Monster Mash – Team Awesome-er
Project Plan

Author:	Joshua Bird, Phil Wilkinson, Thomas Hull, David Haenze, Christian Morgan, Kamarus Alimin, Szymon Stec, Lewis Waldron
Config Ref:	SE_02_PP_03
Date:	02-11-2012
Version:	1.4
Status:	Final

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to describe the project plan for the Monster Mash CS22120 Group Project 2012/13.

1.2 Scope

This project plan aims to set general design goals, high-level software decisions and goals to work towards, given the client's requirements. All project members are required to read this document to gain an understanding of the project parameters and goal solutions.

1.3 Objectives

The objectives of this document are:

- To indicate the general overview upon which the development of the project is based upon. This should include the choice of platform that the project will be developed on, technologies under-pinning this and the project's target users;
- to explain how users can interact with the system via a use-case diagram;
- to give a proposed GUI design of the completed project through interface sketches and drawings;
- to give a general time-line spanning the course of the project and proposed start/end dates for different aspects, via a Gantt chart;
- to analyse potential risks that may appear during the development of the project and propose solutions that may help mitigate such risks, by drawing up a risk table.

2. PROJECT PLAN

2.1 Overview

We have been asked to design a “Monster Mash” game in which users can fight, trade and breed their monsters over a network with other users. The users will need to create an account first before they can start playing, and in order for them to return and carry on playing will need to enter their username and password to authenticate themselves as the users. There will also be an element of monetising for the user's to be awarded after winning a fight, and in turn the user can pay for breading partners or new monsters. The monsters can also die in fights and of natural causes. In the game the user will have the ability to search for friends on the game and add them to a friends list, to enable the user to pick fights, or pay for breeding of one of their monsters. The user will only be able to interact with friends on their friends list, their friends will be identified by their email address and be added by a request confirm mechanism.

The game is supposed to be designed as an educational tool to popularise and inform people of evolution and genetics. As this game is supposed to be able be played across a network, we as a team decided on using GlassFish servlets in the developing the game, on Java JDK 1.7. This will provide storage and a communication interface. Additionally, we used GlassFish because the others were very limited in what you could do; e.g. the Google App engine, there were far less packages to use, therefore it prevented us from doing certain things.

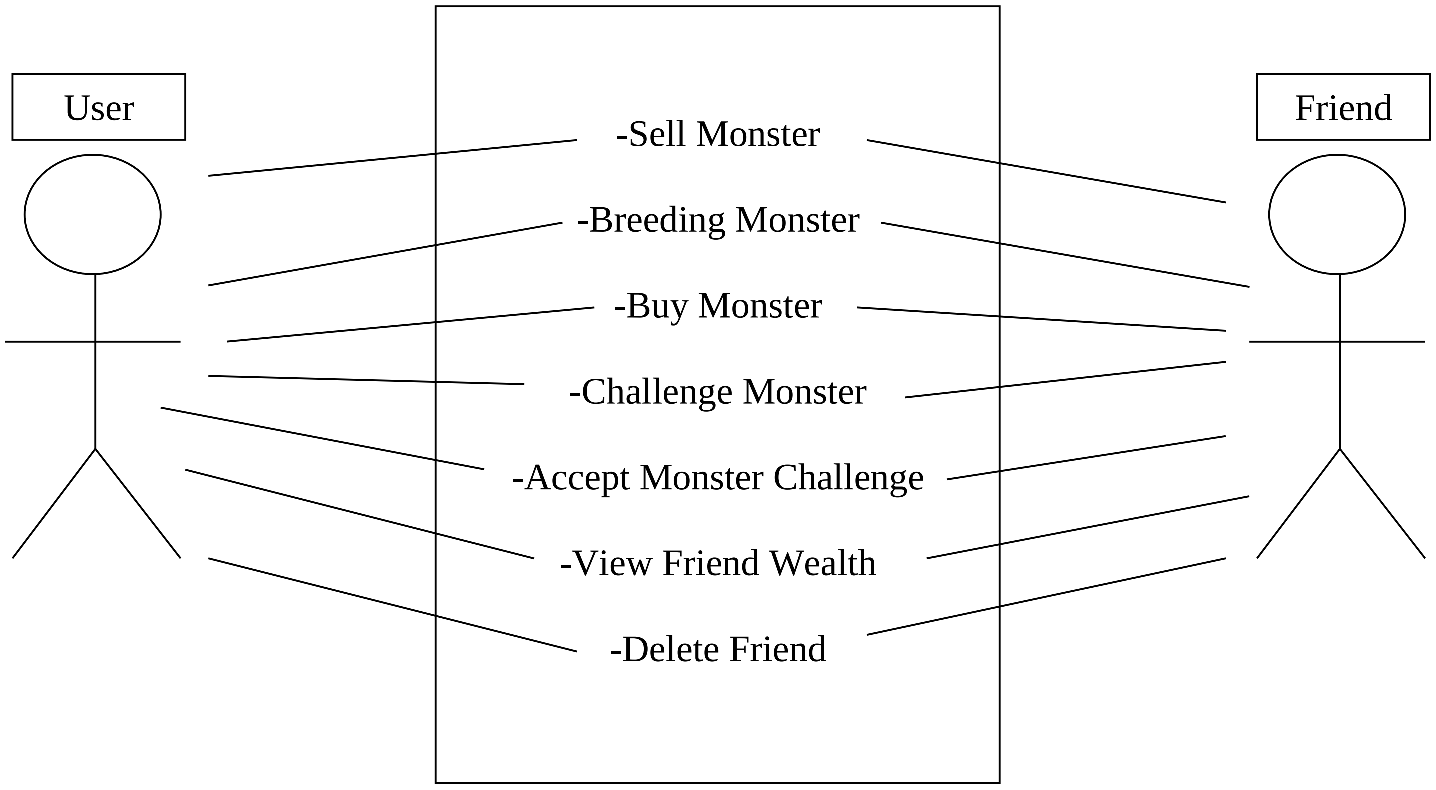
We also have to be able to connect to other users and be able to interact with them and their game. We will standardise the protocols to allow other games on different clients to interact and communicate efficiently. The server will be using the Java Database to store all of the user’s information, as user’s name, their monster’s characteristics and attributes, and who they are friends with in the game. The Java Database will be able to manage all the registered users, allowing new users to be added, and details amended.

We decided on using the Java Database because it is integrated within Java and XML, unlike MySQL which we thought might be more difficult to implement. As this is a web-based user interface requirement we will be using XHTML along with various Java scripts to aid it. The demo-graph we are aiming for with our users will be young people from the age of 14-23 who play web-based games who are interested in monsters or evolutions.

We shall be using a high-level architecture, in which the web page will send and request information via HTTP from the GlassFish servlet. From here the servlet will then communicate with the Java class packages. These packages will then save and load from the Java database which will be using a XML Template, and shall store itself as a XML files. From our discussions we have decided to use GitHub as a source code repository, in order to make sure that we have a clear view on who has updated parts of the project. GitHub supports ticketing which will be useful. With this feature the group leader is able to give out assignments, flag important tasks that need to be completed and can set deadlines. GitHub also comes with managing features, which can limit some team members to only be able to read certain files, and allow others to write to these files - this can be useful in limiting accidents. GitHub also comes with the ability to create Wiki's for the project.

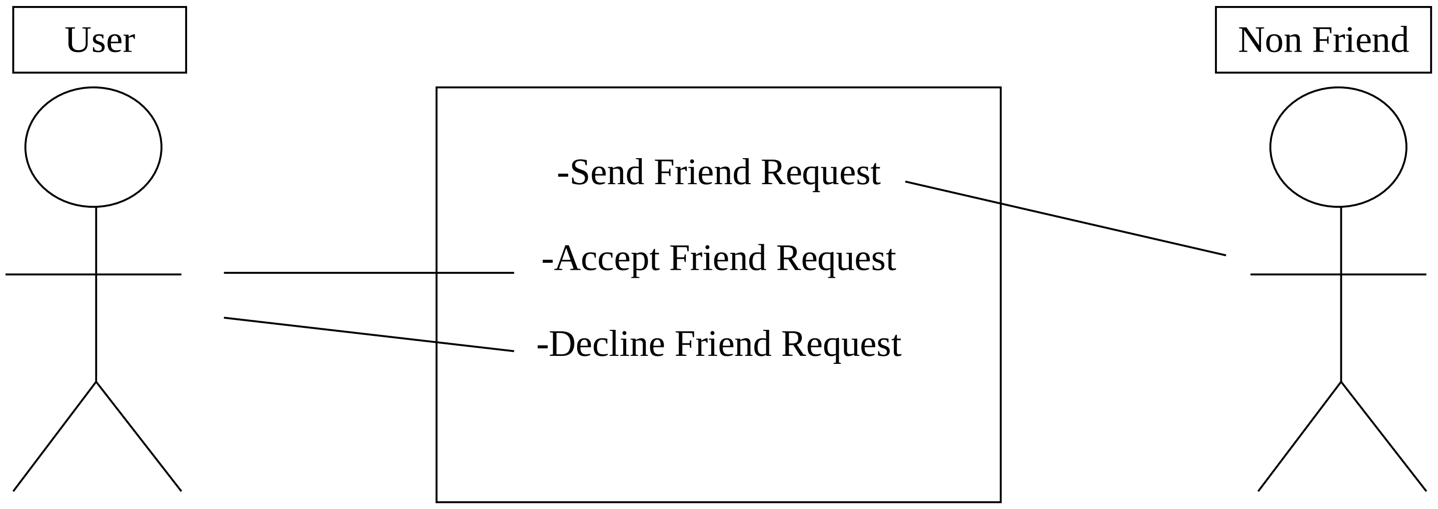
2.2 Use Case Diagram

2.2.1 Use case friend diagram



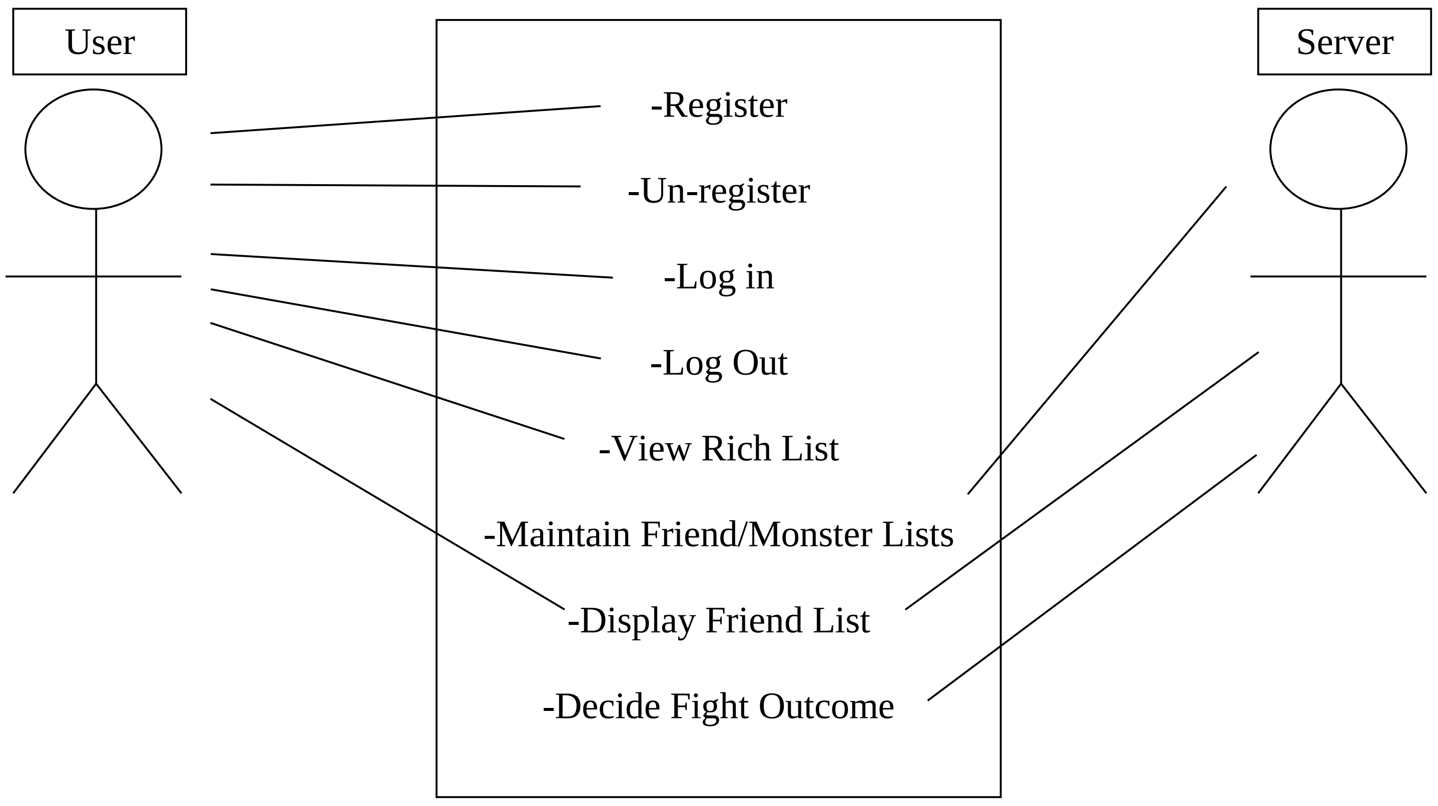
The above use case diagram shows the interactions that users and friend. The both users can sell their monsters to other users. The can also breed monsters with friends monsters, and produce offspring. The users monster can fight other users monsters by challenging each other, this is done by a send request system, were the other user has to accept the challenge. Another feature you can view friends wealth, and users can delete friends.

2.2.2 Use Case Non Friend Diagram



The above use case diagram shows the interaction taken place when users send friend request to each other, and what options the user can take. The user can either accept or decline friend request.

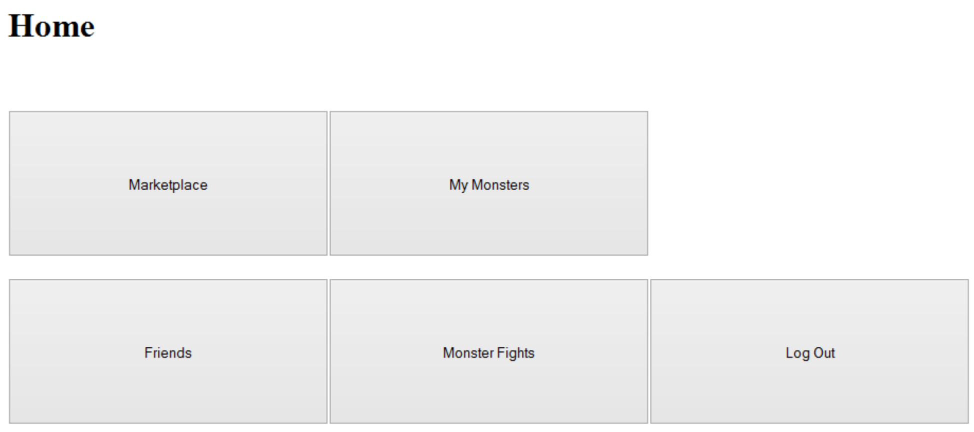
2.2.3 Use Case Server Diagram



The above use case diagram shows the interaction between the user and server. The user can register to play the game. The can also un-register if they wish to, they can also log in and log out from the game. The user can also view the rich list, which shows the wealthiest players in the game. The server also maintains the friend and monsters list in the game. The user can also display friend list while the sever can also decide fight outcome between two monsters.

