Group 06 - Software Development Life Cycle Document

14th Febuary 2012

Amy Rebecca James, James Slater, Dan Mcguckin, Samuel Mills, Felix Farquharson,
Ben Brooks, Christopher Krzysztof Ilkow, Aiman Arafat
Project Coordinator: Nigel Hardy

Status: Release

Computer Science Department, University of Wales, Aberystwyth

1 Contents Page

Group 06 - Project Plan

23rd October 2012

Amy Rebecca James, James Slater, Dan Mcguckin, Samuel Mills, Felix Farquharson,
Ben Brooks, Aiman Arafat, Christopher Krzysztof Ilkow
Project Coordinator: Nigel Hardy

Config Ref: SE.QA.05B
Version: 1.5
Status: Release

Computer Science Department, University of Wales, Aberystwyth

Contents

1 Introduction

1.1 Purpose of the Document

This document describes the outline design for the Software Development Life Cycle Group Project 2012. It should be read in the context of the Group Project, taking into account the details of the group project assignment and the group project Quality Assurance (QA) plan [1].

1.2 Scope

The design specification splits the project into individual implementable parts and describes the interfaces and interaction between these components. The design specification refers to the Requirements Specification for the group project. It is important that this document is read by all members of the project group, especially the implementation team.

1.3 Objectives

The objectives of this document are:

- To identify and describe the main features of the Monster Mash game.
- To note the details of the criteria that the group project application must achieve.
- To administer interface details for each of the central classes in the Monster Mash game.

2 Overview of Proposed System

2.1 Design Choices

We were tasked to make a website, there are a few decisions that had to be made relating to the design of the website that are listed here.

2.1.1 Using the web as a platform

Using the web as a platform for this app means that the pages will be accessible by a large number of people, this means that the potential audience is larger because it is not limited to one machine like a Windows PC. The data needs to be centralised for this game to work and this is ideal because the data can be stored on a central machine or server. The web is fairly easy to develop for as there are a number of frameworks and platforms out there to use. This means that development can be faster and the application is more accessible to users. Designing websites for the web can be problematic and it requires a good sense of design and most importantly some experience with development. We should choose a method that allows us to keep the design consistent throughout the site and that allows us to keep it up to date easily as web design trends change quickly and often.

2.1.2 High Level

We chose to use tools that were high level however they came at a small performance cost. We decided this as a group and there were a number of reasons for doing this. Firstly because it allows developers to develop at a higher speed, which means less time until there is a working solution and faster development overall. It means less time working out bugs in the code because higher level tools can be more intuitive and often easier to debug. Using high level tools can be good for security if the platform has been tested thoroughly because it takes the burden away from the developer. As well as being easy for developers to learn at this stage it is also useful for anyone who has to maintain the code afterwards, less time is needed to figure out what particular features do and it's therefore easier to fix and troubleshoot issues. The only real downside of this is that there is a performance sacrifice at some stages. It was judged on all tools that we used that the performance cost is worth the time spent less on development.We decided to use the bootstrap library for creating a consistent design and we also used jQuery to help make functional JavaScript more quickly. In Java we are going to use DERBY server and client in order to reduce the amount of SQL that we will have to write to use the database.

2.1.3 User centric design

The application will work by presenting the user with a number of options which they can select and when the user selects an option a new option will be presented. The user is always at the center of what is going on the website so we are going to make sure that they are presented all of the possible actions and features they would expect. Most importantly it should be easy for the user to access what the user is expecting to do, which is fight monsters.

2.1.4 Accessible Aesthetics

The UI needs to be clear and understandable for the user. The links on the website will all need be obviously clickable and allow the user to navigate effectively. The general appearance needs to be free-flowing and user friendly, especially as the site is targeted towards students who may vary in age. The UI needs to work on a variety of platforms and we anticipate 20% of our traffic will be from mobile devices so we need to make sure we have a responsive design to suit it. An objective of ours is to ensure the game itself is fully playable on android and iOS mobile operating systems. As well as mobile browsers we need to consider the differences between the most popular browsers of today and we need to ensure our application is fully functioning in terms of appearance and performance. We are using the bootstrap library to help with some of these issues.

2.2 Choice of web platform

We wanted our platform to be well documented and detailed and to have materials to help introduce the platform to members of the team. We have decided to use the Glassfish Open Source version as our platform. Our platform needed to be able to produce a website so that multiple people could develop it effectively. We wanted to be able to deploy the website to a server somewhere and to have a platform that is well supported in production environments.Choosing a platform to support this was important to us as it is an essential and important feature of our application.We did not consider the Oracle Glassfish Server because it would cost to use it for the project and as expenditure is limited this was not suitable. We considered the following platforms:

- Glassfish Open Source Server
- Google App Engine
- Apache Tomcat

2.2.1 Glassfish Open Source Server

There are a number of benefits to Glassfish Open Source Serve in comparison to the other platform options available to us.The main two reasons for using this server are because it is an open source platform and most importantly because some members of our group have previous experience developing with it.Another reason is because we expect there will be support for this environment available. Both from the University and from the constrictions to the Glassfish project. Glassfish has many more features than Tomcat, which was the other open source option available to us. Spike testing was carried out and it was found that this piece of software was easy to use and appropriate for the nature of our project.

2.2.2 Google App Engine

The main reason we didn't choose this software is because it proved unreliable in tests. This software is also closed source and using it would mean that you rely upon Google when the application is released which was not suitable for our project.

2.2.3 Apache Tomcat

Tomcat was not as fully featured as Glassfish, and no one in the group had ever used it before which would result in lack of experience and there would be a steeper learning curve for the team.

2.3 High Level Architecture

2.3.1 Version Control

For version control we are going to be using Git. Git is a distributed version control system, which some members of the group already have experience with. Distributed version control systems give a slightly different development pattern which suited the qualities of a group better than SVN.Version control systems we considered:

- Git
- Bazaar
- Subversion

2.3.2 Integrated Development Environment

We have decided to use the NetBeans IDE because it is available free to users and it is the preference of the majority of the group. Modules are available for NetBeans to help with Version Control (Git) and JUnit.IDEs considered:

- Eclipse
- NetBeans

2.3.3 Documentation Tool

We decided to use LaTeX because it is widely supported, there is a template provided, and because it was preferred by the majority of the group. Also some members of the group have had experience with using LaTeX so group members could easily apply their existing knowledge. Methods of documentation we considered:

