuyCost does much the same as the previous function, except rather than setting a breed price it sets a price to sell the monster entirely.
getRichList is used on the friends page to your friends in order of richest to poorest, as well as display the amount of money that they have.
reloadfreinds is a method that reloads the friends that a user has and refreshes them, and generates the SQL necessary.
reloadmonsters does the same as the above, but for a list of your own monsters.
The final method in the class isreloadnotifications, and refreshes the notifications that a user has, also generating the SQL for it.

2.2.1.6 Monster.java

This is used to manage the creation and life of monsters. Firstly, monster has its own method also called Monster, which creates an instance of it with a name and an owner.
There are then two methods to handle the money that a monster sells for, one called getCashSell that gets what another user has set their monster to be sold for, and another called setCashSell that allows a user to set a sell value for a given monster.
Next there are two methods to handle the amount that it costs to breed with a monster, one called getCshBreed, which gets the amount that it costs to breed with a monster, and one called setCashBreed, which allows a user to set an amount that it will cost to breed with a monster.
Then there is generateStats, which generates the attributes of a new monster.
The above calls generateGenetics and generateBases, the latter generating base attributes of a monster, and the former generating a random amount to add or take away from that to create a unique set of stats.
There is also ageHealth and calculateAge, which relate to the ageing and status of the monster through its life.

2.2.1.7 MonsterStats.java

Contains the same methods as the above class, except for the four relating to sell and breeding prices.

2.2.1.8 Request.java

Request just sets up a number of variables for use as an instance of itself, and has no major functions that affect or put to use other methods or classes.

2.2.1.9 RequestState.java

RequestState is an enum class that contains the states that a request can be in, these are ACCEPTED, VIEWING, PENDING, and DECLINED.

2.2.1.10 RequestType.java

RequestType.java, like the above, is also an enum and contains the types of request there can be, including battling, breeding, and adding as a friend.

2.2.1.11 Status.java

This again is an enum class and contains the different states that a monster can be in, it should be noted that they are not all implemented in the currently released version of this program, these options are NORMAL, SICK, DEAD, and HAPPY.

2.2.1.12 User.java

User mainly sets up variables for the creations of instances of itself, it does contain the getMonster method which gets a list of all monsters that a user owns.

2.2.1.13 UserManager.java

The first method in this class is fetchUserFromDatabase, which gets a user's details from the database by their ID, then there is another of the same name that fetches the data by Username.
Then there is cheakForUsersMonsterss, which checks whether a user has an existing monster.
This method is followed by generate, which generates a new monster if the user has none.
Then there is fetchMonsterFromDatabase, which takes the user ID and finds the monsters that that user has.
Following this is fetchFriendsMonstersFromDatabase, which finds all of the monsters that friends have and sends them to the user.
There are two fetchUser methods, one returns a username from a given name, the second returns a username from a given ID.
There are then two small methods to add and remove a user, as well as one to create

an instance of a new user, and a final one to create a new request between two users, which takes the two users' IDs and their two monsters' IDs.

2.2.2 HTML and CSS

The HTML bases all styles off of the stylesheet base.css, so all visual changes to the HTML style must be done through the CSS stylesheet. A number of parts of the HTML that the css affects are generated by the javascript, which fills a web page with the user's data that is then changed by the CSS into a more aesthetic style. These HTML tags are added in by the javascript, and so styling of these parts of the web page should be adapted via changes to the javascript files and the data that they output.

2.2.2.1 index.html

There are many small html pages that simply contain the different titles for each page and are applied over the Index.html to create the appearance of individual pages. Therefore index.html is the key file and contains all of the parts as affected by the CSS file, the tabs and other parts are implemented here but any style changes should be made in base.css.

2.2.2.2 base.css

This contains all of the different styles that are applied to the web site and any changes that need to be made to the look of the site should be implemented here.

2.2.3 Javascript

The Javascript is used in transactions of data between the client and the server; it validates data and conducts minor processes on that data before making sure that it is stored. Alternatively, it can conduct a process on the data within the database, before displaying it to the user by manipulating it into the correct HTML tags during web page generation.

There is a javascript file relevant to each webpage that the user can visit within the monstermash site, each of these contains a number of methods that are then called and used within the particular page that they are relevant to.

2.2.3.1 auction.js

Auction.js contains getMonsters, which allows a user to see a list of all their monsters, tehre is also getFriends so that a user can see a list of all their friends. There is also getFriendsMonsters called so that a list of friends' monsters can be displayed. These are all added into the HTML code taht is generate by way of a number of functions that added on click events to names so that menus can be expanded.