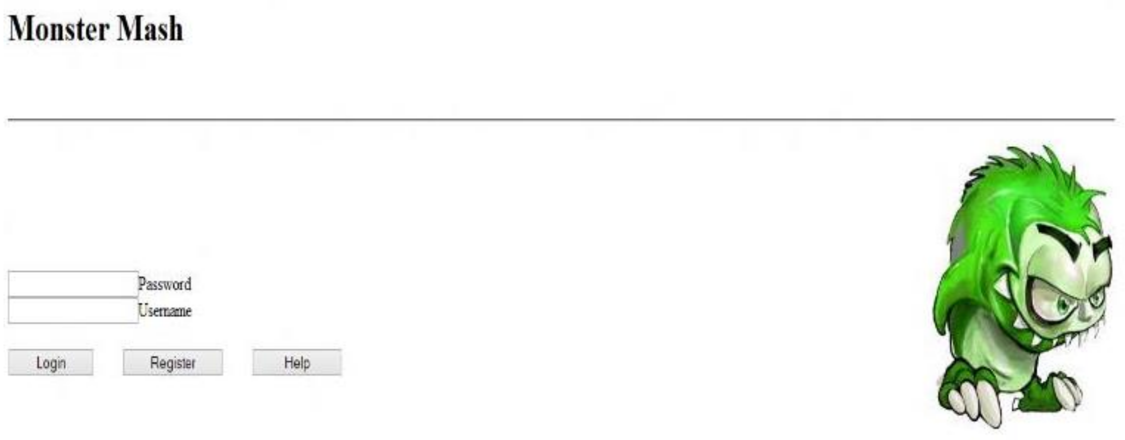
2.3 User interface design

2.3.1 Home screen



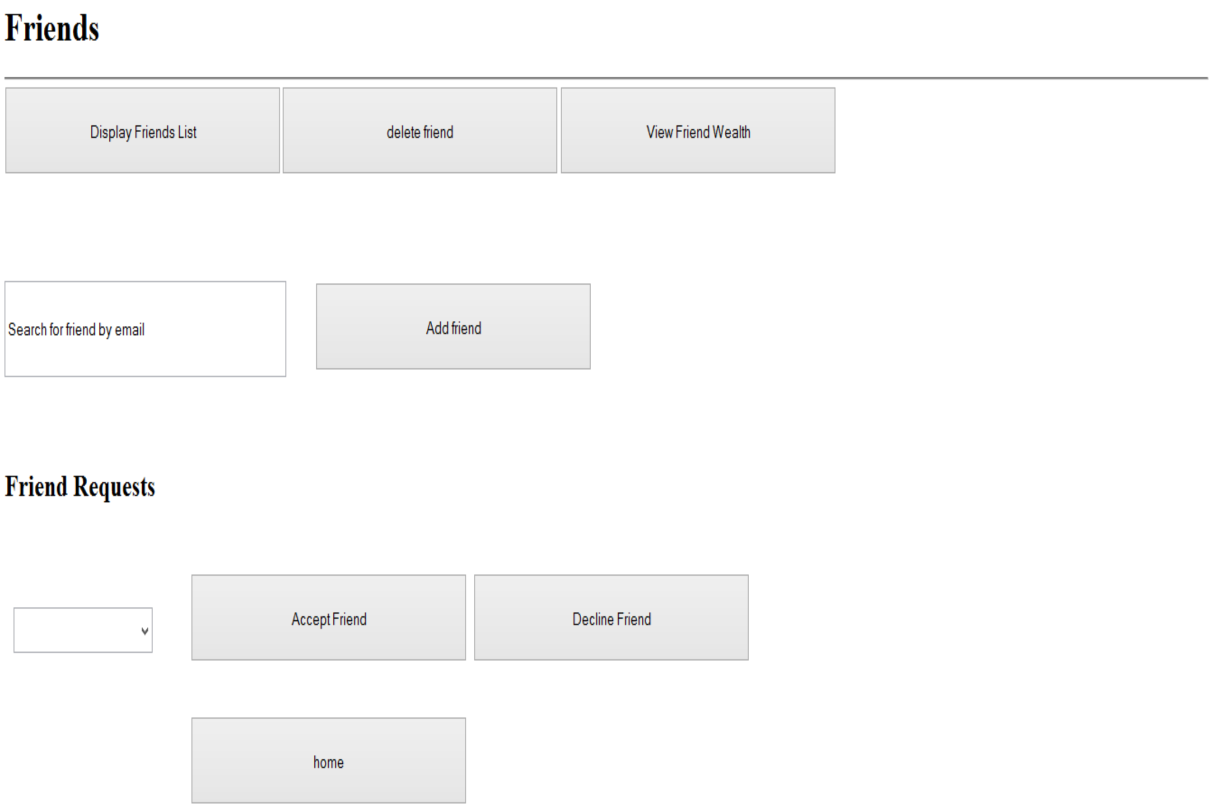
After logging in with the correct credentials, the next page to the user will be the home page.
Each button will take the user to the following:
Marketplace > marketplace.html (where monsters are bought and rented)
My Monsters > myMonsters.html (where one can breed, sell and loan monsters)
Friends > friends.html (used to see and delete current friends as well as add new ones and manage friend requests.
Monster Fights > monsterFights.html (To challenge monsters of other friends)
Log Out > login.html (Takes the user back to the log in page.)

2.3.2 Login screen



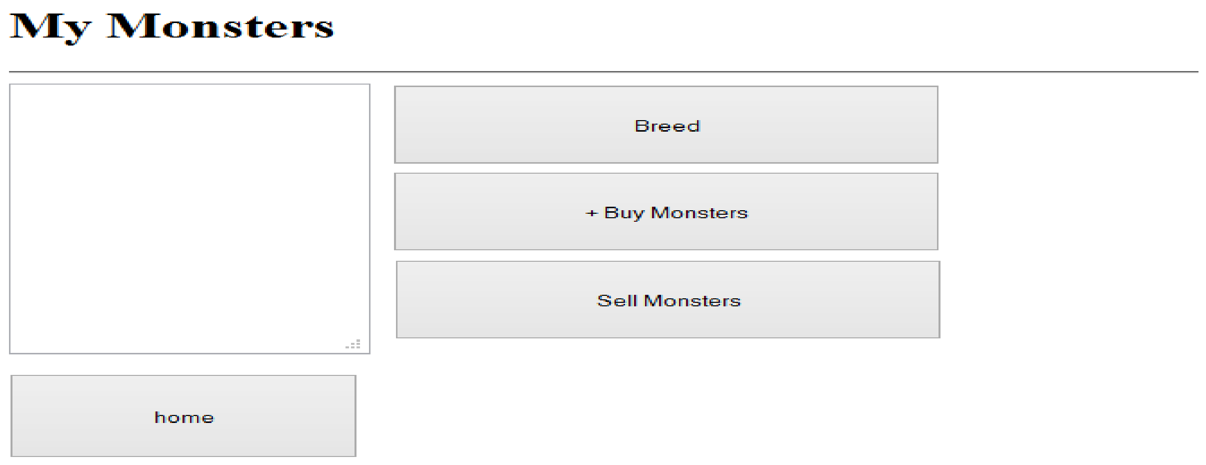
When the game is started the first page the user should see is the log-in page. As the user can see here, one can log-in with their correct credentials. If the user were not to have any, they can register (or view the help page).

2.3.3 Friends



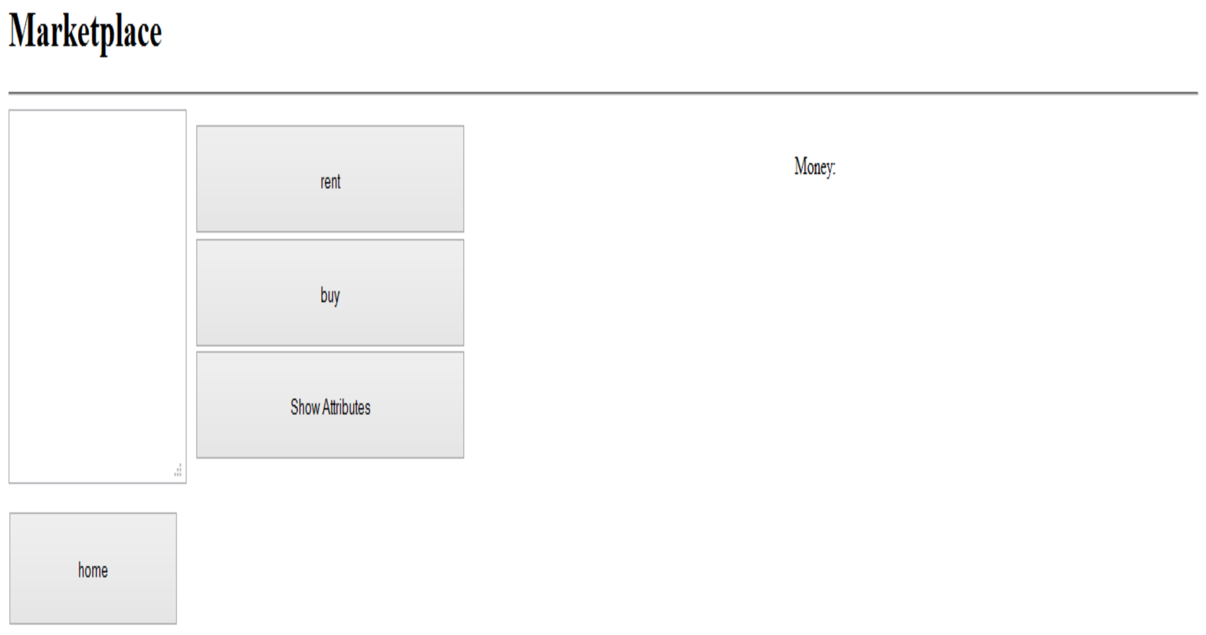
After clicking the ‘friends’ page the following page will be displayed. In this page one can manage their current friends by deleting and adding them. The ‘display friends list’ will bring up a list of all currently connected friends. ‘View friend wealth’ will show a rich list of the richest players in terms of money in the game. By entering a valid user email address users will be added via the ‘add friend’ button. Friend requests will show in the drop down box towards the bottom left of the screen shot. By selecting the appropriate request it is easy to see how the user will accept or delete the friend request. As per usual the home page takes the user back to the home page.

2.3.4 List of user’s current monsters



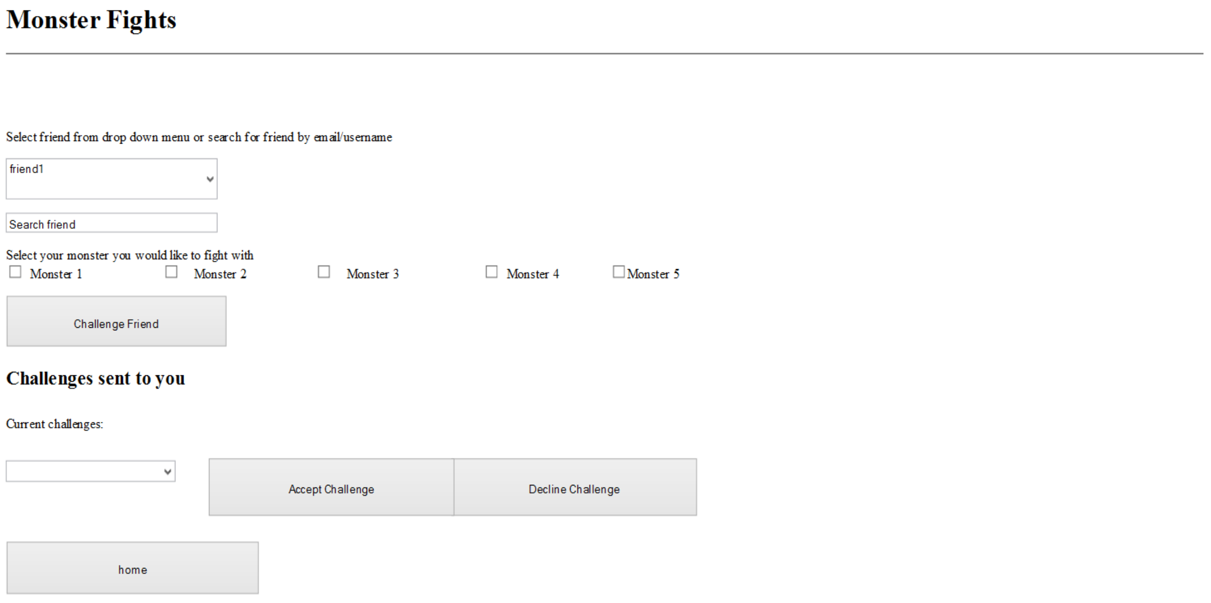
After clicking the ‘my monsters’ button on the home page the following page will be shown. A user will be able to breed a monster, as when the button is pressed, a prompt box with a drop-down box of your monsters will be shown, enabling the user to choose which monsters to breed. The ‘buy monsters’ button will take the user back to the marketplace page. As for the ‘sell monsters’ button, a prompt box will display, enabling the user to specify whether they want to sell a monster or loan it out to another user, and for how much money. The home button takes the user back to the home page.

2.3.5 Buy a new monster



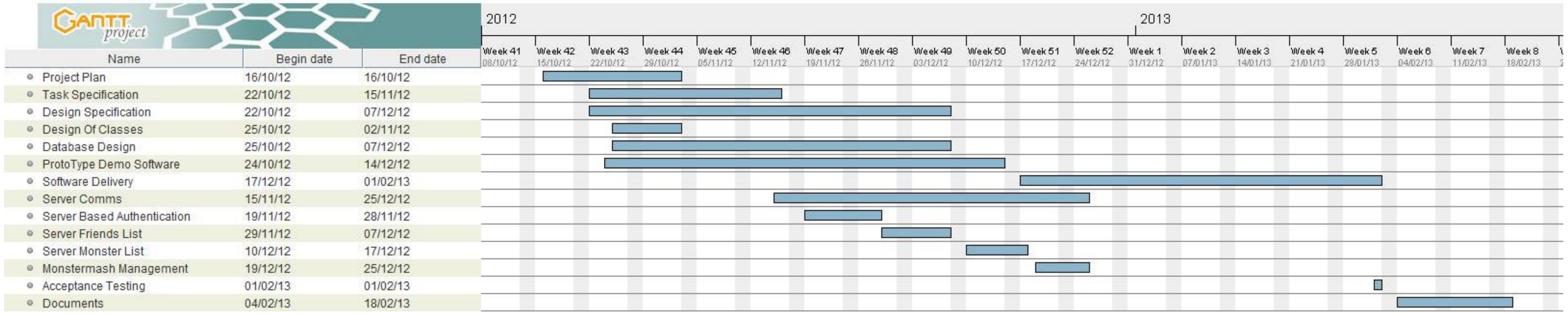
After clicking the marketplace button on the home page this page shall load. There is a list box on the left-hand side which will display monsters that other friends are selling or loaning out. After clicking on the monster, the user shall be able to rent or buy them by clicking the appropriate buttons. The ‘show attributes’ button will tell the user more general info about the monster. At the top right corner of the screen where it says “Money:”, this will show the funds available to the user. The home button will take the user back to the home page.

2.3.6 Fight a monster



Via the ‘monster fights’ button on the home page this page shall be reached. To challenge a friend the user can either search for them by their email, or select them from the drop down menu. The user will then select which monster to fight with and click ‘challenge friend’ - a pop up box will then display the fight outcome. Challenges sent to the user can also be managed in a very similar way to friend requests. Current challenges are displayed in the drop-down menu, and with the appropriate buttons, one can accept or decline the challenges.

2.4 Gantt chart



Name	Begin date	End date
Project Plan	16/10/12	16/10/12
Task Specification	22/10/12	15/11/12
Design Specification	22/10/12	07/12/12
Design Of Classes	25/10/12	02/11/12
Database Design	25/10/12	07/12/12
ProtoType Demo Software	24/10/12	14/12/12
Software Delivery	17/12/12	01/02/13
Server Comms	15/11/12	25/12/12
Server Based Authentication	19/11/12	28/11/12
Server Friends List	29/11/12	07/12/12
Server Monster List	10/12/12	17/12/12
Monstermash Management	19/12/12	25/12/12
Acceptance Testing	01/02/13	01/02/13
Documents	04/02/13	18/02/13

2.5 Risk Assessment

Risk assessment is the act of determining the probability that a risk will occur and the impact that event would have, should it occur. This is basically a “cause and effect” analysis. The “cause” is the event that might occur; while the “effect” is the potential impact to a project should the event occur.

Risk No.	Threat	Risk Description	Mitigate Actions
1	Aberystwyth’s servers crash	Machine crashes, losing the project work.	Mitigated by backing up after work with a pen drive or other storage device. While also keeping records on ‘GitHub’
2	Health risk	Lead coder or one of the members falls ill.	Mitigated by setting up regular meetings between other coders to explain the current progress. Also well documented code so other coders can understand what is going on with the project.
3	Project deadline	Work not completed on dates set.	Have target deadline set on the week before “official” deadlines.
4	Copyright	Copyright infringement or violation of the copyright law. For example, the use of an image which has watermarking embedded download from the internet.	Mitigated by obtaining the author’s permission or give appropriate credit to the author. Or by finding license free images.
5	Sabotage by another group	Project might be sabotaged by another group due either pranks or malicious attack	Mitigated by seeking the higher authority for the action to be done
6	Communication barrier	Miscommunication among member might slow down the process of the project work.	Use Appropriate channel of communication or medium to convey the message better.
7	Act of nature	Unforeseeable event that is outside our control, no one held responsible for what have been done.ie Natural event such as hurricanes, apocalypse and etc.	Unfortunately, no mitigation possible.
8	Rescheduling Informal meeting	If a member unable to attend to an informal meeting due to lectures, social activities and sport activities.	The project leader should coordinate a common free time that everyone will agree to attend.
9	Some or all group members are not used to group projects	Some people are unaccustomed to working in a group and feel uncomfortable participating or initiating group discussion.	Mitigate by assigning roles to each member in which they feel responsible for the project. Besides that, an ice breaking activity such as going to a pub etc. on the weekend is a good way of getting to know each member.
10	If a group member withdraws from the course	This could lead to chaos and management problem within the project group. Illness, accident or family obligations are some of the possible reasons for the withdrawal.	Seek the lecturer assistance to adjust scope of the project. Project leader should reorganize roles and task to compensate for the loss of manpower.
11	Wet Floor	If the floor has been cleaned in the university campus and a member trips and causes an injury.	Mitigate by being observant for wet floor signs while around the university campus and also as mentioned previously have other members with knowledge of the project to replace the member if needed.
13	Work being copied	If a member is in a public place working on the project someone could watch them and copy work.	Members should be observant while working on the project in public places and try and make sure sensitive data isn't lost to others.
14	User not familiar with tools	If a member does not know how to use GitHub or an IDE we are using to work on and is unable to do the work because of it, this could lead to a loss of work done.	Any group member unsure how to use any of the software should ask for help from others.
15	More than one user trying to update a file at the same time	If more than one user edits a file on GitHub and tries to save, someone else's work could be overwritten.	Making use of GitHub's ticketing feature to be able to keep track of updates to files.