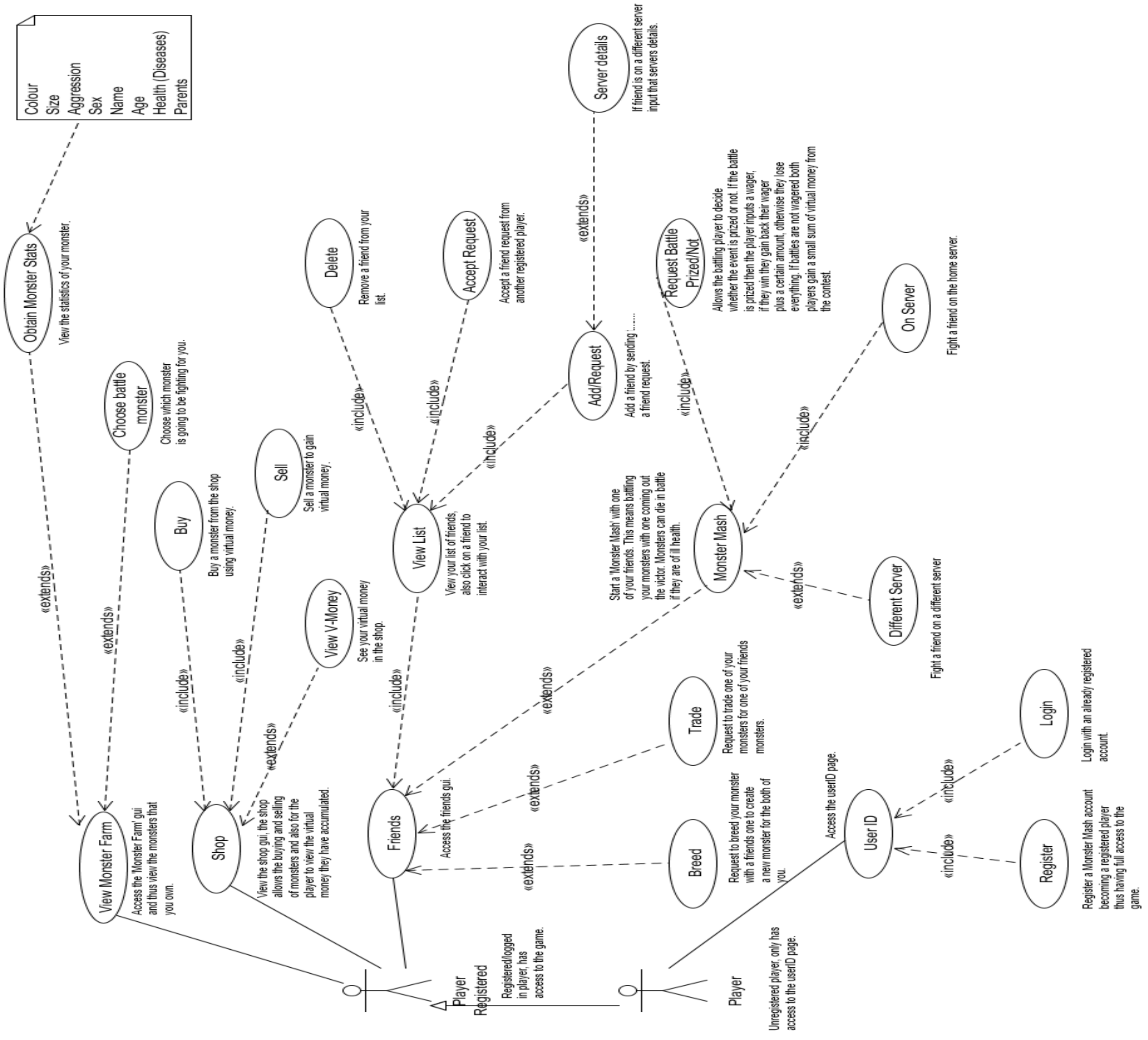
- LaTeX
- Open Office/Libre Office
- Microsoft Word

2.4 Description of Target User

The target user of our website will be young people. As this is an educational tool to inform students about the genetics and mechanics of breeding, the users will be typically aged between 11 and 16.We will have to make sure that text information on the website is user friendly and isn't too complicated and we will have to make sure that all content is appropriate. Other things to consider, are:

- Make sure that there are no really lengthy tasks to do
- Make sure that it will fit around the lifestyle of a young person of that age. ie. Around school, limited access to a computer.