2.2.3.2 base.js

Base.js implements the logout and isLoggedIn functions from the java, useful on every page to check for or end sessions.

2.2.3.3 battle.js

Battle.js implements sending data to and getting it from the getFriends, getFriendsMonsters, and getMonsters methods to create a list of friends and monsters that can battle and that you can choose to battle each other.

2.2.3.4 friends.js

Friends.js implements the exchange of data using the getFriends and getRichList methods to create lists of friends and a way that you can see your friends organised by wealth. It then has built in outputFriendsList and outputRichList functions which display the returned data.

2.2.3.5 login.js

There are two validation parts at the start of this file that validate both the login screen and the registration screen. There then follows a postLogin function, which directs user data to the server, a redirect function, which takes the user to the game if the login data is correct, and a validateUserDetails function that outputs a message if there is a problem with the validity of data input.

2.2.3.6 menu.js

This is a currently unimplemented javascript file that was planned as a control file for the navigation menu.

2.2.3.7 notification.js

This javascript file starts by getting data from the getAllNotifications method, allowing it to find all the notifications that a user has. This is followed by the addResponseEvents function that adds an on click option to each of these so that a user can accept, decline, or view certain notifications and their results. It then has the writeNotification function, which writes out the buttons relating to each function as well as the html to display the notifications themselves.

2.2.3.8 profile.js

The profile page firstly uses the java method getUserData to get data about the user from the database, as well as the getMonsters java method to get the monsters owned by that user from the database. It also outputs a table of those monsters from the javascript function that accesses the method. It also creates the change name button for monsters by using the addChangeNameEvents, which is appended to each monster within the monsters table. There is then the buildProfileHTML function that creates the page and displays everything.

2.2.3.9 results.js

The results.js file first uses the getAllResults java method to get any results left from battles or breeding that are within the database, there is also an addResponseEvents that handles hiding these results by pressing an OK button. There is the writeResult function, which begins writing the results page, this is then followed by the functions writebattleResult, writeBaby, and writeBuyResult, which manage writing out certain results to the page and formatting them respectively. The last function is writeResponse, which handles the result fading away after OK is pressed.

2.2.3.10 general.js

Within the general.js file there is the getGETvars function that finds the functions that have been passed through Get to the URL, it then decodes the arguments and returns them. Following this there is the outputList function that outputs the HTML of a page, tyhen the getParentID, which gets the ID of an object that is selected. Following this there is an isint function for general use that parses ints from other variable types, a buildMonsterHTML function that builds a general table of monster details for use in the website, a buildFriendHTML function that does much the same but for friends, a verboseType function that turns a response from the server into a message readable by a user, and finally a generic server response function called writeServerResponse.

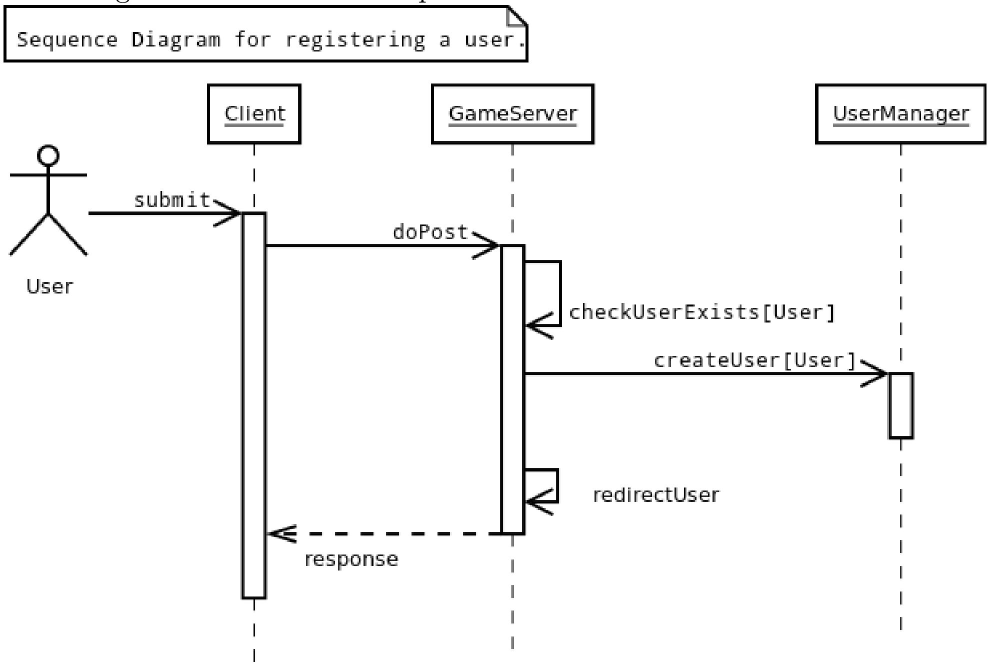
2.2.3.11 validate.js

This is the final major javascript file and is used for validation on a number of parts of the website; there is the validateEmail function that takes an input and decides whether it is a valid email address, then there is the validatePassword function that validates whether a password is long enough to be a valid password, then lastly there is the validateNewPassword which does much the same as the previous function, but for a new user's password.