16	Stress	Any group member could have their work affected by stress, be it work related or otherwise.	Those being effected should notify the project manager or seek help from the university in order to deal with it. The project leader can also be spoken to if the member feels like they have too much work to do, so that it can be reviewed.

REFERENCES

[1] Software Engineering Group Projects: General Documentation Standards. C. J. Price, N. W. Hardy. SE.QA.03. 1.5 Release

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	20-10-2012	Created Initial Document	PW
1.1	N/A	25-10-2012	Added draft versions of sections to the body	JB
1.2	N/A	02-11-2012	Updated to latest versions of sections.	JB
1.3	N/A	28-01-2013	Updated use-case	CM
1.4	N/A	28-01-2013	Updated gantt chart	PW

Monster Mash – Team Awesome-er
Test Specification

Author:	Joshua Bird, Phil Wilkinson, Tom Hull, Dave Haenze, Chris Morgan, Kamarus Alimin, Szymon Stec, Lewis Waldron
Config Ref:	SE_02_TS_04
Date:	14-11-2012
Version:	1.2
Status:	Final

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to outline how the design specification/program will be tested against the tests in the test specification table.

1.2 Scope

The test plan aims to set standardisation for the testing of the program, and parameters in which the program will be tested to. All program testers are required to read this document to understand the guidelines they will follow when testing the program.

1.3 Objectives

The objectives of the test plan are to see whether the features in the program work, given different variables. The testing will be done in the following way:-

- the first column will contain the test reference;
- the second column will contain the requisite being tested;
- the third column is test content which will give a short description of what is being tested;
- the fourth column will contain the input, i.e. what action will be done;
- the fifth column will contain the output after the action has taken place;
- the sixth column will contain the pass criteria i.e. how the test will be passed.
- the seventh column will contain the Test criteria i.e. to determine if that each test cases is a success or failure

2. APPROACH TO TESTING

Testing is a fundamental component of the software quality assurance and represented a review to meet the requirement as specified in the document QA Document SE.QA.01. The technique used to test is the black box testing techniques, where we perform functional testing to ensure that the functionality meets the requirements. In this approach, we focus on determining whether or not a program does what it is supposed to do based upon its functional requirements. Black box testing attempts to find errors in the external behaviour of the code (output) which is grouped into the following categories ; incorrect or missing functionality; interface errors; errors in data structures used by interfaces; behaviour or performance errors; and initialization and termination errors. Through this testing, we can determine if the functions appear to work according to specifications. The main black box testing tools used are the record and playback tools - the IDE used to test the system against these requirements is Netbeans. Black-box testing focusses on the inputs and outputs of the software without knowing their internal code implementation .The following is the guideline that should be followed during the testing phase.

1. Conform to the requirements and specifications that is shown and set in the QA documents
2. Conduct a number of test cases by choosing valid input to check the positive test scenario and invalid input to check the negative test scenario of the system
3. Tester should determine the expected output from the input given.
4. Using the specified inputs, check that they match the expected output/result
5. Chosen test cases are executed and documented for the outcome
6. If any errors occur, it should be fixed if possible
7. Test table is drawn up to show the number of tests conducted and whether they passed or failed.

3. TEST SPECIFICATION

Test Ref	Req. being tested	Test content	Input	Output	Pass Criteria	Test pass/fail?
SE-N02-001	FR1 + FR6(register)	Test whether the “Register button” on the login page will navigate the user to the register page	Click the register button first - the button on the left on the login screen	A page which consists of forms and buttons will be shown on the user’s browser	The page should be load correctly	Pass
SE-N02-002	FR1 + FR6(register)	Check if the system does not allow adding already existing user	Enter an user email that already exists in database	Account is not created, user gets error message	System displays "User already exists" error message	Pass
SE-N02-003	FR1 + FR6 (Register)	Check if the system does not allow adding new user if all required fields are not filled	Leave all the fields blank	Account is not created, user gets error message	System displays "You have to fill all required fields" error message	Pass
SE-N02-004	FR1 + FR6 (Register)	Check if the system does not allow using restricted characters	Enter special characters (^%#) in email section	Account is not created, user gets error message	System displays "Incorrect input" error message	Pass
SE-N02-005	FR1 + FR6 (Register)	Check if system creates new user account	Enter user email and all required data correctly	List of stored users should now include this user.	Data should be stored correctly	Pass
SE-N02-006	FR1 + FR6 (Unregister)	Check if system deletes user data from database	Logged user clicks “Unregister” Option	List of stored users should not include this user. User is redirected to login page	Data should be remove correctly	Fail
SE-N02-007	FR6 (Unregister)	Check if system deletes user data from database	Logged user clicks “Unregister” Option	Could not remove the user	System displays "Some problems occurred, please try again or contact administrator " error message	Fail
SE-N02-008	FR2 + FR6	Check if the “friends button” on the Home page navigates the user to the ‘friends.html’ page	Click the friend button on the home page	The ‘friends.html’ page will be shown on the user’s browser	The page should be load correctly	Pass

SE-N02-009	FR2 + FR6 (Add Friend)	Check if system does not allow to add non existing user	Enter user email that does not exist in database or leave it blank	Friend should not be added	System displays “Sorry, user does not exist” error message	Pass
SE-N02-010	FR2 + FR6 (Add Friend)	Check if user is added correctly if request accepted	Enter friends email correctly and person accepts friend request	Friend should be added correctly	Friend is added to user’s friends list	Pass
SE-N02-011	FR2 + FR6 (Add Friend)	From the test SE-N02-012, has the other user been notified with the respective friend request	Log-on to the other account which is about to receive the friend request	There should be a notification in the friend list page with the details of the email of the user.	The friend request should perform well	Pass
SE-N02-013	FR2 + FR6 (Add Friend)	From test SE-N02-014, we check if the user is able to successfully send two friend requests.	Send the two friend requests by using their email in the friend request page. After that, login to that chosen email account	Both account should receive the friend request	Adding more than one friend request in the friend page is successful	Pass
SE-N02-015	FR2 + FR6 (Decline friend request)	To test the decline button of the friend request session	From the test SE-N02-010 ,we test it out with the decline button	If the person is declined, that user will not appear on the friend request box.	Friend invitation should be performing well	Pass
SE-N02-016	FR2 + FR6 (Remove Friend)	Check if system allows removing friend without selecting any from the list	No friend is selected, use delete button	Error message should be displayed	System displays “You have to select a friend first.” error message	Fail
SE-N02-017	FR2 + FR6 (Remove Friend)	Check if friend is deleted correctly	Select friend from “friend list”, use delete button	Friend should be deleted correctly	Friend is deleted from user’s friends list	Fail Delete Friend function is unavailable
SE-N02-018	FR3 + FR6	Check if the “monster button” on the homepage will navigate the user to the monster menu page	Click the “monster button” shown in the homepage	The browser loads myMonster.html onto the screen	The navigation button should direct the user to the selected page	Pass
SE-N02-019	FR3 (New account)	To verify if a Newbie package such as virtual money, is given to new users after their registration	Create an account on the login screen	After creating an account, the system will automatically allocate virtual money to the new account	Newbie package system should works	Pass
SE-N02-020	FR4	Test whether the user monster can challenge another monster	We test out by using an admin account: search another valid account or search from the list of friends	The chosen user will appear from the drop-down menu	The monster challenge system should works	Pass

SE-N02-021	FR4	Check if the user can accept a challenge from another user	Use an account to invite another member to a challenge	There will be a notification from the homepage that another user wishes to challenge	Challenge system should works	Pass
SE-N02-022	FR4	Check if the user can decline a challenge from another user	Use an account to invite another member to a challenge and have that other member decline the challenge	There will be a notification from the homepage that other user declined the challenge	Declining challenge from the user should works	Pass
SE-N02-023	FR4	To confirm that the winner of the monster challenge receives their reward	From test SE-N02-017, which is another account accepts the challenge from the current account to test which has a more powerful monster	There will be a short video play on screen of monsters fighting while the system calculates the winner	The monster challenge should works	Fail No media is loaded
SE-N02-024	FR6 (Offer for sale)	Check if system allows selling a monster without selecting any from the list	No monster is selected, use sell monster button	Error message should be displayed	System displays “You have to select a monster first.” error message	Pass
SE-N02-025	FR6 (Offer for sale)	Check if system allows monsters to be sold	Select monster from “monster list”, use sell monster button	Monster should be added to the market list	Monster is added correctly to the market list	Pass
SE-N02-026	FR6 (Offer for sale)	Check if system allows monsters with no price to be sold	Set monster price to 0 or leave it blank	Monster is not added to the market list, error message should be displayed	System displays “You have to set a monster price first” error message	Fail
SE-N02-027	FR6 (Buy monster)	Check if system allows user with not enough funds to buy a monster	Select monster in market, use buy button	Monster is not added to users monsters list	System displays “You don’t have enough money to buy this monster” error message	Fail
SE-N02-028	FR6 (Buy monster)	Check if system allows buying a monster without selecting any from the market list	Select no monster in market, use buy it button	No monster should be added to users monsters list	System displays “You have to select a monster first” error message	Fail
SE-N02-029	FR6 (Buy monster)	Check if system allow to buy a monster from other user	Select monster, use buy button on the user	Monster should be added to users monsters list and removed from market	Monster is correctly added to users monsters list and removed from market	Pass
SE-N02-030	FR6 (Offer for breeding)	Check if system allows breed a monster without selecting any from the list	No monster is selected, use breed monster button	Error message should be displayed	System displays “You have to select a monster first.” error message	Fail
SE-N02-031	FR6 (Offer for breeding)	Check if system allows monsters with no price to be bred	Set breed price to 0 or leave it blank	Monster is not added to the market list, error message	System displays “You have to set a breed price	Pass

				should be displayed	first" error message	
SE-N02-032	FR6 (Offer for breeding)	Check if system allows monsters to be bred	Select monster from "monster list", use breed monster button	Monster should be added to the market list	Monster is added correctly to the market list	Pass
SE-N02-033	FR6 (Purchase breeding)	Check if system allows user with not enough funds to buy a monster	Select monster in market, use breed button	Monster is not added to users monsters list	System displays "You don't have enough money to buy this monster" error message	Pass
SE-N02-034	FR6 (Purchase breeding)	Check if system allows buying a monster without selecting any from the market list	Select no monster in market, use breed button	No monster should be added to users monsters list	System displays "You have to select a monster first" error message	Fail
SE-N02-035	FR6 (Purchase breeding)	Check if system allow to buy a monster for breeding	Select monster, use breed button	New monster should be added to users monsters list	New monster is correctly added to users monsters list	Fail
SE-N02-036	FR1 + FR7(Log-in)	Test whether the "login button" on the login page works – this will navigate the user to the home page	Entering a valid username with the password and clicking the login button	After the login process, the home page will be loaded onto the user's browser	The page should have been load correctly	Pass
SE-N02-037	FR7(Log-in)	Check if the system allows to log-in not providing any details	Leave blank user log-in and password	Error message should be displayed	System displays "Please, enter your username and password." error message	Pass
SE-N02-038	FR7(Log-in)	Check if the system allows not existing users to log-in	Enter new user email and password	User is not logged-in, error message should be displayed	System displays "Wrong username or/and password" error message	Pass
SE-N02-039	FR7(Log-in)	Check if the system allows user to log-in no password	Enter only username, leave password blank	User is not logged-in, error message should be displayed	System displays "Wrong username or/and password" error message	Pass
SE-N02-040	FR7(Log-in)	Check if the system allows user to log-in with incorrect password	Enter username and wrong password	User is not logged-in, error message should be displayed	System displays "Wrong username or/and password" error message	Pass
SE-N02-041	FR7(Log-in)	Check if the system allows user to log-in	Enter username and password	User should be logged-in	User is logged-in and redirected to the home page	Pass
SE-N02-042	FR7(Log-out)	Check if the systems allows user to log-out	Use log-out button	User should be logged-out	User is logged out and redirected to the log-in page	Pass
SE-N02-043	FR8(monsters list)	Check if the system loads the list of user's monsters correctly even though it is empty	Use "my monsters" button	System should show an empty list	System should be loaded an empty list correctly	Pass

SE-N02-044	FR8(monsters list)	Check if the system loads the list of user’s monsters	Use “my monsters“ button	System should show a list of monsters	System should be loading the list correctly	Pass
SE-N02-045	FR8(friends list)	Check if the system loads the list of user’s friends correctly even though it is empty	Use “friends“ button	System should show an empty list	System should be loading an empty list correctly	Pass

REFERENCES

[1] Software Engineering Group Projects: General Documentation Standards. C. J. Price, N. W. Hardy. SE.QA.03. 1.5 Release

[2] Software Engineering Group Projects: Test Procedure Standards. C. J. Price, N. W. Hardy. SE.QA.06. 1.6 Release

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	14/11/2012	N/A - original version	PW
1.1	N/A	09/02/2013	Approach to testing created and tests fixed	KA
1.2	N/A	10/02/2013	Fixed spelling/grammatical mistakes	PW

Monster Mash – Team Awesome-er
Design Specification

Author:	Joshua Bird, Phil Wilkinson, Tom Hull, Dave Haenze, Chris Morgan, Kamarus Alimin, Szymon Stec, Lewis Waldron
Config Ref:	SE_02_DS_04
Date:	07-12-2012
Version:	1.3
Status:	Final

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to outline the detail and description on various parts of the design for the group project. This document should be read in a manner that takes into consideration the details of the group project assignment, and the group project design specification standards [1].

1.2 Scope

This document is intended to give a breakdown of the group project design via various methods and descriptions, and to also give diagrams showing the interactions between the client and the server.

This document should be especially read by all members of the coding team and by the rest of the group, so they have an understanding of what is happening inside the program.

1.3 Objectives

The objective of this document is to:

- Give a description of the group project components from a high-level point of view
- Illustrate the flow of data between the client and server
- Depict a basic overview of the classes and how they are linked through a UML class diagram
- Give an understanding of the relationships and dependencies between modules in the dependency description
- List and give details of the interface classes, so that it can be used by other coders.

2. ARCHITECHTURAL DESCRIPTION

The monster program consists of three components:

- The DATABASE component
- The CLIENT component
- The SERVLET component

2.1 The DATABASE component

We will be using a Java Database which is a database management system (DBMS) that we use to store and retrieve information to/from a database. It is used to organize the data according to the subject, and it is easy to track and verify the data. It can store information about how different subjects are related, so that it makes it easy to bring related data together. We will create tables such as the user’s profile and their monster’s information, login database and the friend’s database. The database will be implemented in a way that will meet the requirements of (FR1), (FR2), (FR3), (FR6) and (FR11) of the requirement specification documents.