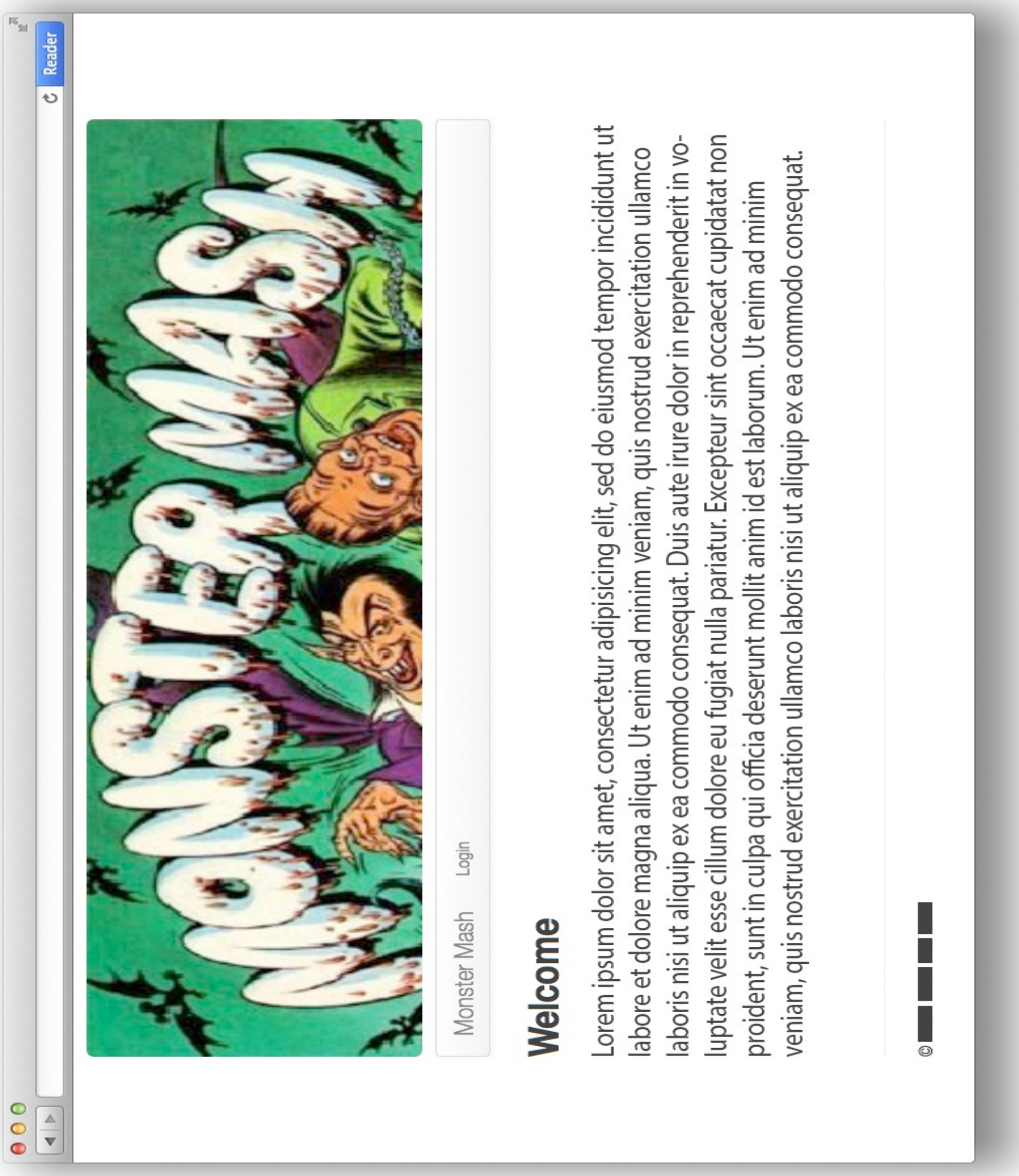
3 Use-case Diagram


Colour
Size
Aggression
Sex
Name
Age
Health (Diseases)
Parents
Obtain Monster Stats
View the statistics of your monster.
«extends»
View Monster Farm
Access the 'Monster Farm' gui and thus view the monsters that you own.
«extends»
Choose battle monster
Choose which monster is going to be fighting for you.
«include»
Buy
Buy a monster from the shop using virtual money.
Shop
View the shop gui, the shop allows the buying and selling of monsters and also for the player to view the virtual money they have accumulated.
«include»
Sell
Sell a monster to gain virtual money.
«extends»
View V-Money
See your virtual money in the shop.
Delete
Remove a friend from your list.
«include»
Friends
Access the friends gui.
«include»
View List
View your list of friends, also click on a friend to interact with your list.
«include»
Accept Request
Accept a friend request from another registered player.
«include»
Add/Request
Add a friend by sending :......
a friend request.
«extends»
Server details
If friend is on a different server input that servers details.
Player
Registered
Registered/logged in player, has access to the game.
«extends»
«extends»
«extends»
Start a 'Monster Mash' with one of your friends. This means battling your monsters with one coming out the victor. Monsters can die in battle if they are of ill health.
«include»
Request Battle Prized/Not
Allows the battling player to decide whether the event is prized or not. If the battle is prized then the player inputs a wager, if they win they gain back their wager plus a certain amount, otherwise they lose everything. If battles are not wagered both players gain a small sum of virtual money from the contest.
Breed
Request to breed your monster with a friends one to create a new monster for the both of you.
Trade
Request to trade one of your monsters for one of your friends monsters.
Monster Mash
Player
Unregistered player, only has access to the userID page.
«extends»
«include»
Access the userID page.
User ID
Different Server
Fight a friend on a different server
On Server
Fight a friend on the home server.
«include»
«include»
Register
Register a Monster Mash account becoming a registered player thus having full access to the game.
Login
Login with an already registered account.

4 User Interface Design

4.1 Index Page

This screen is the page that a user will see when visiting the Monster Mash website. The user will be provided with a link on the menu bar to Login into the website to allow them ti play the application. If they do not have an account, they can create a new one on the Login page too.