2.3 Algorithms

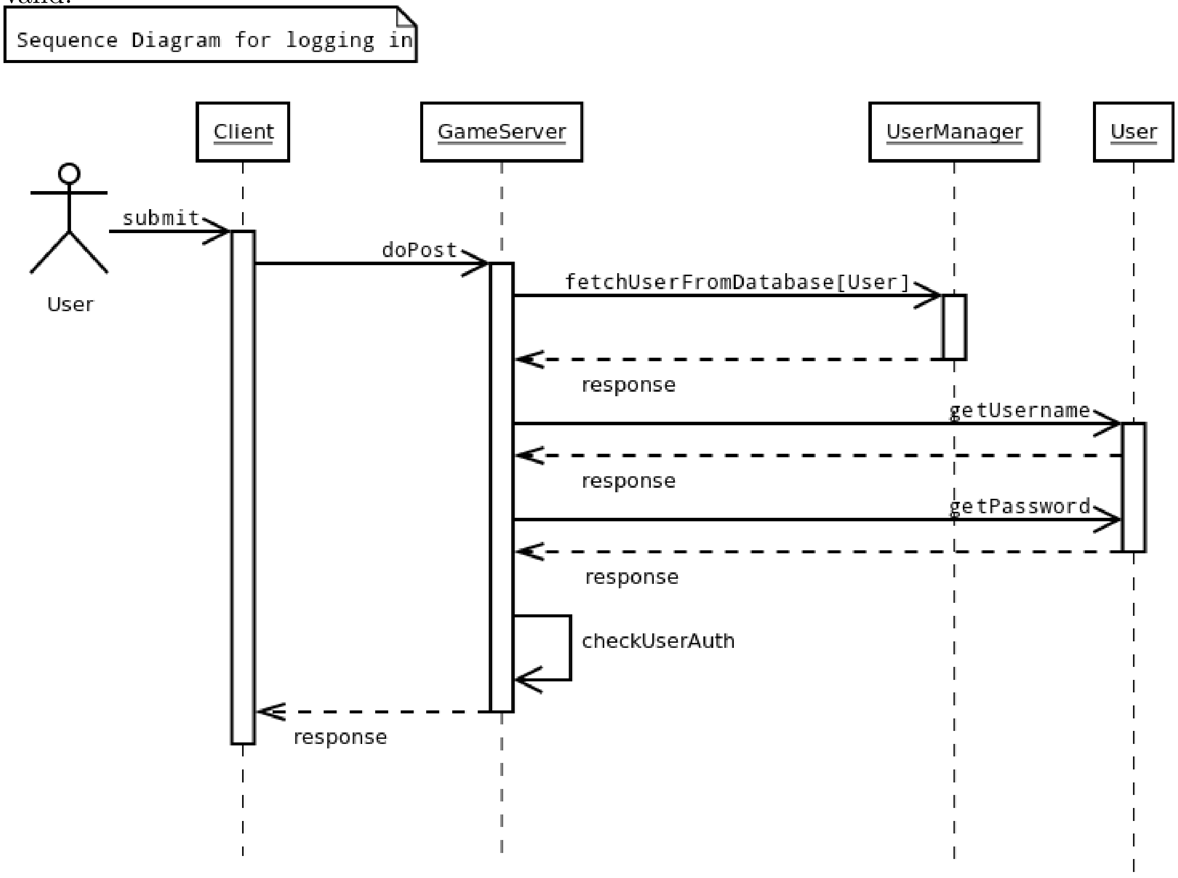
2.3.1 Registering a New User

This sequence diagram shows a user accepting a battle request that has been sent to them by a friend. The diagram shows the users response being sent to the server using the doPost method. The request then runs through the relevant classes in order to gain the information required.



2.3.2 Logging In

This sequence diagram shows a user logging in to monstermash. The user submits their login information and it is posted to the server using the doPost method. The request then runs the the relevant classes, checking the login details. A response is generated if the login details are invalid or the user is logged in if their details are valid.