2.2 The CLIENT component

The client is where the user can request a webpage from the server. Our program web-pages will consist of:

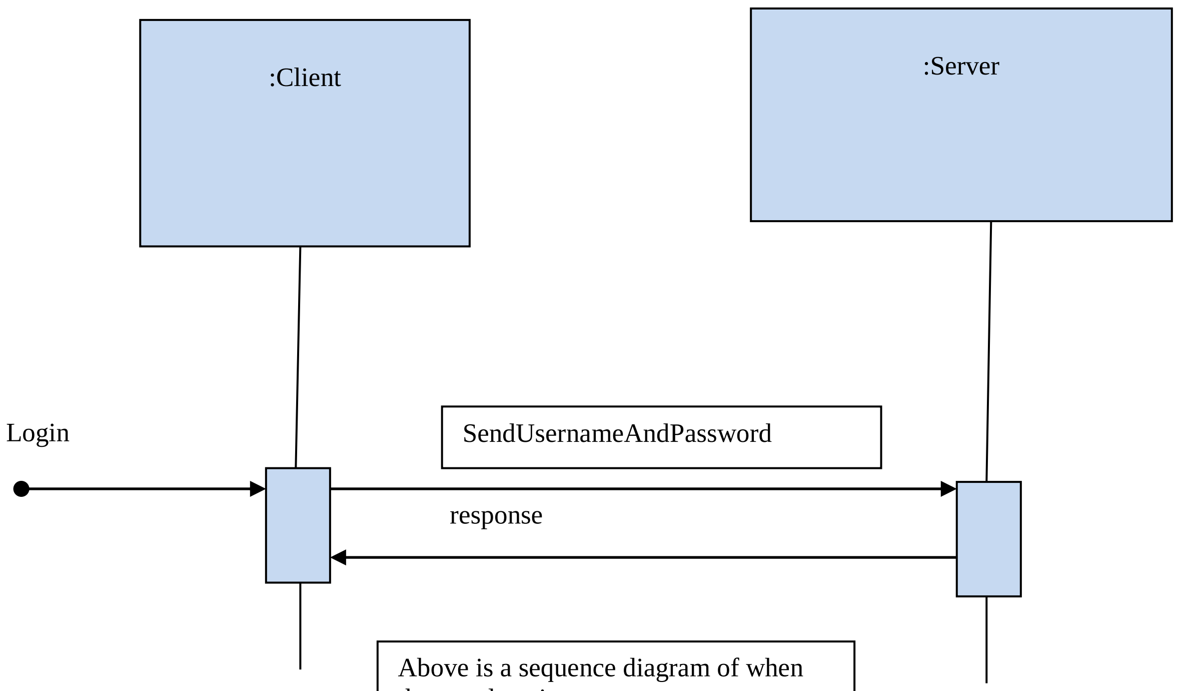
- Login page - this is where the users can login existing account or register a new account.
- Home page - the users will be able to choose various links to navigate.
- Marketplace page - this page allow users to view the monster that are to be bought or rent.
- My monster page - this page allow users to breed the monster they own.
- Monster Fights page - the user will be able to challenge other monsters through this page.
- Friends page - this page allow users to interact with other users.

The user will be able to communicate the server through POST and GET method. The POST method is used to catch the user input field which are in the forms. The GET request is used to tell a HTML file which webpage to display. The client will be implemented in a way that will meet the requirements of (FR5), (FR6), (FR7), (FR8), (FR10), (FR11), (PR1) and (PR2) of the requirement specification documents.

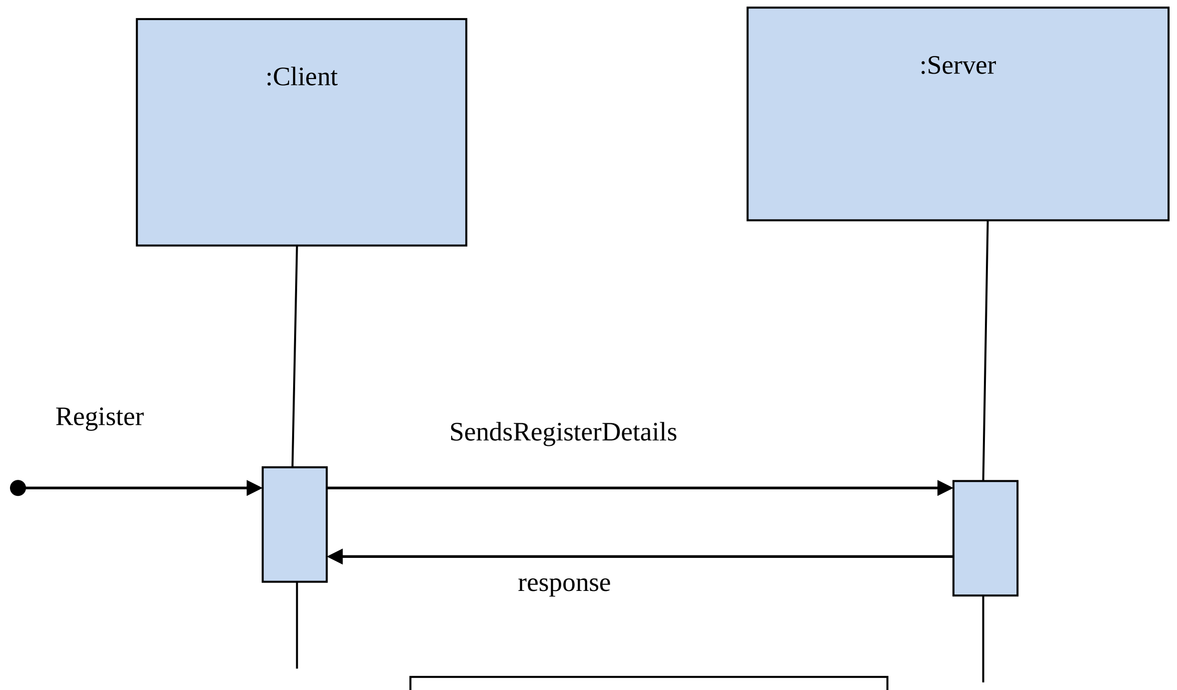
2.3 The SERVLET component

On the servlet side, we are using Glassfish servlets as a development tool which provides storage and communication storage. The servlet is used to produce a response to the request made by the user in the html page and send it back to the requesting browser through the GET and POST requests. The servlet first looks for incoming request data: if it finds none, it presents a blank form. If the servlet finds partial request data, it extracts the partial data, puts it back into the form, and marks the other fields as missing. If the servlet finds the full complement of required data, it processes the request and displays the results. The servlets will be implemented in a way that will meet the requirement of (FR1), (FR2), (FR3), (FR4), (FR5), (PR1), (PR2), (DC1) and (DC2) of the requirement specification documents.

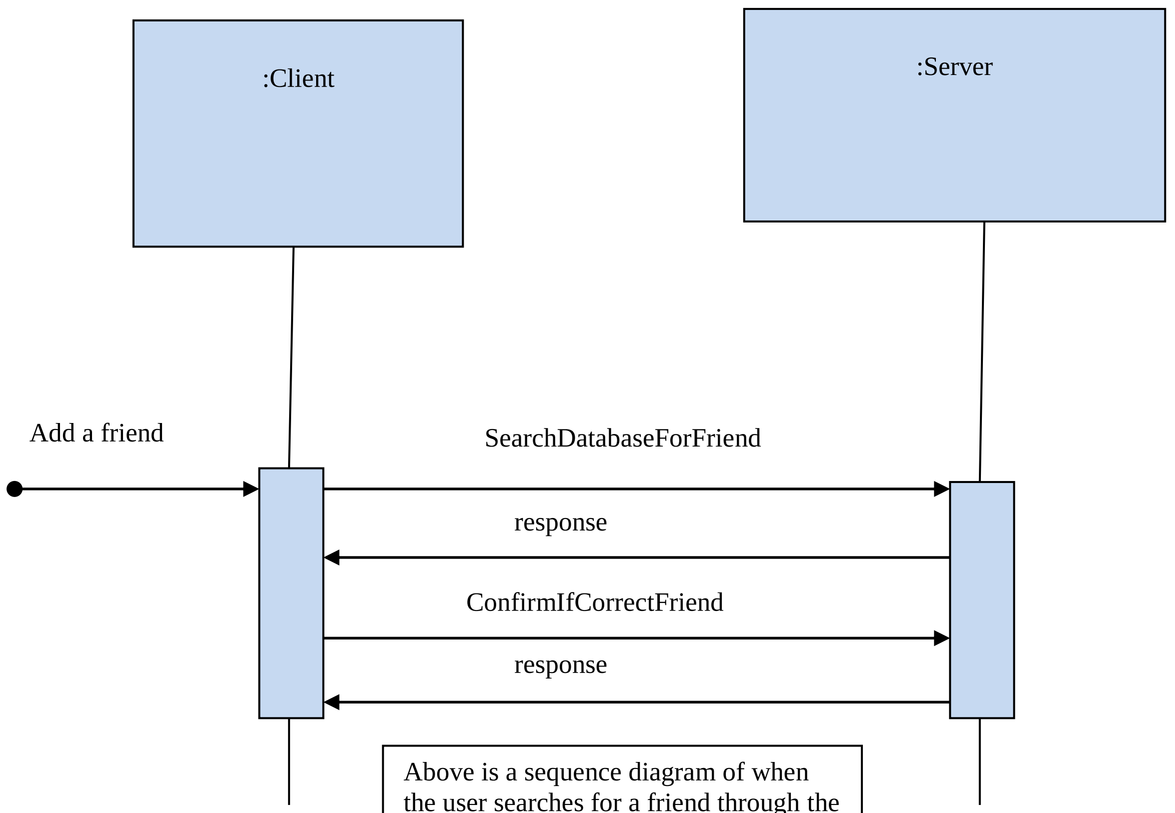
3. SEQUENCE DIAGRAMS



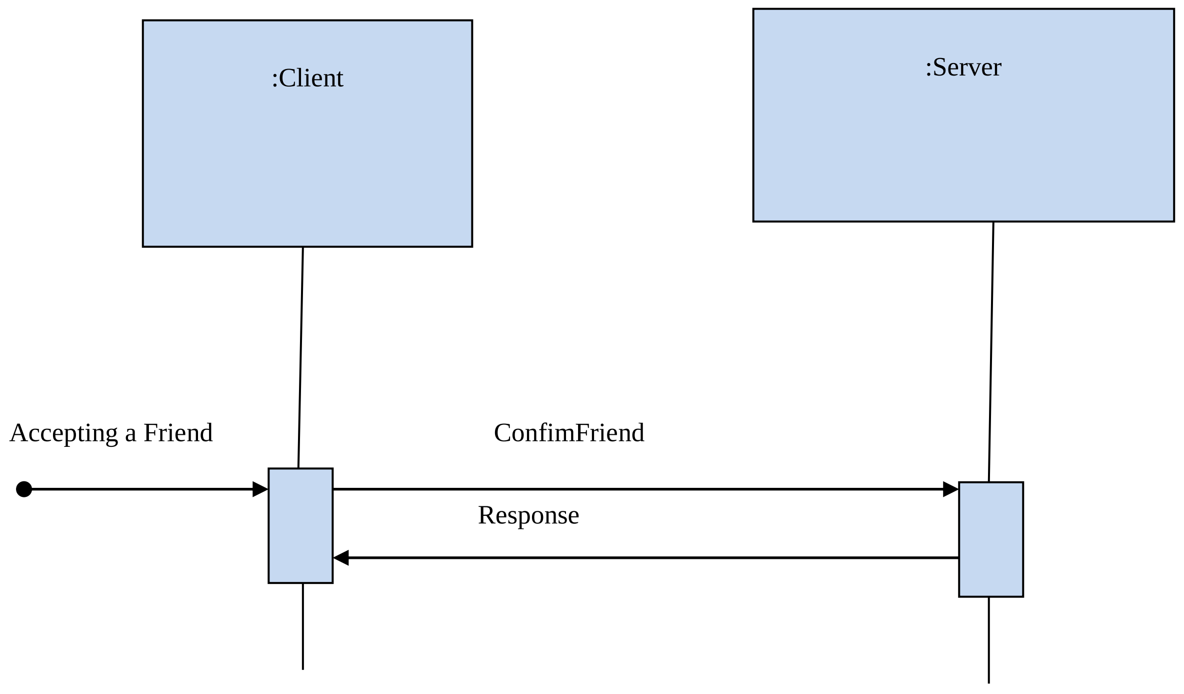
Above is a sequence diagram of when the user logs in.



Above is a sequence diagram of when the user register to play the game.

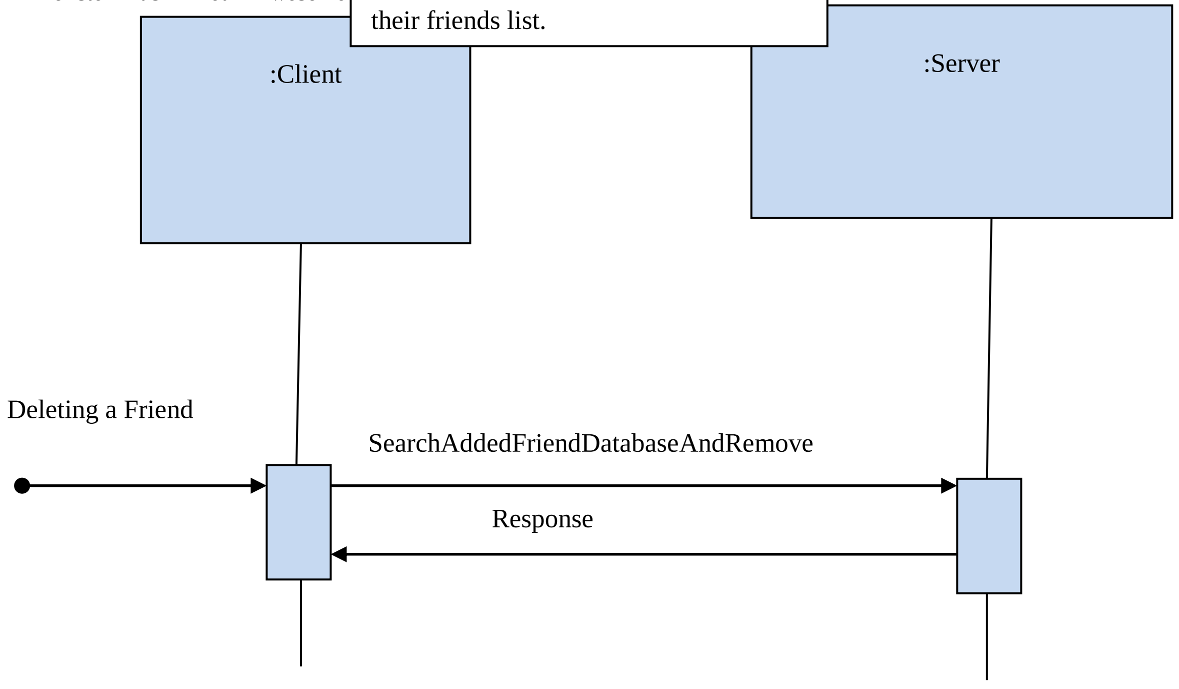


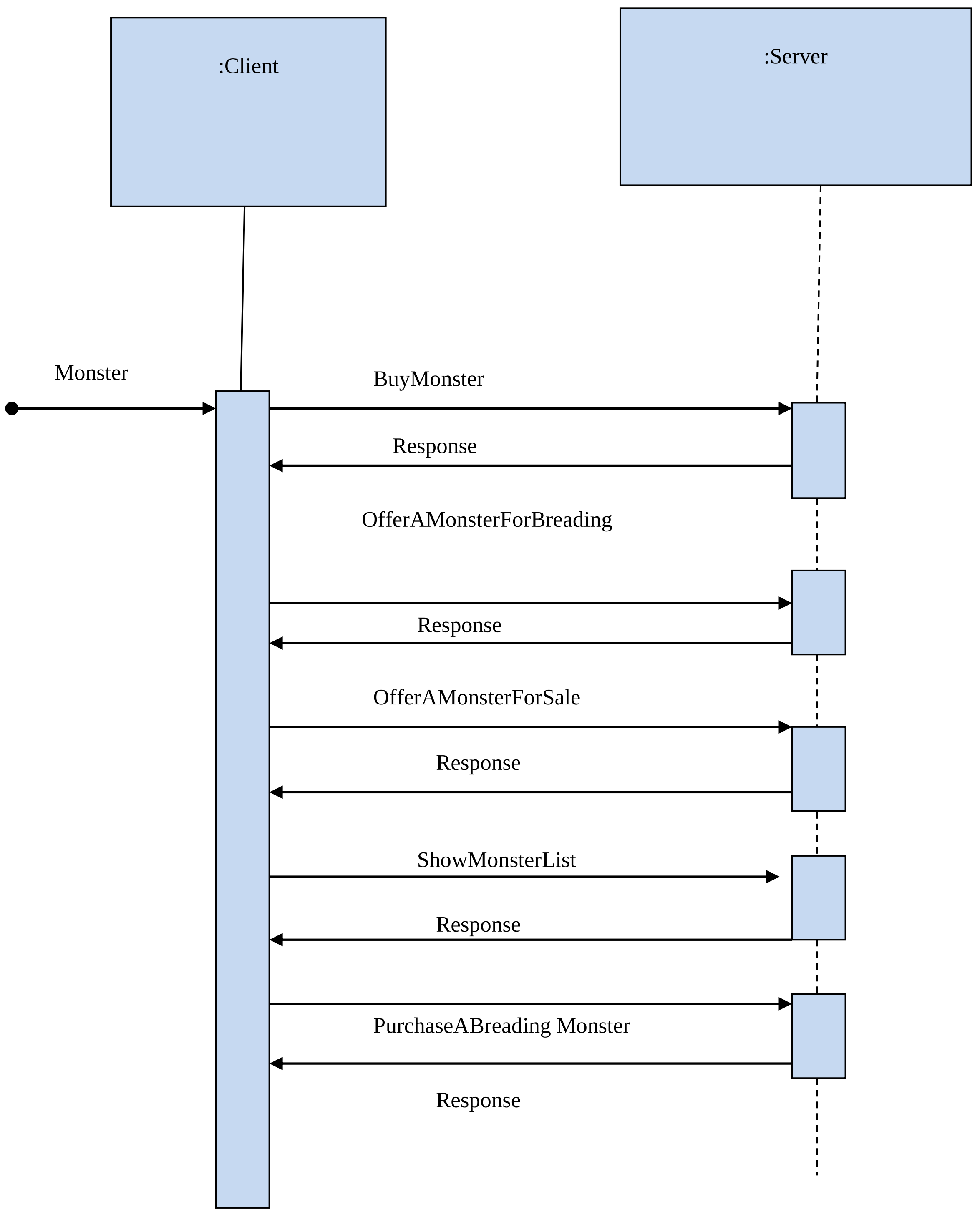
Above is a sequence diagram of when the user searches for a friend through the database and then adds that friend.