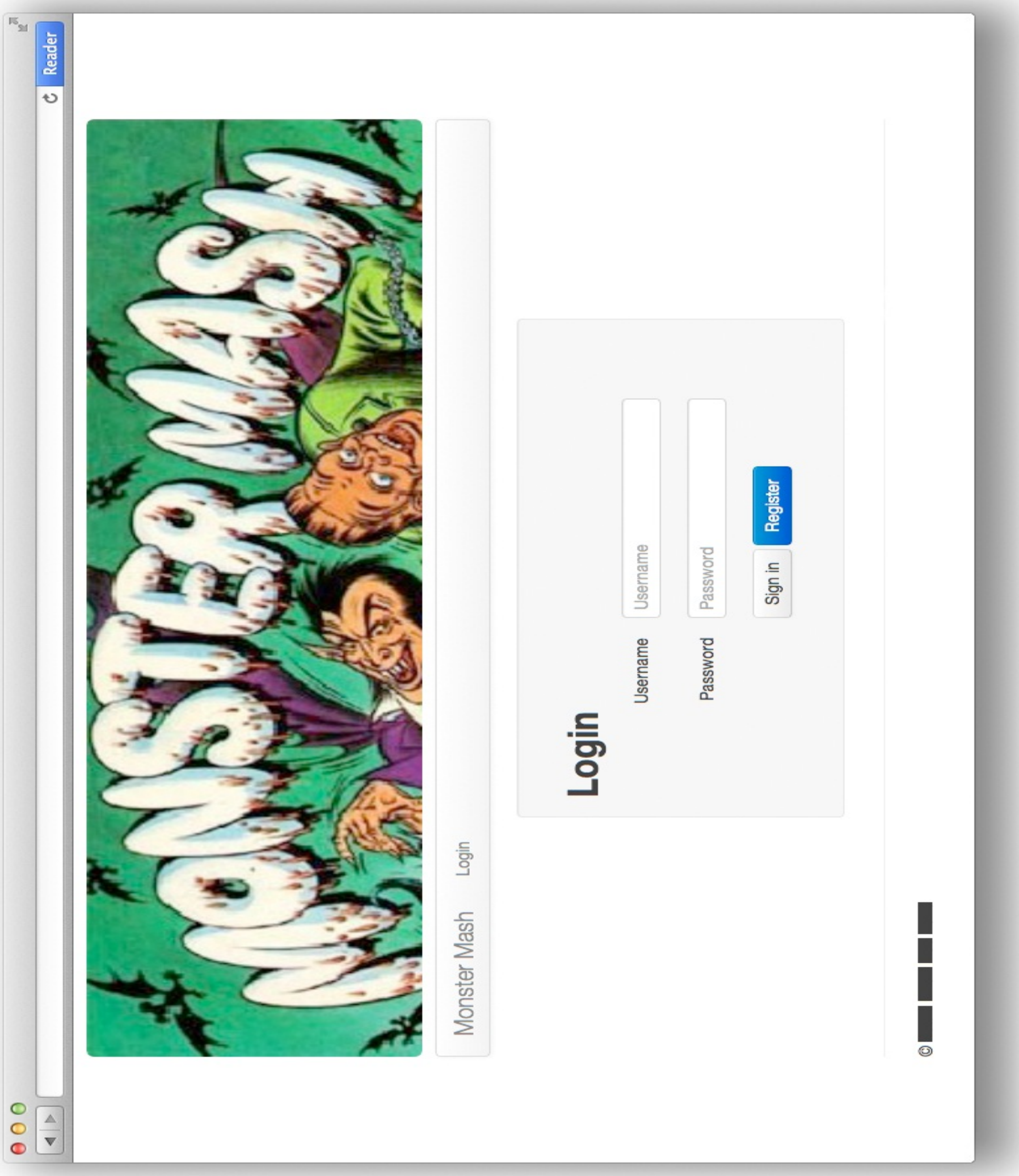
Welcome

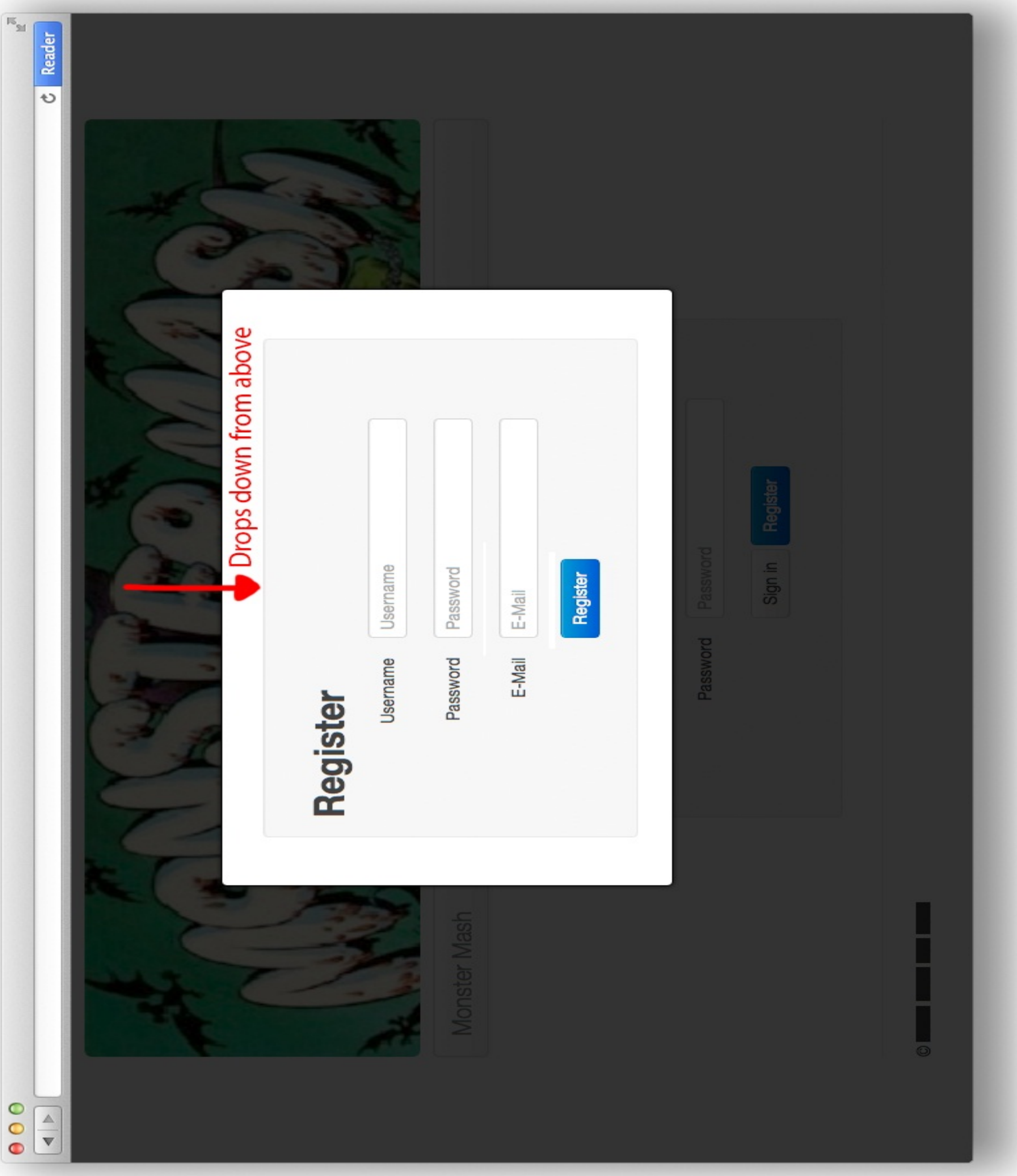
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

© [redacted]

4.2 Login/Register Page

On this page, a user can login with an existing account or register a new account by clicking on the Register button. This brings up a dialog box on top of the existing page which lets a user register. Upon successful registration they are brought back to the login page so that they can proceed to login.

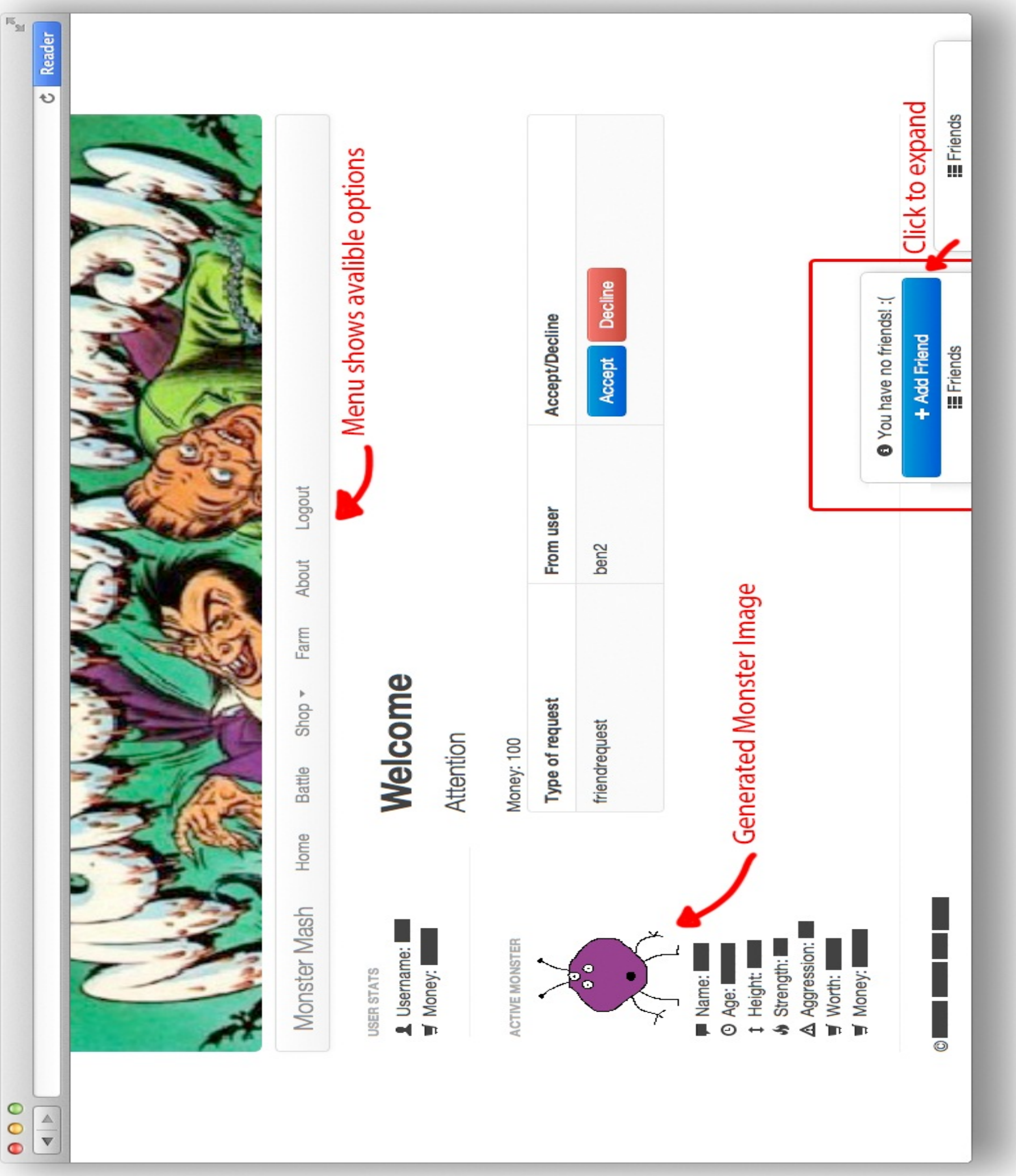




Drops down from above

4.3 Home/Add Friend

This is the page a user views once they have logged in. The menu bar now has additional links that a user can follow which indicates to the user they have successfully logged in. On the left it shows their personal user statistics, and their primary monster's statistics. On this page, friend requests, battle requests and other kinds of requests are displayed with an Accept/Decline button. In the bottom right hand corner is a friends list, to bring up the list of friends that a user has. If the user clicks on it, it slides upwards to display all of the user's friends. If they have none, a message to the user is displayed instead and a button to add friends is shown. Upon clicking this, a window slides down with fields to enable the user to add a friend.When typing in the user name box, a list drops down to suggest friend names to add. This list is populated from the user database.