1 Reflective Reports

1.1 Aleksandra Badyda

The project Monster Mash, while being largely enjoyable to work on, has undergone many changes since the beginning. This, together with the fact that the information flow from Bernie to various groups was less than ideal, has caused quite a bit of chaos, and forced us into rewriting most of the back-end code during a Tuesday afternoon and the week that followed.

My task was to be in charge of version control (via using GitHub as our platform of choice) as well as writing up the HTML and CSS for the front-end of the application. Ive volunteered for both tasks, mainly due to my previous work experience as a part of a small web developer group. The main task that I had to deal with was teaching my group mates the basics of using git and later on fixing up various mistakes or just maintaining some sort of order in the project repository.

Work with my group members has been a pleasure, despite one of two moments when the pressure has gotten the better of me. Overall, given our circumstances and the amount of work that we have all put into the project, I feel that it was a very worthwhile activity and certainly a pleasant one, despite all the stress in the testing and implementation week. Weve been lucky enough making sure that everyone worked at least a little and avoided any sort of a major crisis.

1.2 Dave Clark

My initial thoughts when we were first told we had to develop a project within a group of 9 people; it seemed rather a daunting task, as I didnt know who I would be working with or if the group would get along as a whole. The project being a monster mash game seemed interesting enough and I thought it seemed a good project to work on. I also hoped that there would be people in my group who would be keen and eager to get started on the project.

My first memory of the project was meeting up in AberVaults to meet my fellow group members as arranged by a Facebook group set up to announce meetings and general news to the group as most people have Facebook. Here we discussed ideas, designs and functionality; we also used the time to get to know each other.

The best thing about the project I thought was meeting new people I might not have talked to otherwise and improving relations with fellow students who do the same module. It also made you think about working more as a bigger group and how you had to co-operate together to get tasks done.

My final impression of the project was; I felt that we had done a good job at implementing the game, as we only had 2 functional requirements missing. We worked well as a group and always seemed to get on with each other.

In summary I thought it was a worth-while thing to do. I learnt a lot about how to work as part of a group of people to develop a game. This was a change to working on my own or in a smaller group. It gave me an insight into how you needed to do things differently when working as a group and how it helped to have such software such as version control and control forms to keep on top of things and to help manage a project more successfully.

1.3 Christopher Marriott

The task that we were set for this assignment was to create an interactive monster fighting/breeding selling game that could interact with other servers over the web. At the start of this project we all met up for the first time to meet our other team members and get to know them. Upon first meeting a few roles were set out, the first role that was assigned was my own, this was for me to be project manager. The reason being was I came with my laptop and a pen and paper to take notes. We also discussed peoples strengths and weaknesses in the group. I was very fourtunate to have a rather large group in which everyone had a different part they would be happy to work on.

At this first meeting we discussed things like what it would look like and a few other minor details. If I had the chance one thing I would have done then is steer the conversation more towards the technical aspect of the system first. At the end of the meeting we decided to meet up more regularly and set a more regular date od weekly on Thursdays. At the next couple of meetings more detailed roles were decided for people and work began on the system and documentation.

Next the we worked on creating the project plan and risk assesment, designing the interface and the database design and handling. This seemed to go rather well and we managed to get a decent project plan in on time. Some of the system had started to come together by this time. The whole team seemed to be working well together with people exchanging ideas and with no one person trying to tackle it all themselves.

As the weeks went on we worked on creating all the documents required, with team members learning LaTeXto aid in the document standards and GIT to keep everything backed up and safe. The systems parts were coming along nicely at this point. Then the christmas holidays came and work slowed down a little. As Project leader I felt I should have been more on top of this and keep everyone more informed and provided more tasks. When we came back after the holidays we were all aware that there was a lot of work that had to be done.

As the deadline approached so did 'Intergration and Testing week'. This was where the team really pulled together to produce some amazing work. Major credit should got to both Samuel Jackson and Chris 'Tux' Lloyd who work amazingly as a team and got the system working up to the point its currently at. Other team members performed well during this time such as Tom Reed who kept documenters up to date with tasks and allowed the programmers to continue on with their work and reported issues/bugs to them as and when needed.

I feel this project went well and everyone suited the roles that they were assigned. However, I believe I could have performed better as project manager by keeping more on track than I did and on several occasions I had to ask for help from Samuel Jackson who helped me out as deputy project manager. The overall result of the system is brilliant with excelent documentation. I could not have asked for a better more well rounded group.