Above is a sequence diagram of when the user adding a friend.

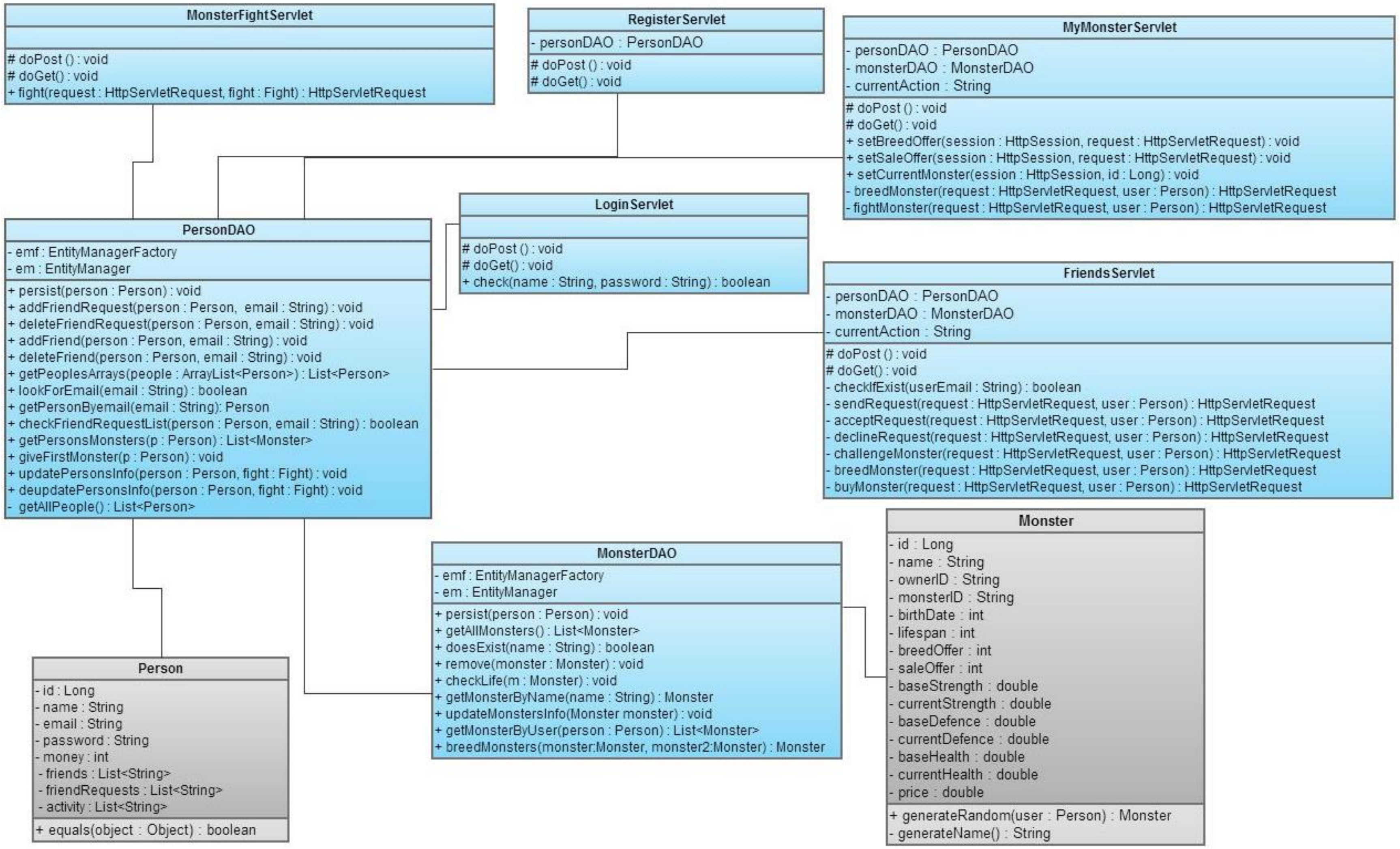
Above is a sequence diagram of when the user wants to delete a friend from their friends list.





Above is a sequence diagram of what options are available to the user to do in the game. The user has the option of buying a monster. Offering a monster for breading, or offer a monster for sale. The user can also see their monsters. They can also purchase a monster for breading.

4. CLASS DIAGRAM


MonsterFightServlet
# doPost () : void
# doGet() : void
+ fight(request : HttpServletRequest, fight : Fight) : HttpServletRequest
RegisterServlet
- personDAO : PersonDAO
# doPost () : void
# doGet() : void
MyMonsterServlet
- personDAO : PersonDAO
- monsterDAO : MonsterDAO
- currentAction : String
# doPost () : void
# doGet() : void
+ setBreedOffer(session : HttpSession, request : HttpServletRequest) : void
+ setSaleOffer(session : HttpSession, request : HttpServletRequest) : void
+ setCurrentMonster(ession : HttpSession, id : Long) : void
- breedMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
- fightMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
LoginServlet
# doPost () : void
# doGet() : void
+ check(name : String, password : String) : boolean
PersonDAO
- emf : EntityManagerFactory
- em : EntityManager
+ persist(person : Person) : void
+ addFriendRequest(person : Person, email : String) : void
+ deleteFriendRequest(person : Person, email : String) : void
+ addFriend(person : Person, email : String) : void
+ deleteFriend(person : Person, email : String) : void
+ getPeoplesArrays(people : ArrayList<Person>) : List<Person>
+ lookForEmail(email : String) : boolean
+ getPersonByemail(email : String): Person
+ checkFriendRequestList(person : Person, email : String) : boolean
+ getPersonsMonsters(p : Person) : List<Monster>
+ giveFirstMonster(p : Person) : void
+ updatePersonsInfo(person : Person, fight : Fight) : void
+ deupdatePersonsInfo(person : Person, fight : Fight) : void
- getAllPeople() : List<Person>
FriendsServlet
- personDAO : PersonDAO
- monsterDAO : MonsterDAO
- currentAction : String
# doPost () : void
# doGet() : void
- checkIfExist(userEmail : String) : boolean
- sendRequest(request : HttpServletRequest, user : Person) : HttpServletRequest
- acceptRequest(request : HttpServletRequest, user : Person) : HttpServletRequest
- declineRequest(request : HttpServletRequest, user : Person) : HttpServletRequest
- challengeMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
- breedMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
- buyMonster(request : HttpServletRequest, user : Person) : HttpServletRequest
Monster
- id : Long
- name : String
- ownerID : String
- monsterID : String
- birthDate : int
- lifespan : int
- breedOffer : int
- saleOffer : int
- baseStrength : double
- currentStrength : double
- baseDefence : double
- currentDefence : double
- baseHealth : double
- currentHealth : double
- price : double
+ generateRandom(user : Person) : Monster
- generateName() : String
MonsterDAO
- emf : EntityManagerFactory
- em : EntityManager
+ persist(person : Person) : void
+ getAllMonsters() : List<Monster>
+ doesExist(name : String) : boolean
+ remove(monster : Monster) : void
+ checkLife(m : Monster) : void
+ getMonsterByName(name : String) : Monster
+ updateMonstersInfo(Monster monster) : void
+ getMonsterByUser(person : Person) : List<Monster>
+ breedMonsters(monster:Monster, monster2:Monster) : Monster
Person
- id : Long
- name : String
- email : String
- password : String
- money : int
- friends : List<String>
- friendRequests : List<String>
- activity : List<String>
+ equals(object : Object) : boolean

5. DEPENDENCY DESCRIPTION

5.1 Introduction

Each of the web pages depends on its servlet in order to function. Each of the servlets then depends on the DAO session beans in order to access the database. These access classes then rely on the database server for them to function.

5.2 Class descriptions

5.2.1 Servlet Classes

Each servlet class serves the function of a single webpage.

5.2.2 Login Servlet

This takes the log in credentials from the user with a post and tries to find them in the database using the check method. If the credentials are correct, the user will be forwarded to a home page, otherwise the page is reloaded with an error message.

5.2.3 Register Servlet

This takes the users input with a post, checks to database to insure the email address is unique. If the email address is unique then a new person is added to the database and the user is forwarded to the log in page. Otherwise the page is reloaded with an error message.

5.2.4 Friends Servlet

The friends servlet will allow the user to display a list of their friends, add new friends and confirm friend requests. All of this will be done by accessing methods in the PersonDAO session bean.

5.2.5 My Monsters Servlet

This will allow the user to get a list of his monsters and their stats. There is a method to breed monsters which will take two monsters and produce a new monster based on their genetics.

5.2.6 Market Place Servlet

This gives access to monsters that have been flagged for sale and allows the user to flag their on monsters. Sell Monster will flag a monster for sale, Buy Monster will transfer a monster from the user selling it to the user buying it, Buy Monster To Breed will give the user access to another user's monster's genetics.

5.2.7 Fight Servlet

The methods in this class facilitate actual fights. Send Fight Request will add the users name to a list in his friend’s record flagging that he/she wants to have a fight. Accept fight will take one of these requests and initiate the fight using the fight method which takes the two monsters fighting, decides the outcome and updates the users/monsters profiles to reflect the outcome. Get monster list will return the list of a user’s monsters and in order to choose one to fight.

5.2.8 Session Beans

Session Beans allow access to the database using entity managers. Each of the servlets has an injected instance of the PersonDao Bean and through this is able to access the users and since monsters will most likely be accessed in reference to a user, the Monsters Bean will be accessed through this class.

5.2.9 PersonDAO

Has an entity manager to access the database. Persist will add a new entity to the database, Check email will take an email address and return true if this email address is found in the database, get person by email, will take an email and return the person that has that email.

5.2.10 MonsterDAO

This will provide much the same functionality as the person one only with reference to the monster table.

5.2.11 Entity Classes.

These are the form that entity is saved in the database, they contain all the attributes needed for each type and gets and sets for these attributes. They will be created in their respective session bean and added to the database files.

6. CLASS INTERFACE DESCRIPTION

6.1 Class Interface for Login Servlet

```
/* Outline for LoginServlet Class */


/*
This class handles the logging in for users.
It provides the ability to check the provided password and username
combinations against the database. Also allows performance of POST and GET
HTTP request methods.
*/
package [package name]
import [whatever]
public class LoginServlet {
      protected void doPost(){
      }
      protected void doGet(){
      }
      public boolean check(String name, String password){
      /* Takes name and password from form as parameters */
      /* Compares against database */
      /*
      Returns true if match, false if either String doesn't match
      */
      }
}
```

6.2 Class Interface for RegisterServlet

```
/* Outline for RegisterServlet Class */


/*
This class handles the registration of new users.
It provides the ability to check if the provided username exists in the
database. Also allows performance of POST and GET HTTP request methods.
*/
package [package name]
import [whatever]
public class RegisterServlet {
      protected void doPost(){
      }
      public boolean checkIfAlreadyExists(String email){
      /* Takes email provided by user as parameter */
      /* Compares against database */
      /*
      Returns true if match, false if String not found in database
      */
      }
}
```

6.3 Class Interface for FriendsServlet

```
/* Outline for FriendsServlet Class */


/*

This class handles all the functionality to do with managing the player's
friends list.

It provides the ability to send, accept and decline friend requests.
Additionally, it allows the user to pull up their list of friends or the
list of a friends monsters. Thirdly, allows the deletion of friends. Also
allows perfomance of POST and GET HTTP request methods.

*/

package [package name]

import [whatever]

public class FriendsServlet {

      protected void doPost(){

      }

      protected void doGet(){

      }

      public void sendFriendRequest(int friendID){

      /* Takes friendID (int) as a parameter */

      /* Searches database for selected friend ID */

      /* Sends request */

      }

      public void acceptFriendRequest(Friend friend){

      /* Takes specified person from list of requests */

      /* Adds to database of friends */

      }

      public void declineFriendRequest(Friend friend){

      /* Takes specified person from list of requests */

      /* Removes from list */

      }

      public List<Person> getFriendList(){

      /* Retrieves list of friends from database

      }
public List<Monster> getFriendMonsterList(Person friend){
      /* Searches database for friend specified in parameter */
      /* Pulls friend's list of monsters from database */
      /* Returns List<Monster> of friend */
      }
      public void deleteFriend(int friendID){
      /* Takes friendID parameter and searches database */
      /* Upon finding it, removes user from Friend list */
      }
      public boolean checkIfExists(int friendID){
      /* Takes parameter friendID as query for database */
      /* Searches for friend in the db */
      /* If found, return true, else return false */
      }
}
```

6.4 Class Interface for MyMonsterServlet

```
/* Outline for MyMonsterServlet Class */


/*
This class handles some of the basic functions to do with the player's
monsters.
It provides the ability to breed monsters, as well as check their status
and stats. Also allows perfomance of POST and GET HTTP request methods.
*/
package [package name]
import [whatever]
public class MyMonsterServlet {
      protected void doPost(){
      }
      protected void doGet(){
      }
      public Monster breed(Monster monster){
      }
      public Monster checkMonsterStatus(Monster monster){
      /* Takes parameter monster and accesses database */
      /* Returns monster and its stats (health, strength, etc */
      }
}
```

6.5 Class Interface for MarketPlaceServlet

```
/* Outline for MarketPlaceServlet Class */


/*
This class handles the various processes of the online marketplace for
trading monsters, etc.
It provides the ability to buy, sell and list monsters. Also allows
perfomance of POST and GET HTTP request methods.
*/
package [package name]
import [whatever]
public class MarketPlaceServlet {
      protected void doPost(){
      }
      protected void doGet(){
      }
      public void sellMonster(Monster myMonster, int money){
      /* Takes monster and amount of money (price) from params */
      /* Removes monster from player list and adds it to friend's */
      /* Decreases friend's money and increases player money */
      }
      public void buyMonster(Monster monster, int money){
      /* Takes monster and amount of money (price) from params */
      /* Adds monster to player list and removes it from friend's */
      /* Increases friend's money and decreases player money */
      }
      public void buyMonsterToBreed(Monster monster, int money){
      }
      public List<Monster> getMonsterList(){
      /* Returns list of Monsters */
      }
}
```

6.6 Class Interface for MonsterFightServlet

```
/* Outline for MonsterFightServlet Class */

/*

This class handles the functionality of the monster fights.

It provides the ability to accept, decline and send fight requests to users
on the player's friends' list. Additionally, manages the actual fights
between users' monsters. Also allows perfomance of POST and GET HTTP
request methods.

*/

package [package name]

import [whatever]

public class MonsterFightServlet {

      protected void doPost(){

      }

      protected void doGet(){

      }

      public void sendFightRequest(Person friend){

      /*

      Takes friend parameter and sends fight request to that person

      */

      }

      public void acceptFight(Person friend){

      /* accepts fight from friend parameter */

      }

      public void declineFight(Person friend){

      /* declines fight from friend parameter */

      }

      public void fight(int myMonsterID, int friendMonsterID){

      /*

      Begin fight between monsters, pulling stats from database by
      using monsterID

      */

      }

      public List<Monster> getMonsterList(Person friend){

      /* Retrieves list of monsters owned by friend */

      }

}
```

6.7 Class Interface for PersonDAO

```
package [package name]
import [whatever]
public class PersonDAO {
      private EntityManagerFactory emf;
      private EntityManager em;

      public void persist(Person person){
      }
      public boolean checkEmail(String email){
      }
      public Person getPersonByEmail(String email){
      }
      public List<Person> getAllPeople(){
      }
}
```

6.8 Class Interface for MonsterDAO

```
package [package name]
import [whatever]
public class MonsterDAO {
      private EntityManagerFactory emf;
      private EntityManager em;

      public void persist(Monster monster){
      }
      public List<Person> getAllMonsters(){
      }
      public Monster remove(Monster monster){
      }
}
```

REFERENCES

[1] Software Engineering Group Projects: Design Specification Standards. C. J. Price, N. W. Hardy. SE.QA.05A. 1.6 Relea\se

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	06/12/12	N/A - original version	PW
1.1	N/A	07/12/12	Added architectural description, sequence diagram and class diagram	PW
1.2	N/A	07/12/12	Added class interface and dependency descriptions, fixed spelling/grammar, added reference.	PW
1.3	N/A	15/02/2013	Put in updated class diagram, now final version	PW

Software Engineering Group Project
Final Report

Author:	Joshua Bird, Phil Wilkinson, Tom Hull, Dave Haenze, Chris Morgan, Kamarus Alimin, Szymon Stec, Lewis Waldron
Config Ref:	SE_02_FR_01
Date:	15-02-2013
Version:	1.1
Status:	Final

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB

CONTENTS

1. INTRODUCTION

1.1 Purpose of this Document

The purpose of this document is to give a summary of the complete project in terms of a final report.