Menu shows avalible options

Welcome

Attention

Money: 100

Type of request	From user	Accept/Decline
friendrequest	ben2	Accept Decline

USER STATS

Username: [redacted]

Money: [redacted]

ACTIVE MONSTER

Name: [redacted]

Age: [redacted]

Height: [redacted]

Strength: [redacted]

Aggression: [redacted]

Worth: [redacted]

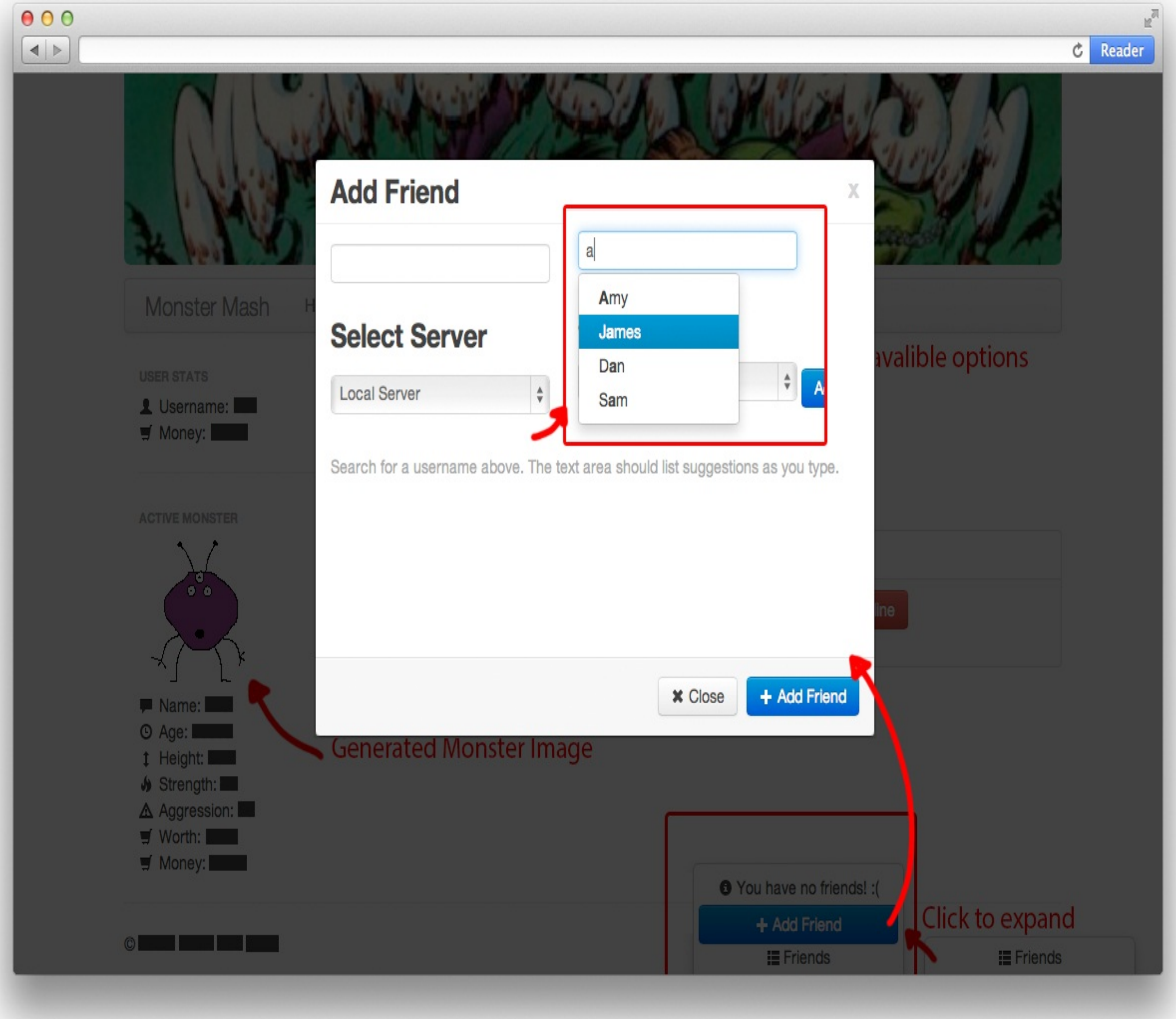
Money: [redacted]

Generated Monster Image

You have no friends! :(

Click to expand

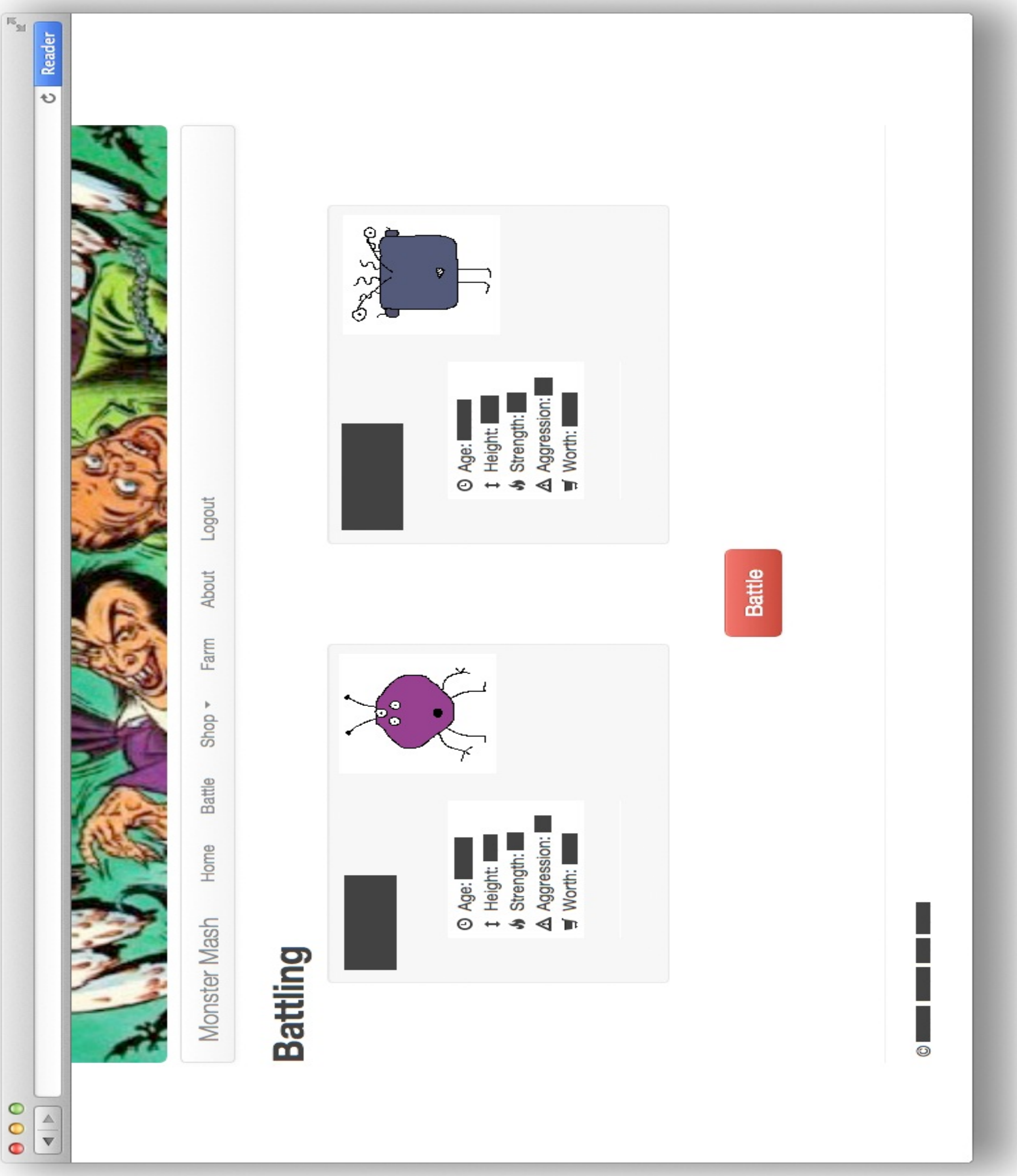
© [redacted]



Search for a username above. The text area should list suggestions as you type.