1.4 Samuel Jackson

My role in the group project was to be one of the main programmers for the front end of the site and to aid with the general design of the system. While the finished system which we produced managed to meet most of the requirements outlined in the brief, I feel that there were a lot of things that hindered the progress and development of the project more than there could of been.

On the whole, I felt that the majority of the early part of the project went reasonably well. Our team managed to schedule and attend regular meetings and could set targets and activities to each of the team members and generally kept to the deadlines set by the project manager. I also feel that people were adequately assigned to their roles according to their programming/design experience and individual preferences.

However, I feel one of the major flaws with the project was the failure to fully analyse the implications of the design we produced. This lead to the design team (myself included) designing a system that could not work in the way intended. This was largely because of concurrency issues when multiple users are logged into the system.

Subsequently, much of the back end of the system needed redesigning on the fly during the early stages of implementation and testing week. This later caused knock on delays with implementing some functional elements of the system (e.g. ageing and server-to-server communication) as well as delaying the start of the testing phase.

This coupled with the added hindrance of frequent university network outages meant that we had to make strategic practical decisions in order to get as many of the functional requirements up and running by the Friday deadline. This included actively deciding to drop some of the more subtle functional requirements in order to concentrate on the more important elements of the system (i.e server-to-server communication). If this were a real software development project, I believe that the user would either of been force to accept a lower quality of product or grant the team an extension of another couple of days.

I would like to outline at this point that if it were not for the network outages I have full confidence in that fact that our team would have been able to get at least one of the missed requirements serviceable by the deadline and quite possibly both of them.

Turning to the more positive elements of the project, the logical data structures and algorithms (such as those used for battling and breeding) which were developed in isolation from the rest of the system and integrated during coding week work straight out of the box. When it came to there integration, all that was needed was for Monster objects to be passed in and the pre-written code did the rest. This meant that more development time could be devoted to other issues with the system.

A key factor in the success of the this project was largely down to the rigour and efficiency of the testing team. Once we had the system to a working but highly buggy stage, the testing team was set loose on the project and ran through all the tests in our test plan and later also using the user acceptance tests emailed out by Bernie, as well as any other creative tests they could think of. I feel that a large part of any successes we made is a down to them.

As we were on such a tight schedule, it was imperative that feedback from the testers reached the programmers as quickly and efficiently as possible. To do this, whenever a bug was found with the system, it was written on paper under the name of the developer responsible for the code that caused it. Each developer then worked through each of the bugs while the testing was still running and attempted to fix each issue. The project would then be redeployed and testing would begin again from scratch. Any new or continuing issues would then be identified and reported.

This cycle continued until all implemented functional requirements passed. This system worked well because it allowed all members (QA's and developers) to be active on the project at the same time, meaning it became very efficient to weed out bugs.

Another technique that was applied to the project that I felt worked quite well was the use of a pair programming technique between myself and the lead server side developer. Because of the issues already outlined in this document, we were often forced to work long hours implementing and integrating the client side and server side of the system.

By working closely together, carefully discussing and occasionally editing each other's code, it allowed use to reduce difficulties getting the two components of the system to talk to each other. It also allowed us to brain storm effective solutions to problems encountered along the way and let us double check what we had each written. This was especially useful when writing SQL queries and JSON strings while sleep deprived where spelling and formatting are paramount.

In conclusion, I feel that the implementation of this project could have gone smoother and some aspects could have been better outlined/developed/analysed, but in general I am happy with the outcome. Furthermore, I would like to outline once again how despite the setbacks and design obstacles our team encountered with the project, I feel that all team members pulled there weight and that producing a system to the level delivered would have been impossible without the effort exhibited by every member group 17.

1.5 Silhab Csoma

When we first started this project, things looked optimistic, we would have a lot of meetings and started working on the initial design early. Everyone was friendly and we delivered most of the documentation in time with only minor issues.

Then, coding week came, and things got a lot more chaotic both for us and all of the other groups as well. Suddenly, the requirements for the project started changing, causing most of the coding we have done up until that point to need rewriting. The campus network would do the impossible and become subject to a major hardware failure, hindering our work and causing us to lose some of our progress. Despite all difficulties, everyone tried their best to deliver as much work as possible and we managed to deliver all but 2 of the requirements in the specification. We even did some bonus features like a help page, a feature to rename monsters and added some detail into the notifications.

In this project, I was a programmer for the straight Java side of the code, I also wrote the testing and some of the documentation related to my work. I helped with the server side as well, mostly with logic problems and some Java functionality. While the communication between Sam and Tux for JavaScript and server side Java was strong, the Java side could have been better. I feel like we could have avoided some of the rewriting of the code if I had received more specifics about how they wanted the two parts of the program to come together.