1.2 Scope

This document should be written by the relevant person(s) (mainly by the project leader) to give aspects of the final result of the project in detail. This should be read and agreed upon by all project members where there is objectivity.

1.3 Objectives

The aim of this document is to cover the following aspects of the project in the format of a final report:
1. The End-of-Project Report.
2. The Project Test Report.
3. The Project Maintenance Manual.

It also gives details of other additional evaluations for the project, and whether the result is successful or not.

2. END OF PROJECT REPORT

2.1 Management Summary

During the production of this project, many of the required functionality has been achieved, these will be listed below and how well a state they do work in. The following contains what parts of the system are not functioning properly.

Using the use case diagram drawn in the Design Documents, the state of the requirements are as follows:

Working - Buying/Selling Monsters, Breeding Monsters (With no validation however), Challenging Monsters, Accepting Challenges, View Friend Wealth, Sending Friend Requests, Accepting/Declining Friend Requests, Registering, Logging In, Logging Out, Display Monsters and Friends and Deciding Monster Fight outcomes.

Not Included - Deleting Friends and Un-Registering.

There were a few of difficulties while producing this project although these were mitigated as much as possible. The major problem we encountered was allocation of roles in the group were organised, in the first meeting held positions were given to members of the group, but not all of the group members arrived and as such we were not aware of everyone's strengths and weaknesses. As a result certain roles were not ideal for the group, for example, Tom Hull was given ‘Dept. Leader’ and as it turns out ended up as a lead programmer. Dave Haenze was given the role of Dept. QA Manager, and was also a lead programmer.
After the first meeting it was apparent that a few of the unassigned members were not on ‘programming’ modules.
In order to mitigate the problem, the members of the group who were not on heavy programming modules were given more work to do regarding documentation side of the project, and those who were experienced with web programming were assigned to create the user interface side of the product.
All in all, the team pulled together in this aspect, and the work was spread as equally between everyone as well as possible.

2.2 Historical Account of the Project

In order to complete the project in time, a time-line of deadlines was created in the form of a Gantt Chart, which was included in the Project Plan.

There were a number of key aspects of the projects that needed to be completed and as such they were split apart, each being allocated an amount of time in which that piece of work needed to be completed in. The leader set deadlines for the group that were a bit before each of these deadlines, in order to try and reduce slippage.

The first main event was the Project Plan - this also gave a foot-holding for the rest of the project in itself and had information such as use-case diagrams, in order to try and gain an understanding of what functionality the project had to actually include. This was completed first as it was needed for the rest of the project.

Next was getting a clear understanding of the tasks needed throughout the project. This was a more detailed version of the functionality, and in order to complete this the QA documents had to be read carefully to extract the useful information that was needed to get to grip with the full requirements of the system. A set of tests were then decided for each functional requirement, so that the coders were aware ahead of time what they would need to be adding to the system.

In parallel, the Design of the project was also started, this was because it was decided that the design of the system was just as important as its functionality. The first designs included paper drawn versions that the whole team could look at to understand how the final solution should look.

Classes were designed just around this time too, because it gave the coders a better idea of how the system would be created behind the scenes.

Database design was next: - tables were drawn on paper and fields discussed by the team, to incorporate everyone's ideas about how the system should operate, and what tables were needed ahead of time with what values included in them. More time was given for this than the class diagram.

Across the whole of this, the ‘Prototype Demo Software’ was allocated. The reason for this: we were expected to present a basic version of the system by the end of the first term. To try and get a head start, it was planned to have spike work being collected (for example on servlets and databases) before trying to put together a functioning prototype.

The server functionality for the prototype was split into sections, to try and reflect this idea of spike work being collected over a period of time, with the work stretching beyond the end of the first term for more difficult work like ‘Monster Lists’. This kept the initial required sections of work like the communication between the server and databases to the front of the required work to be done.

Then lastly were plotted the deadlines for the submission of work and testing and integration week. These were there as the final milestones needed for the hand in of all of the work, and an indication of where the compiling of spike work would be done and the producing of the final version of the program and documentation.

In order for the team to work effectively on all of the work needed, this plan covered the entire span of the projects lifetime, in order to provide detailed information on how far the project was currently at. If at a particular date the project was not as far as the plan indicated, it would tell the team that the workload needed to be increased in order to maintain the quality of the final product.

2.3 Final State of the Project

In the final project, the majority of the functionality is working, except for server to server between groups however. What is working and what isn’t will be discussed below in more detail the management summary:

Firstly, we had issues with the database working near the end of the project, and as such, some functions worked, but in order for the servlets to display what was in the database, the user would have to logout, and back in to see the updates.

Buying/Selling Monsters - This functionality is working, users are limited to buying and selling monsters between their friends only as required as well as limiting the buying and selling to be within valid prices. This only works between users in this group, inter server communication does not work. The user would have to logout and log back in to see the updated monster list.

Breeding Monsters - This functionality does work, users are able to breed with their friends monsters (if they are up for breeding). However the validation that is present for buying/selling is not working for breeding, so unfortunately wrong values are able to be entered, like minus values. Needs the user to logout and login to update.

Challenging Monsters - The ability to Challenge Monsters does work, and as required only between friends. The user selects a monster they wish to fight, and then a monster of their own on the next screen to send out the challenge. Sometimes viewing the challenges sent doesn't always show up your challenges you have pending.

Accepting/Declining Challenges - The ability to accept or decline challenges does work in the system. By selecting accept challenge the fight will start and a winner will be selected. The loser's monster will be killed and removed from their monster list. If the user selects decline challenge, the fight will not take place. However the challenge is not removed from the list which it should do.
Another issue with the requests is that if the monster dies due to a fight or "natural causes" the request will not be removed. Allowing in some cases for a fight to still take place, even though the monster is already dead.

View Friend Wealth - This functionality works, however it is only displayed within the friends list itself, not on a separate page. So when the user looks at their list of friends, at the bottom of the page their friends are listed in order of how much money they have.

Sending Friend Requests - This functionality works, upon typing in a valid users email into the add friend box and clicking enter the request will be sent. The user can display all of their outgoing requests. Validation is also in place to prevent the user from adding an invalid email and also one that has already been added as a friend.

Accepting/Declining Friend requests - The user is able to accept and decline incoming invites from another player. Upon accepting, the friend list will be updated (after logging out and back in again). Upon selecting decline, the friend request is removed.

Registering - Registering works as intended, the user enters their desired username (email) and password, both which are validated to check they are valid. Their user is then created and given some monsters.

Logging In - Works, upon logging in the user is taken to their list of monsters, where they can browse to other pages or check the stats of the monsters.

Logging Out - The user is able to log out and end the session, which will allow another user to log in on the same device.

However there are some problems with the project:

Un-Registering - Unfortunately Un-Registering was not implemented.

Deleting friends - This function was not added to the project.

There were some UI problems, which meant that some of the information for monsters and such overlapped buttons and other text boxes on the screen. This was due to some errors in the CSS with tables mostly.

Risk assessment - After receiving feedback on the risk assessment we found that it is too long and the risks should have been grouped together into categories.

2.4 Performance of Each Team Member

2.4.1 Phil Wilkinson (Agreed upon):

Phil was designated the role of QA Manager, throughout the project it was his task to look through any documents/work that had been created and check to make sure that it was completed to an acceptable standard. He then had to make sure that all work was maintained to match the QA specification.

Throughout the project I would give Phil documents from other members/or tell him where they were located in our Git repository, so that he could manage them.

He also took it upon himself to restructure the documents when looking at them so that they fit better to the pages and looked nicer which I was very impressed with.
In integration and testing week Phil also managed the Git repositories and separated the different parts of work into different branches to allow finding work easier.

On the last day Phil also had to take on a role of another member and redo the user interface of the project as changes needed making after coding had been done.

Phil's performance on a whole was outstanding and I was never let down when asking him to do anything with the project.

2.4.2 Kamarus Alimin (Agreed upon):

Kamarus was designated the role of lead tester, he was to work out what tests needed to be performed and to document anything to do with testing.

At the start of the project Kamarus informed me that he was not comfortable coding anything, and requested to be given the role for testing, to which I agreed to as we had a lot of group members working on documentation.

Kamarus was also given the task of completing a risk assessment for the project, when checking through it, it was quite a bit too excessive, which wasn't too bad. It did need some editing for grammar and spelling however. After a few changes the risk assessment took shape though.

Kamarus worked with Szymon on the test table, each completing half of it. And also created the Final Test Report. Both were completed, and after grammar/spelling checks from Phil with some reformatting were usable.

Overall Kamarus completed the work set, be it a bit late at times, and I believe he completed them to the best of his ability.

2.4.3 Dave Haenze (Agreed upon):

Dave was assigned the role of deputy QA manager in the first meeting, however it became apparent very quickly that he would be more useful as a lead coder. As such he did not perform much document quality assurance or writing many documents.

For the main part of the beginning of the project Dave didn't have much to do, except discussion of class diagrams and such, but was given spike work into researching what repository system to use. Under his advice we went with Git, we had a few issues with it, but in the end Dave managed to work out how to solve these issues.

Dave worked on creating algorithms for the Monster's functions in the system, a lot of these were completed in the first term, however due to problems with Git a lot of the code was lost. Dave recovered in the integration week and created more elaborate algorithms for the monsters than before.

In the last week Dave worked closely with Thomas to finish a lot of the code and at the end java doc's. He also added various bits of validation throughout the system.

Overall Dave worked very well and completed all of the work assigned to an excellent standard, and also went beyond and helped with the final report in places.

2.4.4 Szymon Stec (Agreed upon):

Szymon was assigned the role of a programmer to start, as he was one of the few not given a role that was on a programming degree scheme. This meant that he worked along side Dave and Thomas for the most part.

I also set Szymon to complete some documentation such as the class diagram and half of the testing table. This was because he was not set as much spike work as the others.

During implementation and testing week Szymon worked heavily on the Friends side of the code, a lot of problems were encountered here due to how the database was structured, in the end however the problem was resolved.

Towards the end of the week Szymon wrote the majority of the JUnit tests in the project and helped to complete the Java Documentation as well.

All in all I believe that Szymon worked hard and completed the tasks set to the best of his ability. Any work set was given on time and complete to an acceptable standard.

2.4.5 Thomas Hull (Agreed upon):

In the first meeting Thomas was assigned the role of deputy project leader. During the project I don't believe there was a need to take on this role much however.

It became apparent very soon into the project that Thomas would be "Lead programmer" and as such wrote most of code in the system. He was assigned the task of researching servlets soon after starting the project.

Thomas recommended using GlassFish and so we decided to go with that, he read up on it a great deal and completed a lot of spike work with it which we adopted into our own project.

Thomas also researched ObjectDB databases and implemented it with the servlet work that had been created. As a group we decided to use both of these in the end system.

During implementation and testing week Thomas worked heavily on coding with Dave and Szymon, spending most of every day at the university. He worked on all aspects of the project, mostly with database interaction, and even wrote instructions for the others to use it.

Overall Thomas exceeded my expectations and worked very hard throughout the whole of the project, producing consistent work of a high standard.

2.4.6 Lewis Waldron:

Lewis was assigned with creating the user interface for the project. Using HTML and CSS, he asked for this role because he was not confident with coding.

Lewis was asked to create some designs of the pages based on the use case diagrams which were used in the documentation. This work was received and was to an acceptable standard.

Lewis was also tasked with creating basic pages in html to use for the prototype. These pages were created but in the end were not used for the prototype because Thomas had edited them to fit the code written.

Everything in the first semester went well and I received work on time by Lewis, however in the second semester there were some issues.

He wrote a good, solid initial framework for the designs with CSS in place, however as the code was being written it needed to be changed to fit this. In the end Phil Wilkinson had to redo most of it on the last day because we could not contact Lewis. As such me and Lewis do not agree on what is written here.

2.4.7 Christian Morgan (Agreed upon):

Christian was put in charge of writing most of the documentation in the project, this was agreed upon because he did not feel confident with programming. This did mean that Chris had a lot of work to complete in the first semester however.

Chris was put in charge of a lot of docs but some sections were written by others who had more knowledge of those areas (such as class diagrams). When Chris was not sure how to complete a section of the work he asked for help from those in the group that did, which was very good because it meant that documents were completed in time and I always knew how far the docs were in terms of completion.

In integration and testing week Chris was assigned to go through the documents where we had received feedback on them and to update them accordingly. As such he was not required to be present for much of the week.

Overall Chris worked to a very high standard. All of the documents I received had excellent content, only a few minor grammatical mistakes needed to be edited.

2.4.8 Joshua Bird (Agreed upon by Dave, Phil, Szymon, Chris, Thomas):

In the first meeting I was allocated the role of group leader, this was probably because I had already tried to contact the group to work out who was confident in what aspects of the work before we had even had this meeting.

Throughout the project my role was to manage the whole group and allocate work evenly between everyone, I also had to try and manage time for the whole group.
I did this in a few ways, for example I would set deadlines for work to be handed in earlier than the official deadline, so that we would have extra room to edit the documents and prevent slippage.

In the first semester I had to try and get to grips with the system as a whole as soon as possible, in order to try and work out what major tasks needed to be completed and to get an idea of how I could split the work between the group. I tried to split the work, giving those who were strongest in certain areas the work they would be most comfortable with. I believe that I managed to do this.

In integration and testing week I had to work out what was left to do and try and split it down into tasks to make it easier. I tried to think of milestones that we needed. When we were given the list of tests that we would be checked on I broke it down into bullet points that were easy to read as a “checklist”. I also have written a lot of the Final Report, along with Phil and Dave.

2.5 Evaluation of the Team and the Project

I believe in the end the whole group worked well together as a team. We had quite a lot of work to complete between us and had quite a mixture of personalities and members with a variety of different strengths. At the start of the project we probably encountered the majority of the negative sides of this, some members were not confident with strangers and as such did not speak up much in meetings.

By the end I think that this was mostly resolved however, when deadlines came near the group pulled together and finished the work on time. Everyone worked on their own sections of the project which stopped too much clashing of work.

I think that work could have been improved however, there were a few functions in the system that really could have been implemented if management had been better. For example in the last few days the programmers were writing a lot of java doc, and I now realise that this could have been performed by those good at documentation. Which would have given programmers more time to code functions.

I also think that the roles should have been changed after the first meeting, as Thomas and Dave didn’t really use their roles that much as they ended up doing a lot of the programming.

Lastly I think that more work should have been required in the first semester, if we had had more working code before integration week we would have had a better standing. This was not entirely our fault though as we did lose a lot of code due to Github behaving strangely.

I think that the most important lesson learnt was the importance of time management, although I feel I did a good job, I did feel that at times when I set a deadline it wasn’t taken too seriously and I received work later than I had planned, perhaps I should have been more strict in this regard.

I think that we also learned the value of spike work, without the time invested by Dave and Thomas into servlets/databases/version control we would not have finished the project to the standard that we did.

3. APPENDICES

3.1 Project Test Report

Test Ref	**Requirement being test**	**Pass/Fail Criteria**	**Reason for failure if any;**
SE-N02-001	FR1 + FR6	PASS	
SE-N02-002	FR1 + FR6	PASS	
SE-N02-003	FR1 + FR6	PASS	
SE-N02-004	FR1 + FR6	PASS	
SE-N02-005	FR1 + FR6	PASS	
SE-N02-006	FR1 + FR6	FAIL	Unregister function is unavailable
SE-N02-007	FR6	FAIL	Unregister function is unavailable
SE-N02-008	FR2 + FR6	PASS	
SE-N02-009	FR2 + FR6	PASS	
SE-N02-010	FR2 + FR6	PASS	
SE-N02-011	FR2 + FR6	PASS	
SE-N02-012	FR2 + FR6	PASS	
SE-N02-013	FR2 + FR6	PASS	
SE-N02-014	FR2 + FR6	FAIL	Function to remove friend is unavailable
SE-N02-015	FR2 + FR6	FAIL	"Delete" function is unavailable
SE-N02-016	FR3 + FR6	PASS	
SE-N02-017	FR3	PASS	
SE-N02-018	FR4	PASS	
SE-N02-019	FR4	PASS	

SE-N02-020	FR4	PASS	
SE-N02-021	FR4	FAIL	The challenge function works but the no media were loaded
SE-N02-022	FR6	PASS	
SE-N02-023	FR6	PASS	
SE-N02-024	FR6	PASS	
SE-N02-025	FR6	PASS	
SE-N02-026	FR6	FAIL	Function of buying monster in the market is unavailable
SE-N02-027	FR6	PASS	
SE-N02-028	FR6	PASS	
SE-N02-029	FR6	PASS	
SE-N02-030	FR6	PASS	
SE-N02-031	FR6	FAIL	No error message was shown
SE-N02-032	FR6	FAIL	The Buying function in the monster page is unavailable
SE-N02-033	FR6	PASS	
SE-N02-034	FR1 + FR7	PASS	
SE-N02-035	FR7	PASS	
SE-N02-036	FR7	PASS	
SE-N02-037	FR7	PASS	
SE-N02-038	FR7	PASS	
SE-N02-039	FR7	PASS	
SE-N02-040	FR7	PASS	
SE-N02-041	FR8	PASS	
SE-N02-042	FR8	PASS	
SE-N02-043	FR8	PASS	

4. PROJECT MAINTENANCE MANUAL

4.1 Program description

The program created was a web-based game involving monsters and players. The game involves text-based fights and buying/breeding between users. Users can only interact with friends that they have added via sending requests to each other. The user's aim is to make as much money as they can from their monsters and to be as high on their friend's "rich list" as possible.