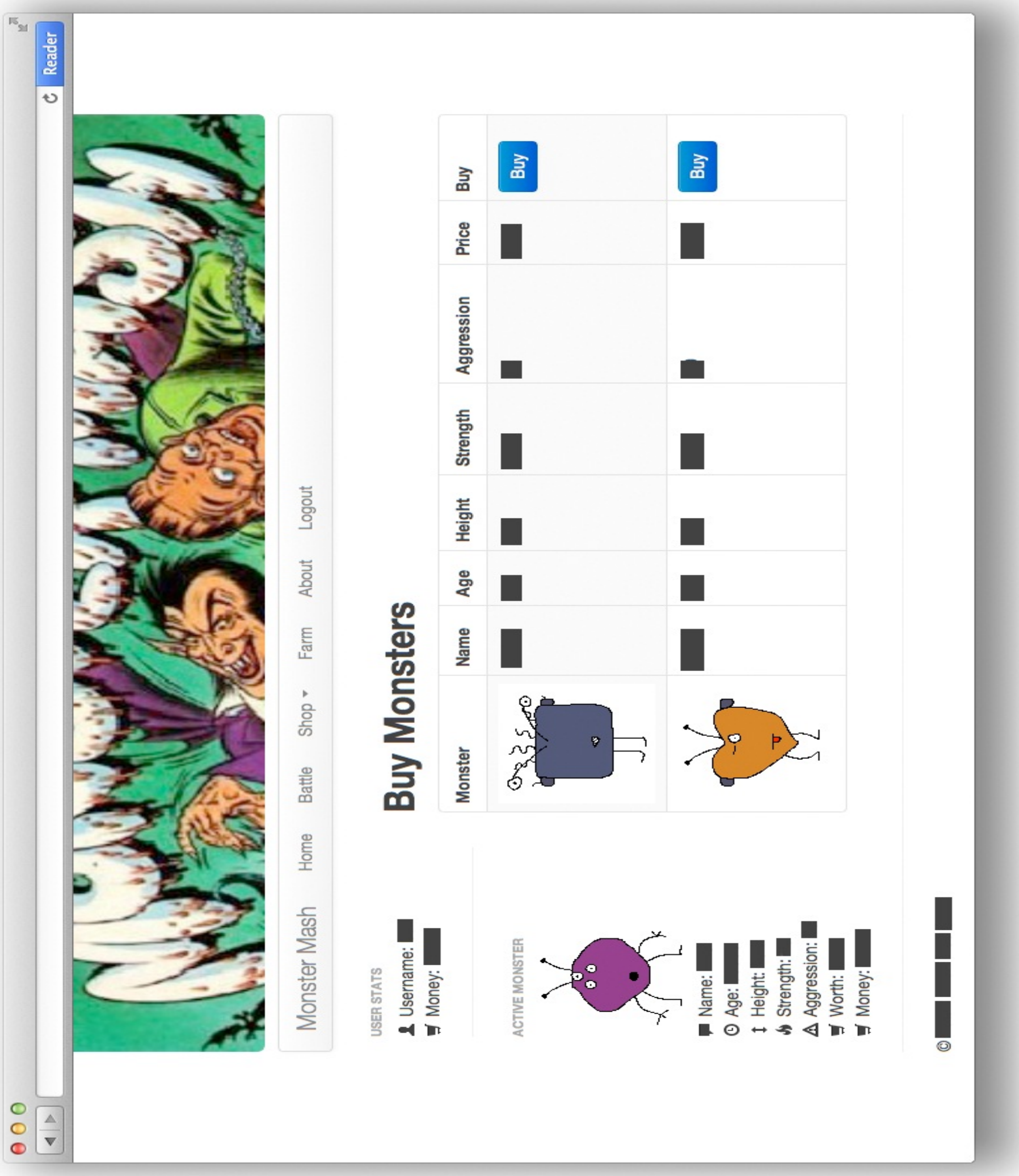
4.4 Battle Page

This is the page a user will see when they enter a battle. It shows each monster side by slide, along with their statistics, and has a "Battle" button to initiate the battle.



4.5 Shop Page

A user can buy or sell monsters in the Monster Shop. Their own monsters will be displayed in a table with an option to sell each of them, or alternatively, a list of unowned monsters can be displayed with an option to buy them. These are automatically added to the shop database when a user creates an account to add populate the Shop. All of the statistics are visible in the table so a user knows what attributes the monster they are buying has.



USER STATS

Username: [redacted]
Money: [redacted]

ACTIVE MONSTER

Name: [redacted]
Age: [redacted]
Height: [redacted]
Strength: [redacted]
Aggression: [redacted]
Worth: [redacted]
Money: [redacted]

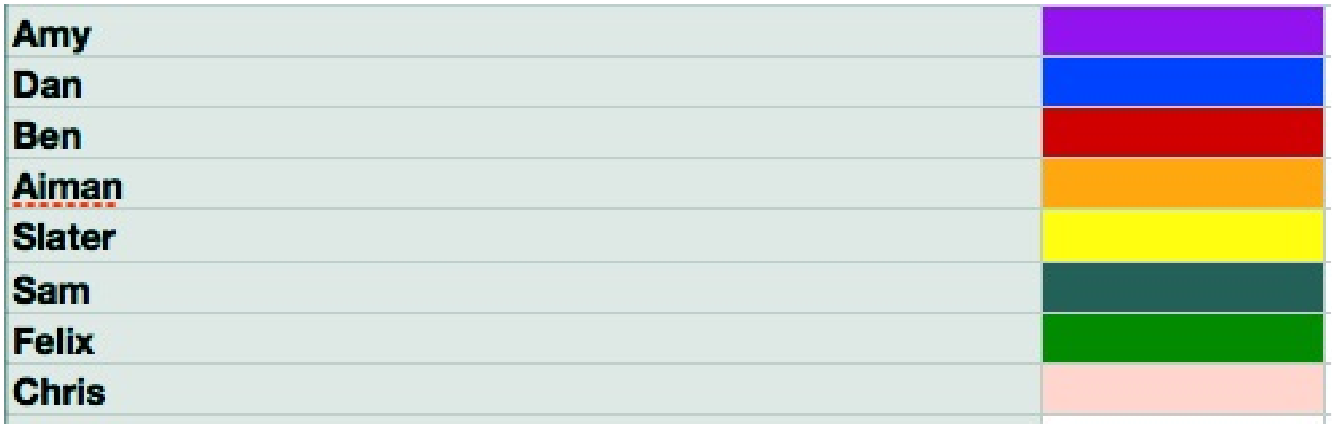
Buy Monsters

Monster	Name	Age	Height	Strength	Aggression	Price	Buy
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	Buy
	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	Buy

© [redacted]