Still, I learned a lot about working together as a group as well as how to create a functional web based application. We fulfilled most of the requirements, if it wasnt for the complications during coding week, we could have certainly done even more.

1.6 Tom Reed

Role: QA Manager
For this assignment we where put into groups and assigned the task of creating an online game based around users owning monsters and being able to add friends in order to battle their monsters or breed with their monsters. During the assignment I was given the role of the QA manager. This role needed me to manage various people and ensure that the project met various standards that would need to be set in order to produce the best possible system.

Positives
During this assignment I felt that I worked well within the group, there was no conflict within the group and as a whole we all got on well with each other. I felt that because of this it was fairly easy to manage people within the group which is something I felt I did well. I also feel that I distributed various tasks to members of the group fairly and efficiently by giving the most competent person in a given area the task that fits their skill set the best. This was something that I tried to do for every task to ensure that the best possible outcome was produced. I also feel that I was able to complete tasks that I had been given on time and to the best standard possible.

Negatives
Allow this whole assignment was mainly positive I have some negative points. When I first started as QA manager I did not have much knowledge about QA. This meant that I had to do a lot of research and as questions os that I could fully understand what was needed of me for this task. I also feel that my lack of programming knowledge hindered me during this task as I did not fully understand the back end Java of the system that was developed this again meant that I had to ask a lot of questions and ask of explanations of that was going on.

What I would have done differently
If I where to do this task again I would concentrate more of overall time management for the group. Although we met every deadline I still feel that we would have managed our time better and been more efficient. I also feel that I could have been a bit tougher and assertive as a manager, more demanding with deadlines .etc. I also thin that I could have spent more time planning and researching the standards and other plans that where set out in order to use our time as efficiently as possible. If I where to do this task again I would defiantly prefer to do a more hands on role as appose to a more overseeing role that I had in this instance. Although I did work

hard at this role and appreciate that experience that I have gained from it I feel that a hands on role is more enjoyable and rewarding.

Summary
Overall I feel that I preformed well in this task. I managed people well and completed task assigned to me on time. Although there where some areas that I struggled with during this project I feel that I over came them. I think that I have learnt a lot from this assignment,working with a medium sized team, managing people, time management for a project, working with new people. Although I would have preferred to have done a more hands on role I did enjoy my role as QA manager and worked hard to understand the role and also to produce the best result.

1.7 Chris 'Tux' Lloyd

On Monday evening of coding week, the coders were made aware that the requirements document had been changed. Breed and buy were no longer operating on a request accept/deny system, but instead an offer system. Due to this it meant that extra fields needed to be added to the monster table in the database. Because of the alteration to that database, it meant that all code relevant to the monsters was now invalid. This meant we could no long do anything as the backend would crash on login due to the fetching of monsters, couldnt even create a new account, as a monster was generated and added to that new account. All this needed to be fixed before I could start to rewrite the breeding and buying. I feel that someone should have really informed the coders before coding week.

Sam and I worked well as team and paired programed for the bug fixing as I handled all the back end and Sam did the java script allowing both of us to get short breaks while still thinking and working on solutions to issues. The documenters also proved to be invaluable to the coders as they would run a full requirements check and the let us know about the issues that came up and Sam and I would work though them. Sam was a big help in keeping me from breaking down and focus on the task at hand.

I had to rewrite some of silhabs classes so that it worked with the rest of the code. More effort should have been put into getting those 2 parts of code to work rather than me doing a rewrite. Silhab did make good use of time and write the joint test in a black box environment that helped with the testing process.

The IS server outages caused mayor disruptions to our work flow as we were not able to make commits to git or deploy to my tomcat server. Also not having internet at my house meant that Sam and I spent hours waiting for the connection to return to commit our changes so our latest work would not be lost. Also the M-drive outages were problematic as that is where my eclipse workspace was pointed, this was an issue as my pc was the one that Sam and I were programing on.

1.8 Mike Steel

From the outset our project team has been clearly organised into one group who are more comfortable with code and one group who are more comfortable with documentation, this has been helpful during each stage of development, as it was defined early on who would be best at each task. By giving each documenter a specific part of the code to document it allowed them to get to know the person working on that part of the code, as well as get to grips with how that part of the program would function and allow for documentation. As you can probably tell from what I have mentioned so far, I was a member of the documenting team, and had little part in the actual coding of the program, however, even though the documenters and the coders worked relatively seperately on the different parts of the project, when we needed to consult with each other over designs and the like we worked together well, and as a whole the group itself worked together well as a single unit completing each necessary part of the project.