4.2 Program structure

Our design uses several different diagrams, primarily a class diagram and sequence diagram. Those can be found in design specification. There is also a Use Case diagram available.
Not counting the JSP files (of which there is one for every servlet), the program consists of several source files. First of all, there are the Java files for the two main objects used by the program, Person and Monster, as well as their respective DAO files (PersonDAO.java and MonsterDAO.java, respectively). There is a .java file for Fight objects, which contain information about the challenger and their monster, among others. Additionally, there are the servlet files, one for each JSP file. These are: FriendsServlet(managing the friends list), MyMonsterServlet(for managing your monsters), MonsterFightServlet(for managing fights and fight requests), LoginServlet(for logging in to the game) and finally Register(creating new users).

4.2.1 PersonDAO.java

public void persist(Person person)
Creates a connection with database server.

public void addFriendRequest(Person person, String email)
Opens connection with database server. Finds person entity in database by unique ID. Adds new friend request to ArrayList of friend requests. Updates database and closes the connection.

public void deleteFriendRequest(Person person, String email)
Opens connection with database server. Finds person entity in database by unique ID. Deletes friend request from ArrayList of friend requests. Updates database and closes the connection.

public void addFriend(Person person, String email)
Opens connection with database server. Finds person entity in database by unique ID. Adds new friend to ArrayList of friends. Updates database and closes the connection.

public void deleteFriend(Person person, String email)
Opens connection with database server. Finds person entity in database by unique ID. Deletes friend from ArrayList of friends. Updates database and closes the connection.

private ArrayList<Person> getAllPeople()
Opens connection with database server. Gets all database entities from the Person table. Adds all entities to the ArrayList. Returns ArrayList.

public ArrayList<Person> getPeoplesArrays(ArrayList<Person> people)
Creates an ArrayList of Persons then returns that ArrayList.

public Person getPersonsArrays(Person p)
Opens connection with database server and gets the Person object by its unique ID from the database. Finally, it returns the Person object.

public boolean lookForEmail(String email)
Uses getAllPeople() method and checks if provided unique email address exists in database. Returns "true" if found, otherwise returns "false".

public Person getPersonByEmail(String email)

Uses getAllPeople() method and searches for provided email address. Returns the Person if found, otherwise returns "null".

public boolean checkFriendRequestList(Person p, String email)
Opens connection with database server and tries to find the Person entity in the database by its unique ID. Checks if the friends email exists in the person's ArrayList of friend requests. Returns "true" if found, otherwise returns "false", then closes database the connection.

public ArrayList<Person> getPersonsFriends(Person p)
Returns an ArrayList of the person's friends.

public ArrayList<Person> getPersonsFriendRequests(Person p)
Returns an ArrayList of the person's friend requests.

public ArrayList<Monster> getPersonsMonsters(Person p)
Uses getMonsterByUser() method form MonsterDAO class then returns an ArrayList of the person's monsters.

public void giveFirstMonster(Person p)
Uses generateRandom() method from Monster class and adds the new monster to the new user's monster list.

public Person getPersonByID(Long id)
Uses getAllPeople() method. Finds Person by unique ID.Returns Person if found, otherwise returns "null".

public void updatePersonsInfo(Person person)
Opens connection with database server and finds the person entity in the database by its unique ID. Updates the person's money then updates the database and closes the connection.

public void updatePersonsInfo(Person person,Fight fight)
Opens connection with database server and finds the person entity in the database by its unique ID. Adds new fight offer then updates the database and closes the connection.

public void deupdatPersonsInfo(Person person,Fight fight)
Opens connection with database server and finds the person entity in the database by its unique ID. Removes fight offer then updates the database and closes the connection.

4.2.2 Person.java

public ArrayList<Fight> getFightChallenges()
Returns an ArrayList of the person's fight challenges.

public ArrayList<Fight> getFightOffers()
Returns an ArrayList of the person's fight offers.

public ArrayList<Fight> getAllFights()
Returns an ArrayList of the person's fights.

public void addFight(Fight fight)
Adds a new fight to the person's ArrayList of fights.

public void removeFightOffer(Fight fight)
Removes a fight offer from the person's ArrayList of fights.

public void removeFightOffer(Person opponent, Monster oppMonster)
Removes a fight offer from the person's ArrayList of fights if fight offer exists.

public Fight getFightByID(Long person, Long monster)
Returns the person's fight if found, otherwise returns "null".

public ArrayList<String> getActivity()
Returns person's ArrayList of activities.

public void addActivity(String active)
Adds new activity to person's ArrayList of activities.

public ArrayList<String> getAllFriendRequests()
Returns person's ArrayList of friend requests.

public void addFriendRequest(String email)
Adds new friend request to person's ArrayList of friend requests.

public void removeFriendRequest(String email)
Removes friend request from person's ArrayList of friend requests.

public ArrayList<String> getAllFriends()
Returns person's ArrayList of friends.

public void addFriend(String email)
Adds new friend to person's ArrayList of friends.

public void removeFriend(String email)
Removes friend from person's ArrayList of friends.

public void setEmail(String email)
Sets the person's email address.
public void setMoney(int money)
Sets the person's money.

public void setName(String name)
Sets person’s name.

public void setPassword(String password)
Sets the person's user password.

public String getEmail()
Returns persons email address.

public int getMoney()
Returns person's money.

public String getName()
Returns person’s name.

public String getPassword()
Returns the person's password.

public Long getId()
Returns the person's unique ID.

public void setId(Long id)
Sets the person's ID.

4.2.3 Register servlet

All of the methods throws ServletException and IOException, except getServletInfo().

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
Does nothing. Not fully implemented.

protected void doGet(HttpServletRequest request, HttpServletResponse response)
Call the processRequest method.

protected void doPost(HttpServletRequest request, HttpServletResponse response)
Creates a new PersonDAO object for interacting with the database. Assigns request parameters to newly created Strings for checking if an email address is already in the system. If so, displays warning. Otherwise, adds email and password combination to database, then forwards to login page.

public String getServletInfo()
It has no parameters, and simply returns a String "Short description". Has not been fully implemented.

4.2.4 Login Servlet

All of the methods throws ServletException and IOException, except getServletInfo() and check.

protected void doPost(HttpServletRequest request, HttpServletResponse response)
Sets newly created Strings to request parameters email and password. Checks if user has clicked logout button. If not, creates new session and populates attributes with fields from the database (user's monsters, fight requests, etc.) and forwards to myMonsters.jsp. Otherwise, posts error due to wrong password and/or email being used.

protected void doGet(HttpServletRequest request, HttpServletResponse response)
Does nothing. Not used by the program.

public boolean check(String email, String password)
Creates new PersonDAO for interacting with the database. Returns a 'true' response if email and password combination is correct.

public String getServletInfo()
It has no parameters, and simply returns a String "Short description". Has not been fully implemented.

4.2.5 MyMonster Servlet

The doGet, doPost and processRequest methods throw ServletException and IOException.

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
Does nothing. Not fully implemented.

protected void doGet(HttpServletRequest request, HttpServletResponse response)
Call the processRequest method.

protected void doPost(HttpServletRequest request, HttpServletResponse response)
Two objects are re-initialized, personDAO and MonsterDAO. The user is retrieved using the respective session attribute. Next, through a series of if statements, it is decided which method to call. Finally, the page is reloaded.

public String getServletInfo()
It has no parameters, and simply returns a String "Short description". Has not been fully implemented.

public void setBreedOffer(HttpSession session, HttpServletRequest request)
The monsterDAO object is re-initialized and used to find a monster by its ID (provided as a request parameter). A new integer is created and set to the value of the request parameter "breed price". The Monster object's sale offer is set to this integer, then the database is updated. Finally, the currentAction is set to "changeMonster".

public void setSaleOffer(HttpSession session, HttpServletRequest request)
The monsterDAO object is re-initialized and used to find a monster by its ID (provided as a request parameter). A new integer is created and set to the value of the request parameter "sale price". The Monster object's sale offer is set to this integer, then the database is updated. Finally, the currentAction is set to "changeMonster".

public void setCurrentMonster(HttpSession session, Long id)
A new MonsterDAO object is created and the session attribute "current monster" is set to the Monster that possesses the ID from the method's parameters.

private HttpServletRequest breedMonster(HttpServletRequest request, Person user)
Firstly, the monster IDs are retrieved from the session/request and are used to select the appropriate monsters from the database; these are called "stud" and "bitch". Next the seller is found in the database (the provider of the "stud" monster) and their money increases while the user's decreases. A new Monster object is created, using the breedMonsters method. Its owner is set to the user and then all changes are saved in the database and the request returned.

private HttpServletRequest fightMonster(HttpServletRequest request, Person user)
Firstly, the monster IDs are retrieved from the session/request and are used to select the appropriate monsters from the database. After that, the challenged person is found in the database and a new Fight object is created, using the challenger, the challenged and the two monsters as parameters. Finally, the fight is added to both Person objects and their respective database fields updated and the request is returned.

4.2.6 Friends Servlet

The doGet, doPost and processRequest methods throw ServletException and IOException.

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
Does nothing. Not fully implemented.

protected void doGet(HttpServletRequest request, HttpServletResponse response)
Does nothing. Not fully implemented.

protected void doPost(HttpServletRequest request, HttpServletResponse response):
Method uses user response to execute functions related with persons friends such as:
- sending friend request.
- accepting received friend requests.
- declining received friend requests.
- displaying friends monsters.
- requesting monster fights.
- buying monsters.
- breeding monsters.
- printing friends and friend requests lists.

private boolean checkIfExist(String userEmail):
Method checks if given email exists in the database and returns "true" if found, otherwise "null" is returned.

private HttpServletRequest sendRequest(HttpServletRequest request, Person user):
Method checks if friends email exists in database and if found adds new friend requests to friends ArrayList of friend requests and database is updated, otherwise error message is displayed.

private HttpServletRequest acceptRequest(HttpServletRequest request, Person user):
Method finds Person object form database by persons unique email address, if not found displays en error message, otherwise checks if friend requests exists in person friend requests list, if found a new friend is added to user and request sender friend list. Friend request from users friend requests list is deleted and database is updated.

private HttpServletRequest declineRequest(HttpServletRequest request, Person user):
Method finds Person object form database by persons unique email address, if not found displays en error message, otherwise checks if friend requests exists in person friend requests list and deletes selected friend requests. Database is updated.

private HttpServletRequest challengeMonster(HttpServletRequest request, Person user):
Method sends fight request to users selected friend and updated the database.

private HttpServletRequest breedMonster(HttpServletRequest request, Person user):
Method gets users and friends selected monsters to generate new monster and updates sellers and users money, the database is updated. If not enough money to buy breed the appropriate message is displayed.

private HttpServletRequest buyMonster(HttpServletRequest request, Person user):
Method checks if user have enough money to buy a monster, if not the error message is displayed, otherwise sellers and users money is updated and sellers monster changes its owner ID to current user ID,
Monster, seller and user information's in database are updated.

public String getServletInfo()
It has no parameters, and simply returns a String "Short description". Has not been fully implemented.

4.2.7 MonsterFightServlet

protected void processRequest(HttpServletRequest request, HttpServletResponse response)
Does nothing. Not fully implemented.

public HttpServletRequest fight(Fight fight, HttpServletRequest request)
personDAO is re-initialized and used to create two new Person objects, challenger and user. Then, the monster's are retrieved from the Fight object in the parameters. Please refer to the "Algorithms" section below for information on the fight algorithm that is in this method. After the fight has taken place, the monster information is updated and a check is made to see which monster died. If the opponent's monster died, it is removed from the database, the request attribute "fight result" is set to "You won" and user money is increased by 100. Otherwise, the user monster is removed, the attribute is set to "You lost" and user money is decreased by 100. Finally, the user and challenger have the fight offer removed and their info is updated; the request is returned.

protected void doGet(HttpServletRequest request, HttpServletResponse response)
Calls the process request method.

protected void doPost(HttpServletRequest request, HttpServletResponse response)
Used to post-process requests. Allows user to set session attributes such as current person, current monster, set up fight requests and delete said requiests.

public String getServletInfo()
It has no parameters, and simply returns a String "Short description". Has not been fully implemented.

4.3 Algorithms

4.3.1 Fight algorithm:

The algorithm for fighting between monsters is loosely based on the method for conducting fights in the game *Dungeons & Dragons*. Each round of the fight involves a dice roll by each monster – in our program it is the generation of a random double float between 0.0 and 1.0. If the dice roll is equal to 1, or if it were a 20-sided die, a "natural 20", a critical hit is scored and half of the attacker's strength is removed from the defender's health. Otherwise, the attacker's strength is added to the randomly generated float and the sum is compared to the defending monster's defence. If the sum is higher, a hit is scored and the defender's health is reduced by a quarter of the attacker's strength. Otherwise, it is a miss.
This is conducted more or less simultaneously for both monsters. At the end of the loop, a check is made to see if one of the monsters' health is below zero, if so it is removed from the database and an appropriate message is broadcast.

4.3.2 Breeding algorithm:

Firstly, an 'empty' Monster object is created, this is the resulting 'baby' Monster. Its stats are based on the stats from its parents. A dice roll (results being 0, 1 or 2) determines which stat is inherited from which parent (strength, defence or health) and which one is averaged from the two parents. For example, if the result of the dice roll is 0, then the baby inherits its strength from the first parent, its defence from the second parent and the

health is an average of the two parent’s health. Finally, there's a 5% chance of random mutation. So, a random number between 1 and 20 is generated, and if the result is 1, then another number is generated that decides which stat is augmented by 25%.
Lastly, the baby monster is given a set lifespan, its name is created and birthdate set.

4.3.3 New monster generation:

Monster generation is fairly easy, it revolves around number generation using the Random class. Each stat is generated in turn. If it is below 0.5, 0.5 is added to it (giving an effective range of 0.51-1.0). A name is randomly chosen from a set list. Finally, it is given an ownerID, or the user's email address.

4.3.4 Lifespan calculation:

This is one of the simplest algorithms. It simply subtracts the difference of the current time and the monster's birthdate from the monster's lifespan. If this results in the lifespan being less than 0, the monster is removed from the database.

4.4 Main data areas

There are only two objects used in this program, those are Monster and Person objects. They contain all the information needed for the functioning of the program (e.g., monster stats, monster ownership, etc). Objects of type Monster are stored in a database file, monster.db, whereas Person class objects are stored in person.db.
The Person class also contains four fields that are ArrayLists – three of these are String type, and the fourth is a custom type, 'Fight'. These four lists contain the friends, friend requests (sent and received), fights (challenges sent and received) as well as a list of all the most recent activity of that user (fights fought, monsters bought, etc) (not implemented).

4.5 Files

The only files used by our program are two objectdb files. The first one is monster.db, used for storing monster objects and their attributes – owner, various stats such as strength and their name. The second file is person.db, used for storing the person objects and any other info pertaining to people – list of friends, list of monsters, friend and fight requests, etc.
If these files are not found, new ones are created.

4.6 Interfaces

All of the servlets handle various session variables as well as POST and GET requests. However, none of the servlets implement GET, it's all done POST.

4.6.1 MyMonsterServlet

The `doPost()` method in the MyMonsterServlet handles various actions, such as setting the breed or sale offer for your monsters, selecting the current monster, as well as choosing which monster you want to use when responding to a fight request or breeding.

4.6.2 FriendsServlet

The `doPost()` method in the FriendsServlet handles various actions, such as sending friend requests, accepting or declining received requests, and challenging/buying/breeding monsters.

4.6.3 Register servlet

The `doPost()` method in the Register servlet checks that a person's email is not already in use, then adds them to the database.

4.6.4 LoginServlet

The `doPost()` method in the LoginServlet processes the user logins. It checks the password and email are correct before forwarding them to the MyMonsterServlet.

4.6.5 MonsterFightServlet

The `doPost()` method in the MonsterFightServlet processes various request actions, such as accepting a fight, refusing one, displaying challenger stats, etc.