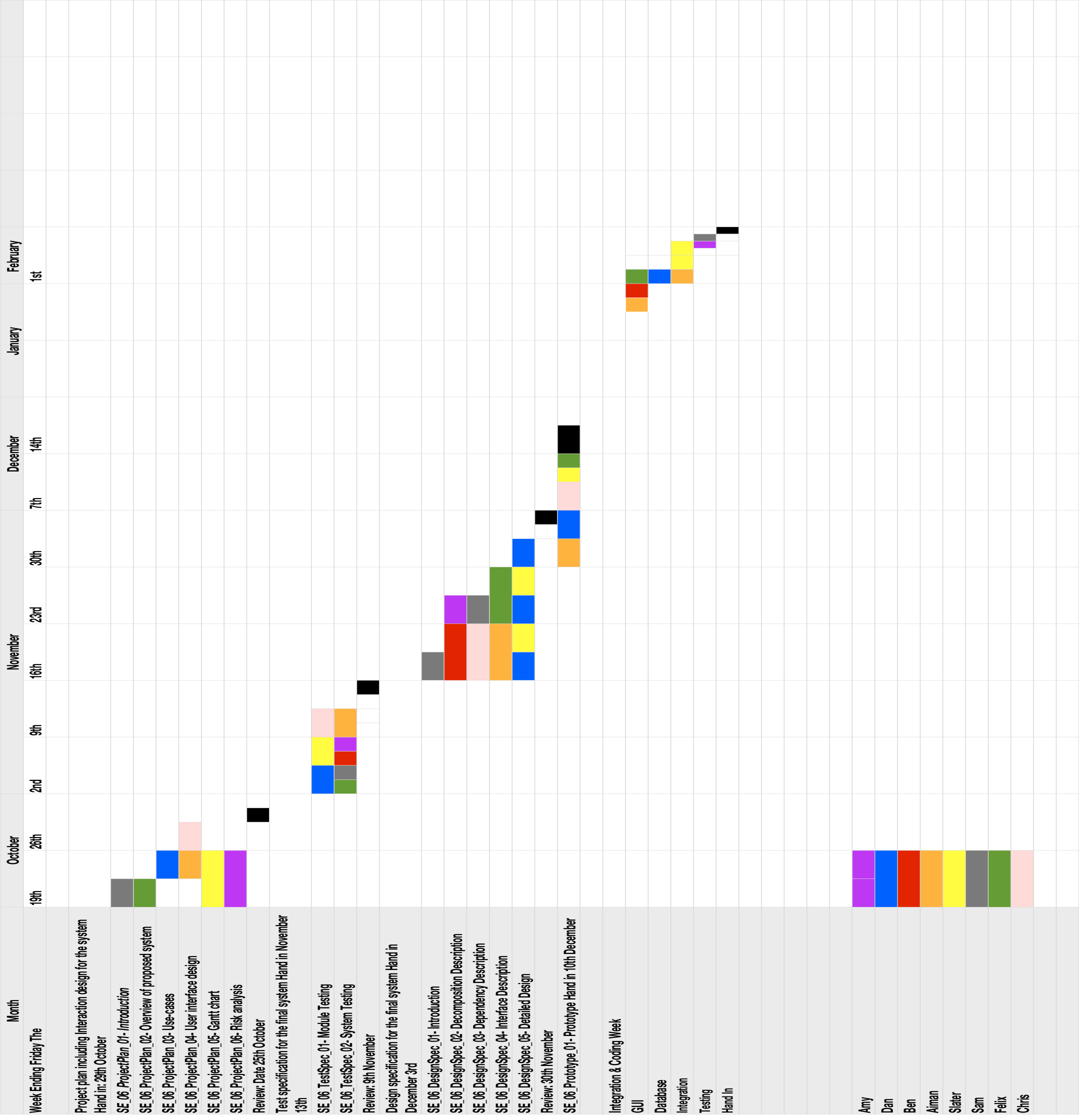
5 Gantt Chart

5.1 Key for Gantt Chart



Amy	🟪
Dan	🟦
Ben	🟥
Aiman	🟧
Slater	🟨
Sam	⬛
Felix	🟩
Chris	⬜

5.2 Gantt Chart



6 Risk Analysis

When planning and developing a piece of Software, there are many different aspects of the development process. This is why Software Engineers need to try and predict and possibility and minimise the chances of problems having any serious effect on the development of an application e.g. - losing time, unable to make a deadline or even to the extent of a particular part of the project being left incomplete or not working.
Primarily it is the job of the Project Group to identify these risks and to remain committed so that they can be avoided at all costs or at least minimised in regards of damage to development.

Description	Possible Outcomes	Actions to take
Time allocation being unrealistic.	Team members unable to finish their work or finish it to the best of their ability.	Plan correctly, ensure that all members are happy that work can be completed in that length of time.
Communication - All members need to have an idea of what they and other group members are doing in terms of development.	Lack of communication can lead to tasks being incomplete.	Smooth running project and deadline being met.
Illness of key group members.	Group members are unsure of what needs to be done.	Deputy Project Manager needs to know what the Project Leader has planned so that they can carry on without major disruption.
Members of the group leaving the project group due to unforeseen circumstances.	This could result in work being left incomplete halfway through, work having to be reassigned to group members by the project leader and slowing down overall progress.	Have a contingency plan in place that can deal with the event of group members leaving. Have a backup project plan which has work already reallocated in the event of a person leaving.
Learning how to use new programs.	Simple tasks may be more time consuming. E.g. in Latex and GitHub.	Use of books and other group members to ensure they understand new technology in a suitable timespace.
Software and Hardware difficulties.	This could result in data and progress being lost and not being able to develop the application effectively.	Have alternative hardware and software options. E.g Personal Computer or other Software.
Difficulty integrating different group members code.	Hard to pull code together and integrating the GUI.	Ensure that coders work closely together so that this problem is mitigated.
Group repository inaccessible/technical difficulties.	Can lead to missed deadlines and lack of development.	All members should backup their own work individually so that these issues have minimal effect.
Complex algorithms.	Can lead to code that contains bugs that are difficult to fix or can't be fixed.	Write small segments of code at a time and then compile. This make it easier compared to a large segment of code being compiled and containing bugs as there is less code to debug.
Inaccessible network.	Leads to lost time within the project.	Saving files locally as well as on server. Also pulling and pushing to Git regularly to ensure latest version is available.

7 References

[1] Software Engineering Group Project Plan Specification Standards. Bernard Tiddeman. SE.QA.05B. 1.1 Release

8 Document History

Table 1: Document History

Version	CCF No.	Date	Changes made to document	Changed by:
1.0	N/A	25/10/2012	Creating Project Plan	arj18
1.1	N/A	29/10/2012	Making changes from review	arj18
1.2	N/A	30/10/2012	Adding final GUI designs	arj18
1.3	N/A	28/01/2013	Adding feedback changes to document	arj18
1.4	N/A	12/02/2013	Adding revised GUI designs.	arj8
1.5	N/A	13/02/2013	Spell Checked and added some Risk Analysis	sam39

Group 06 - Test Specification

15th October 2012

Amy Rebecca James, Samuel Mills, Felix Farquharson,
Ben Brooks, Aiman Arafat, James Slater, Ben Brooks, Dan McGuckin, Christopher Krzysztof Ilkow

Project Coordinator: Nigel Hardy
Config Ref: SE.QA.06
Version: 1.4
Status: Release

Computer Science Department, University of Wales, Aberystwyth

Contents

1 Introduction

1.1 Purpose of the Document

The purpose of this document will be to establish any defects or flaws which may exist within our Monster Mash application, and whether the features of the application are going to be functional and operable for the user. Our testing document will explore all features and actions that can be carried out within the application and identify possible problems or errors which the user could experience.