By the time it came to implementation and testing week the group were used to working together enough that the final implementation of the project and especially the testing worked out very well. Most people turned up when required so that documentation was completed between testing phases, and the optimum amount of work was done for the time that we had. If there had not been the network problems and last minute project specification changes then we would have most certainly finished all functional requirements of the project, and it is a shame that those problems that we encountered were out of our control. Overall, the final outcome of the project is something that I am very happy with, and would work with the same group again any time. We all did what needed to be done, and helped each other out as well as included everyone in decision making.

I have enjoyed this project very much, and even though at times the work load has been more than I was expecting I know that some group members have worked on this project for far longer and harder than I would ever be able to make myself do, and I am truly impressed by some of the levels of work that some group members have put into this project. The experience has certainly prepared me for future group work, both in future university projects and in the world of work. This project has made me a more able group member, and has allowed me to experience every stage of an industry-worthy project. Therefore, I am now a more competant and experienced worker, and ready to do more work within groups in the future.

2 Management Summary

This project has managed to achieve 25/27 of the functional requirements it was aiming to complete. See (cite requirements doc) for list of functional requirements. The only two that were not managed to be completed on time were, monster aging and server to server. The mosters ageing requirement, although not implemented, there was a formula that we had that worked. Given a bit longer on the project I believe we could have got the monster ageing working. The server to server side however was a bit further off. Silhab Csoma had a look at the server to server but due to the change in requirements and internet down time there was not time to implement server to server.
All the documents have been worked on thoroughly and I feel that they are up to a more than reasonable standard.

A few difficulties got in our way during the development of the system. One of these issuses was the change of requirements. Originally it was specified that the monster battle and breed would be offered up or requested but as the deadline approached the functional requirements were changed to specify that breed and battle requests had to be offered. This meant that part of the system had to be redesigned which took time off working on bugs and other functional requirements.

Another major issue that arose was that there was internet downtime in the week before the deadline. This caused several issues, the first of which was the inability to test the system properly. This lead to work on the system being slowed down and thus resulted in a less complete system. It also meant more two of the coders(Sam Jackson and Chris 'Tux Lloyd) had to put in more hours to complete what was necessary to get a functional and bug free program. The second issue the internet downtime caused was a loss of work as when uploading the latest copy to git there was an error in doing so. We then tried to access the project which was stored on the M: drive however, this was unaccessable. We decided to leave the university grounds and head down into town to find somewhere with a working internet connection. Here the two coders managed to get access to the temporarily back up internet and M: drive and obtain a copy of the latest code. From here it was then uploaded to GitHub with an issue that meant the latest CSS code would be Lost. This meant Aleksandra Badya had to put more work into finishing of the design of the system.

The team preformed outstandingly overall. However special credit has to go to Chris 'Tux' Lloyd and Samuel Jackson as they worked perfectly as a team. Between them they managed to sort out a large quantity of the bugs and get a large chuck of the system operational. The rest of the team fulfilled there roles and more, they managed to do what was asked of them and more.

3 Historical account of project

The main events of this project were as follows; The first main event that happened was the meeting of the group. This is classed as a main event as we did not all know each other so we decided to meet up and get to know one another so we would find it easier to work with each other. This proved a good idea and thankfully every team member seemed to get on well with each other. We also assigned tentative roles to team members.
Next we started work on creating the system and project plan document. During this document creation we had weekly meetings and regular reviews of the document. The system was also starting to take shape at this time with parts started to be developed.
After the project plan document we moved onto the test specification document and designing the web interface. Again with this document we also had weekly meetings and reviews of the documents being produced. As well at this point in the project team members were getting more familiar with using LaTeXto standardise documents and github for version control. System continued to develop at this point.
Work then began on the design specification document for the system and getting a prototype ready for demonstration. The whole group worked briliantly on documenting, testing and implementing all aspects of the system. After this we worked on improving the documents and the system up to a standard that would be ready to hand in.
The final stage of this project was 'testing and implementation week' where the project was developed into the best state it could be. Here Samuel Jackson and Chris 'Tux' Lloyd managed to get more or less the whole system working even after internet down time cause issues with the project as well as a change of functional requirement. The documentors helped by testing the system and reporting any issues/bug so they could be fixed. Then the final document was produced which is a combined working of all the documents that have been worked on.
The team worked very well with each other communicating when needed and completing tasks set for them.