4.7 Suggestions for improvements

Even though a complete working version of the project was created, there are always room for improvements. If there was extra time to work on it more features would be added, suggestions are below:

Deleting Friends - This is a feature a lot of "games" or projects similar to this (anything that uses "friends" tends to have. Sometimes the user does not want to keep in touch with other users anymore or there are falling outs. Due to time constraints we couldn't include this. Adding this feature would just take more time, no change to the structure needed.

Un-Registering - This was planned to be included, in case a user just wanted to remove themselves from the game completely. No need to change structure either.

More detailed combat - The fight system as it is very boring, it only shows who won the fight. Ideally we would have liked to include a step by step rundown of the fight as the algorithm was run. This would require some extra communication between the servlets and the code, this would probably not require that much change to the system.

Pictures for monsters - An aesthetic change to make the users more engaged with the game. Suggestions for this would be an algorithm that would set a random picture of a monster when it was born/created. We already had the pictures but not the time to implement it. This might take a bit more to code because it would mean adding more fields to the servlets to interact with. And we didn't attempt to use images anywhere else, so it could potentially need some more libraries to work.

4.8 Things to watch for when making changes

Generally speaking, there is not much to look out for when making changes except for the obvious. For example, files are called upon, such as `forwardURL="friends.jsp"` so renaming files will require updating their various references. Also, certain objects and/or variables are passed down between different methods or even whole classes, so care should be taken when renaming them and/or changing their type, so as to avoid an incompatible type being passed down (e.g., an `int` is changed to a `double` in one of the first methods, when it is passed down to the next method that still requires an `int` errors will occur).

4.9 Physical limitations of the program

To the best of our knowledge, the program is not very resource intensive, and so may run on a variety of systems. It has successfully run on machines in various computer labs around the Aberystwyth University campus, as well as personal machines of several group members.

4.10 Rebuilding and Testing

All of the project files for the code and configuration is automatically controlled by the IDE Netbeans, and is located in the 'Coding' or in the 'trunk' branch of the github repository called 'team-awesomer-monster-mash'.

The entire project can be loaded from the project folder and Netbeans will automatically recognise the project setup. The tests that can be run for the project can be found in the produced test specification, and the tests on the specification are passed if the actual result is equal to the expected result.

REFERENCES

[1] Software Engineering Group Projects: Producing a final report. C. J. Price, N. W. Hardy and B.P. Tiddeman .SE.QA.11. 1.6 Release

DOCUMENT HISTORY

Version	*CCF No.*	*Date*	*Changes made to document*	*Changed by*
1.0	N/A	15/02/2013	N/A - original version	PW
1.1	N/A	15/02/2013	Updated formatting and fixed few errors	PW

Thomas Hull Personal Reflection For Group No.2

During the first meeting of the group I was elected the role of deputy project leader. I also expressed my interest in being more involved in coding rather than documentation or quality control. As such I was tasked with researching Java servlets.

While researching servlets I recommended using GlassFish and after discussion with the group we adopted this. Once the platform had been chosen I started some spike work, completing tutorials and experimenting with GlassFish's capabilities. While doing this I ended up using ObjectDB database system and we ended up using it the final implementation of the project.

With my familiarity with the platforms we were using I was able to help with designing at meetings, I helped map out the class diagram and what database entries we would need. I also provided advice on how to use the database and GlassFish. To this end I also created a short instruction set for the use of the database.

When we were ready to start implementation I was involved in a lot of coding. I created the structure for our user entity and created the registration and log in facilities for the prototype demonstration, including all the data validation checks on user input.

After the prototype was completed I created the facility for displaying a users monsters. It was at this point I hit a bug where I was receiving null pointers when retrieving ArrayLists from the database. At that stage I could not find the source of the bug and simply made work around and forgot about it. Unfortunately at a later stage Szymon hit the same problem while implementing friends and after a long time we found that the problem was ObjectDB itself. We tried updating to the latest version however the problem persisted. At this stage we were mid coding week and unable to retrieve any ArrayLists from the database using queries. For this I created a work around using a find function however this was not as effective as using queries. With Szymons help I then finished the friends requirements.

I was also responsible for taking David's fighting algorithm and implementing it in the program and worked with David to implement breeding and selling. As part of the coding team I was present for all

Personal Report – David Haenze

Role in group

At our first group meeting I was assigned the role of deputy QA manager, with Phil being the QA leader. However, it became apparent over the successive group meetings that I would be more valuable to the group as one of the lead programmers, hence I spent very little time assuring the quality of our project, with Phil having done most of that work. I still kept the title, however.

After we realized the skill sets of the various group members, I in essence was the deputy head programmer, under Thomas Hull. Together we figured out much of the program. I wrote the algorithms for breeding and fighting, as well as improved the initial design of our JSP files. Additionally, I worked on some of the client-side validation, for selling prices and the like. Finally, I spent a lot of time on just general code tidying up and Javadoc writing.

I was also assigned the role of "GitHub guru", and I was tasked with figuring out how git works and how to interface it with our IDE of choice, Netbeans. Unfortunately, I was less than succesful in this venture, and Phil had to step up and take over for me, especially once I became more preoccupied with the coding side of our project. However, I still continued to research the functioning of Git, especially as we encountered various problems whenever we tried to sync commits.

Troubles encountered and resolution

Initially, there was a lot of trouble with figuring out how the code worked – Thomas had written a lot of it all on his own, so the general outline, i.e., most of the database files (Monster.java, MonsterDOA.java, etc), the servlets and the JSPs had (at least) rudimentary code, so getting my head around it took a little time, but with some research, both online and with Thom, I managed to understand our code and work with it.

Our version control system, GitHub caused us a lot of trouble, with deletion of code happening more than once. Part of this was due to failures on my part, as I did not wholly understand the functioning of git, and part due to improper training in the use of git, leading to us coders overwriting the wrong files at times and the like. However, after many hours spent perusing the various online manuals for Git, I came to understand it and these troubles disappeared.

Overall success

Overall, I feel our group was successful, at least in terms of providing a functional program that fit the specification and met most of the testing requirements. Personally I would never play such a game, and there were many other things that could have been done better, especially our interface – due to time constraints, with our interface having to be effectively re-written the morning of the submission. I myself had more of a vision for it, including images for our monsters, play-by-play description of a fight, such as "You attacked, but missed! Enemy monster attacked and did 10 damage!" etc, but in the end we were only able to implement a simple "You won! Here are you monster stats".

To sum up, we did a good overall job fulfilling requirements but could have delivered a much better and more polished product with better time management.

Phil Wilkinson Reflective report

Group project task: For this project, I was tasked with the job of QA manager, which essentially meant that I had to make sure that any work produced was up to an acceptable/great standard, especially with the documentation produced. Not only this, I had to maintain the documentation in accordance with the QA specification - this meant having an awareness of the format and content of the documents, and writing the introduction section for each of the document deliverables. I was also tasked with maintaining the structure of the repository (Git) which was synced to Github.com (as a remote repository) so that version control was correctly implemented. Furthermore, I put myself forward for re-designing the project user-interface (layout of web pages), when the person tasked with this: firstly (and in my opinion at least), did not produce a layout of reasonable quality for the time given, and secondly, was not available on a crucial night/morning to implement new additions of code into the layout.

Problems encountered: One of the biggest problems encountered was github malfunctioning multiple times including an entire branch disappearing for no apparent reason when a team member synced their commits the remote repository. This had to be recovered from the local backup made on another team member's computer, which still meant that a little bit of work had been permanently lost, and had to be produced again. After a while, there seemed to be less problems occurring in that aspect, and more common conflict problems which needed to be resolved when people committed different versions of the same file they had been working on. Other team members had difficulty migrating to using github as version control and hardly used it for this.

What went well: Creating the introductions to the document deliverables were fairly straight forward for me and setting up github was not difficult due to the "Github for Windows" application (more advanced options/features still had to be accessed via the command-line however). Maintaining the structure of the documents were also fairly easy to do.

If I were given this project again, what things would I do differently: If I were to be given the project to do again, I would volunteer myself to also take on the user-interface aspect as I believe that I am a good web coder and could most likely could do a much better job given a week, than the few hours I had to do it in (when taking over).

Personal reflection report of group project - Lewis

For this project my role was to design the interface of the game via web technologies in the form of HTML and CSS. The reason I volunteered for such a role is that I take great interest in web development whilst it also being the general scope of my degree scheme (internet computing).

Whilst designing the interface I ensured that I frequently checked with the team that they were happy with the image and ethos of the web site to ensure my work was not disliked. After designing the interface my role was to continuously adapt the site to programmers' needs whilst also integrating the use of Javascript and converting the files into Java servlet pages (JSP).

By eventually learning to use a version control system (in this case GitHub) I believe I was able to contribute in an efficient and organised manor. Whereas at the start of the project myself and other team members tended to continuously exchange updated versions of files.

The main reason why I avoided a role in the general object oriented programming side of the project is due to my very average grades I had achieved last year when writing code in Java. I also do not enjoy this process at all, therefore I would have been unable to be enthusiastic about something I do not enjoy which would have degraded the overall group performance.

Most of the time the group were able to communicate and work effectively together. Although there was one time regarding my absence due to my part time job commitments. At first I believe the issue was dealt with unprofessionally but eventually all was resolved in an efficient manor. Asides from that minor instance it is to my knowledge that there were no clashes of any sort between group members. It is also good to see that it seems that everyone was assigned tasks in which they were most knowledgeable in. By splitting the project into assigned roles and responsibilities the project was very much completed in a timely and professional way.

Overall I am very happy with our performance as a group and I would very much like to work with my fellow group members again.

Personal reflective report

Szymon Stec, szs1@aber.ac.uk, Aberystwyth 2012/2013

The Monster Mash group project was the most crucial assignment during the first semester of my second year of studying at the Aberystwyth University.

Not only has it provided me with the experience in working in a team, but also has given me the opportunity to use the knowledge, which I had gained in the Software Development Life Cycle module, in practice.

In addition to this important impact on my self-development, the project has enabled me to appreciate the significance of every step of software life cycle, namely, setting the standards and creating documents concerning the requirements of the program.

Furthermore, I have learned to appreciate the importance of planning the division of duties and tasks, as well as time-management, even more.

What is particularly significant is the fact that the whole project has showed me and the whole group how crucial the communication and cooperation between the members of the group was. This aspect is especially important as far as the division of duties is concerned.

Moreover, the atmosphere of work and the work itself, as far as mine experience is concerned, was very satisfying, especially when it comes to the practical skills it has given me.

In fact, what should be stressed is that I attended almost every formal and informal meeting and the fact that I did testing table, class diagram and was one of the coders made me extremely focused on my tasks and required me to fulfil my duties conscientiously. It has particularly helped me to develop my time-managing and general skills relating to programming and documentation.

Generally, in addition to the above-mentioned benefits which this project has given me, it is worth stating that the experience it provided me with will be incredibly important as far as the Industrial Year is concerned. It is commonly known that group projects are particularly huge and crucial part of working as an IT. Therefore, it is worth noting that this project has enabled me to appreciate what it is like to be working in a team in reality and to find out how significant and how helpful the use of knowledge in software life cycle will be during my Industrial Year.

What is very important, it has made me aware of the practical problems and obstacles I might encounter while being a member of the group, but what is even more significant - how to overcome them effectively and how to eventually succeed.

To sum up, as I have indicated above, the whole project and work in my team went very well as far as my experience is taken into consideration. I have gained a great theoretical and practical knowledge as well as the necessary experience which will be extremely helpful and desirable to possess during my Industrial Year.

Personal Report - Joshua Bird

Role in group

In the first meeting I was allocated the role of group leader, with Thomas Hull being Deputy group leader. I believe this was due to the fact that I had already tried to contact the whole group before we had that meeting.

My main role in the group was to try and organise all of the group members and give them tasks to do, splitting up the workload into fair amounts between everyone. I did this by asking everyone what their strong and weak points were, and working around that decided and what parts of the project they should work on.

I completed sections of the work myself as well, such as the Gantt chart, this was used throughout as a guideline to follow, as it had milestones of what needed to be completed by when. I also completed a large portion of the final report myself, along with Dave and Phil.

Troubles encountered and resolution

Throughout the project I did encounter a few troubles, but this was expected. The main issue was probably working out what was needed of the final project early on. It was quite difficult to assign roles when I was not completely aware of what the full problem entailed. In order to solve this I read through the QA docs with Phil trying to break them down into easy to manage chunks of information, from that I was able to get an understanding of what was expected.

Another major trouble was time management, it was my role to make sure that I assigned work giving enough time for group members to finish that work in time for the deadlines. In order to try and solve this I tried to set tasks a bit earlier than necessary, and set deadlines for the group members earlier than the "official" deadline. The reason behind this was that it would give breathing room for the QA manager and I to look through the documents and update them where needed with enough time left to hand them in.

This leads to the next problem however, of group members not sticking to the dates I set, I would give them a deadline and without fail I would receive work later than I had planned, which made it difficult to manage the next set of documents. As the start of the first time I tried to be lenient because we had other assignments, however as time went on I had to insist on work being completed and I had to send multiple messages. This was after only managing to finish a document the night of the hand in.

I had a few issues with contacting group members, in order to solve that I had to find multiple ways of contacting a few of them, and had to send them messages as early as possible in order to get a response in time.

Overall success

Despite a few problems encountered along the way, I was pleased with the final product that we created. We had a project which performed all of the basic functional requirements, such as breeding/selling etc. I think that the group worked well together as the time went on and members became more confident talking between each other.

So to conclude, I believe that in the end we worked well as a team and achieved our goal. There are sections of the project that I think could have been improved on, but with the time that we had I think everyone performed very well.

Personal Reflective Report - Kamarus

Working as a group for these this module have given me such a great experience. This is my first time working as group which lead me to realize how important it is that teamwork and coordination among the member as to experience what real world software developer does in their line of work.

I was assigned to do the part of the documentation of the project by our project leader; Josh Bird. This role is seem appropriate for me as I have a minor knowledge of programming .The task given to me is seems advantage for me, since I'm doing Business Information Technology. From what I've learn in this project that it allow me to critically understanding the software lifecycle systems with the use of Information Technology to facilitate these system.

The first document that I've been assign was the Project Plan, I was responsible for the risk assessment which I've drawn up a table. The table consists of the threat which may rise during the development of the project and their mitigation to solve it. With the help of my project member, we brainstorm all the risk which I've received to be written up for the risk assessment. I've also put some point in my own points in the risk assessment.

The Second document that I've been assign was the Test Specification. In this document I was assign with Syzmon Stec to written up the test plan. Beside that, I was also responsible to written up the general approach to testing part of the Test Specification.

The third document that I've been assign in the Design Specification where I have to written up the architectural description of the project. In this task, I learn the blueprint of the program and how the program works. The final document that I've been assign to is the Documentation which is the final report. In this document, I was responsible for the project test report.

Apart from that, the teamwork and cooperation among the group member was great, I learn so much from them in their own way. Even though I have made some mistake, not being able to attend some of the informal meeting, I was truly grateful to have such a phenomenal project leader who keep me update of any changes. Even though I have any problems or any doubt with the project, the project leader was always there to help.

Overall, I hope that in the future, if I was given another group project; I hope that I could work with the same group.

Personal reflective report - Christian Morgan

I was allocated by our group leader, Josh Bird to do the documentation. I asked to be this as I wasn't the confident programmer in the group and I thought that would be the best role for me to do in the group. In the first semester I had quite a lot of work assigned to me as I was doing the documentation, which I thought was fair as the programmers will get the same amount of work when it comes to implementing it.

The first document I worked on was the project plan, in this document the group was allocated different section as different sections needed different inputs from the teams. In this document I wrote the objectives and the overview of the plan for the project. The second document I had to work on was the design documentation, with this document I had to work with the programmers in order to get an understanding of how the program is going to look in order to get a basic design of it. I didn't work on this document alone, others was allocated segments in this document as they had a better understanding of how they are going to design it.

I made the sequence diagrams in the document with help from Dave Haenze. The next document that needed to be completed was the testing documentation, in which we had to create test specifications. After all this was done, the programmers went and coded the game, in this part of the work, I had little to do as I had nothing to offer them to help. However as I did a lot more work in the beginning they were now doing their share which was fair. Throughout this whole group assignment we had regular group meetings, and I must admit we did struggle at times because of everyone has a rather busy schedule. With some of the members being part of societies and sports clubs, but we did compensate for this and rearranged meetings. We also had specific team member meetings which freed up other members who really didn't need to be in that meeting.

The whole group as a whole I thought worked very well as a team, especially since we didn't really know each other very well. I will also admit we as a group had a lot of different personalities in the group, which I felt helped the group rather than clashing like some personalities would. I also admit there were stressful times and some members did not get on, but we overcame it and I think we all acted maturely. Which is what you would expect from a real life software development team. The only one in the group who I felt that left the group down was Kamarus Alimin and this was only towards the end of the assignment. He didn't keep in contact with the group, didn't show up to meetings, which meant that other members of the group had to take his work load. He wasn't replying to our messages, which was a great shame since he worked quite well through it all.