4 Final State of project

The final state of the project is that 25/27 of the functional requirements were fulfilled. The only two requirements that were not fulfilled were the monsters ability to age and the ability to connect server to server.

This should give a summary of which parts of the project are perceived as correct and which are not. It is as well to be as accurate as possible here - more marks will be deducted for problems that are not declared but are detected by the markers than for problems that are declared in the final report. As well as missing or erroneous features in the software, known problems with documents should be included here.

5 Performance of each team member

5.1 Tom Reed

Tom's duties as Q.A. Manager were to manage other documentors and delegate documenting tasks where he felt appropriate. He preformed brilliantly especially as the deadline approaced, he assigned tasks well and made sure they were completed. He also kept Christopher Marriott(Project Manager) up to date on goings and asked for more work when finished allcurrent work. He was an worked brilliantly in the team and helped the dynamic of the group flow.

5.2 Chris 'Tux' Lloyd

Tux preformed outstandingly. His time and effort put into the code was brilliant and he worked well with the rest of the team. He informed people of changes and explained things clearly. He worked especially well in tandem with Samuel Jackson. He put countless hours of work into the project and helped solved numerous bugs. Tux worked on the server side of things, getting the connectivity working with Samuel Jackson and Silhab Csoma side of code.

5.3 Samuel Jackson

Sam's preformance was outstanding. Not only was he vital in the functionalality of the overall system but on occasion preformed as a brilliant project leader when Christopher Marriott was away. His main role was to work on Javascript for the system. He helped get the system up and running and helped keep the team on track with their roles. He also put in a large amount of time into the project during coding week.

5.4 Silhab Csoma

Silhab worked on data structions and worked on algorithms this was essential to the working of the system. Without these in place there would be no way of storing monster and user info. Also the battle requirement would not be completed if it was not for the algorithm designed. Her testing on parts of the system helped other team members progress in their work. She worked well in a team and completed all tasks set.

5.5 Alexsandra Badya

Alex worked very well in the team. She provided vital support to the team with git support, whenever there was and query or issue she was able to help resolve the problem. Here main role however was the creation of HTML and CSS for the system and in this role she performed very well. She also worked well as deputy Q.A. manager when Tom Reed was away. The team were kept up to date with duties they had to complete. She performed very well on the whole project.

5.6 Dave Clark

Dave was a very good documentor in all areas. He mostly focused on Tux's server side documentation however he also work on other documentation. He also helped find bugs with in the system and inform people of what they were. Other areas he worked on were the database diagram, methodology and reviewing functionality. He also did the minutes for several of the team meetings.

5.7 Mike Steel

Mikes main duty was to document Silhab's work. He performed well within the team and helped in other areas. He helped out on the testing side of the system and fixing the LaTeXdocuments. He also produced the risk assessment and designed the monster mash logo that we use on the main system webpage. He helped update documents and fix any mistakes with them.

5.8 Matt Whitmore

Matt's job was to document and test Alex's HTML and CSS. He completed all tasks on time and worked well with the team. He also helped out on the testing of the system and found some bugs of which he informed the appropriate people.

5.9 Christopher Marriott

Chris' main role was project manager. As project manager it was Chris' duty to assign tasks to other team members, review documents and check that people are completing their tasks. He communicated to the group regularly and organised meetings, he also on occasion took the minutes. He also helped out on the creation of documents and bug testing. He also got people to start using LaTeXto help standardise documents.

6 Critical evaluation of the team and project

6.1 Team performance

The teams performance was brilliant. A few team members also went above and beyond the call of duty to get the system more or less fully functional. Everyone knew their role and were happy with the tasks they were set. If anyone had a problem they spoke to the appropriate person and got it resolved reasonably quickly. Everyone did their roles excelently and fulfilled all that was required and more. Each member of the team communicated clearly with other team members and kept up to date with what had been done and what had not. Each team member was kept up to date and informed of new tasks via a facebook page and a gannt chart. Everyone seemed happy with this approach and there was no major disruptions from within the group.

6.2 Improvements

The project could have been improved in a few ways. The first way would have been to assign Samuel Jackson as the project manager as he was more knowledgeable in the system and would therefore been able to assign more details tasks. The second would have been for Christopher Marriott(Team leader) to have informed the team of new tasks that needed doing and keep up to date with where team members were at with their current task.

6.3 Lessons learnt

Some of the lessons that were learnt during this project were to make sure people understood their role more clearly. To plan the more important and required parts of the system before worrying about how it will look. How to code alongside others, how to manage time more appropriately and skills in being a leader.

REFERENCES

[1] *N/A*

DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2013-01-30	Initial creation	cpm4