1.2 Scope

The testing specification is split up into separate tables which represent each feature or action of the game and identifies the test content, input, output and pass criteria of each test. These features include:

- Login/Edit Account Details Tests
- Homepage/Friends List Tests
- Battle Screen Test
- Farm Test
- Shop Test

It is important that this document is read by all members of the project group, especially the Design and Testing team.

1.3 Objectives

The objectives of this document are:

- To identify any errors, liabilities and flaws that a user may encounter when interacting with the Monster Mash application.
- To show these weaknesses and defects in the format of tables where the features of the Monster Mash game are thoroughly tested and certified to ensure they operate successfully.
- To show the test content, input, output, pass criteria and general comments of each test.

2 Test Tables

2.1 Login/Edit Account Tests

To test that our Login works we will have already have saved two users into our database to allow us to carry out extensive testing on the Login and account features. They will each have a username and password and we will run various tests on these users to test that the interface and server handles the information entered as we would expect.

2.2 Created Users Details

John - Username: John87 Password: computer
Gareth - Username: Gaz19 Password: monster

3 Creating an account

To test our registration system works we will create a user. If this test successful the username and password of the created user will be stored in the database and user will be able to Login.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-001	FR7	Creating User Account	The user will enter desired username and password.	The details should be saved and added to the database.	The user can use these details to access the Monster Mash application.
SE-N06-002	FR7	Creating an account with characters other than A-Z, a-z, 0-9 or an underscore in the user name.	User should create a user name containing an exclamation mark for example.	An error message should appear saying that the user name cannot be created and it should explain the reason for this error to the user.	The user name will only be created if it contains valid and accepted characters.
SE-N06-003	FR7	Testing that the account has been created successfully.	The user will enter their Login details and click the Login button.	User should be taken to the application homepage.	The user should be logged in and be able to start playing the Monster Mash application.
SE-N06-004	FR7	Creating an account using a username that has already been taken.	User attempts to create an account using a username that is already taken.	An error message should appear saying that the user name cannot be created and it should explain the reason for this error to the user.	The user should not be able create an account.
SE-N06-005	FR7	Creating an account with an email address that is already registered to a user.	The user will attempt to create an account using an email address already registered.	An error message should appear alerting the user.	The user should not be able to create an account.
SE-N06-006	FR7	Case sensitive login.	User will attempt to login using a capital or lower case letters incorrectly.	An error message should appear saying that the user name cannot be logged in and it should explain the reason for this error to the user.	The user should not be allowed to log in.

3.1 Testing User Login Information

The following test table will investigate how the program handles incorrect Login details. Although the table only accounts for one, each test will be carried out on both of the users stored in the database already (John and Gareth).

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-007	FR7	Login using correct details.	User will enter correct login details and click Login button.	The correct details will be accepted by the server.	User will be taken to Homepage and able to access the features of the Monster Mash application.
SE-N06-008	FR7	Login using incorrect password but correct username	User will enter correct username with incorrect password and click the login button.	An error message will appear informing the user that the details entered are incorrect.	The user won't be logged into their account until the correct details are entered.
SE-N06-009	FR7	Ensure that both the username and password have been entered into the login fields.	User will attempt to login with one of the fields left empty.	An error message alerting user of the problem should appear.	User shouldn't be allowed to log in until both fields and filled in correctly.

4 Homepage/Friend List Tests

The following tests are designed to assess the option available to each user from the Homepage and from their Friends Lists. For the tests below it is assumed that there is already a database of users available for testing.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-010	FR2	Locate all available links to the user and interact with them and test their functionality..	All of the link locations within the application.	The link goes to the correct page.	If the link is navigating to the correct location and the page is active and functioning.
SE-N06-011	FR1	Test if the user can logout successfully.	Select the logout link.	The page template/source.	The page should identify the fact the user is logged out and is unavailable to interact with the application.
SE-N06-012	FR2	Test that the graphical user interface and graphics show correctly to the user.	The homepage should be displayed correctly as intended to the user.	Pages should be displayed correctly.	The graphics and images should be consistent with the original design.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-013	FR11	Check that the friends list displays correctly to the user when it is not sorted by any custom settings.	The page containing the friends list should be loaded and there should be no specific sorts or filters specified.	The output page source for this page with no friends, lots of friends, and with the maximum amount of friends.	The pages should return standards compliant HTML with no errors encountered. The sorts should be correct and a visual inspection should be carried out also.
SE-N06-014	FR11	Check the friends list displays correctly when it is sorted by recent activity.	The page containing the friends list should be loaded and there should be only recent activity showed.	The output page source for this page with no friends, lots of friends, and with the maximum amount of friends.	The pages should return standards compliant HTML with no errors encountered. The sorts should be correct and a visual inspection should also be carried out.
SE-N06-015	FR9	Check the add new friend function works correctly.	A friend should be added using the friends list form.	The updated friends list containing all the friends, or the database.	The friends list after the friend has been added should now contain the newly added friend.
SE-N06-016	FR6	Check the delete friend function.	One of the friends selected should be deleted using the friends list delete function.	The friends list.	The friends list should not contain the deleted friend.

5 Battle Screen Tests

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-017	FR10	Check that a battle has finished and allows the user to select a friend to start another battle.	The user clicks on Start another battle button from the battle screen.	The monster battle screen swaps into friend list allowing the user to choose one to fight against only if previously selected battle has taken place.	Allows the user to user to select another friend to battle against after finishing current battle.
SE-N06-018		Check that Monster Farm button has been pressed.	User clicks on Monster Farm button from the battle screen.	The monster battle screen swaps into the Monster Farm screen and allows the user to check stats or change the current monster to fight with.	Takes the user to the Monster Farm screen and allows the user to choose monster or check the stats.
SE-N06-019	FR8	Check that Start The Fight buttons has been pressed from the battle screen and starts the battle.	User clicks on Start The Fight button from the battle screen.	The battle screen shows both monsters, the result of the battle and any earnings or rewards.	Generates the battle, show victorious monster, earn rewards.

5.1 Battle Screen Tests Continued

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-020	FR8	Check that the Friend button has been pressed from the side of the battle screen and displays a message.	User clicks on Friend button from battle screen.	An error message is displayed to the user. User cannot leave the battle screen.	Display error message.
SE-N06-021	FR8	Check that Shop button has been pressed from the battle screen.	User clicks on Shop button from battle screen.	An error message is displayed to the user. User cannot leave the battle screen.	Display error message.
SE-N06-022	FR9	Check that Add friend button has been pressed from the battle screen.	User clicks on Add Friend button from battle screen.	An error message is displayed to the user. User cannot leave the battle screen.	Display error message.

6 Farm Tests

These tests are designed to test the Monster Farm of the Monster Mash application. They test various aspects such as viewing monsters, their stats and picking a primary battle monster.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-023	FR1	Obtain your monster's statistics	Go to the "Monster Farm" page"	A list of your monsters with associated statistics should be displayed	The page correctly displays your monsters and their statistics.
SE-N06-024	FR2	Choose your primary battle monster (i.e. the monster you use to fight).	Click on the check box/button next to the monster you wish to make primary battle monster.	That monster should now be selected to be your primary battle monster.	The chosen monster is now your primary battle monster.
SE-N06-025	FR3	Exit the Monster Farm	Click on the available navigational links which take you to a particular screen.	You should exit the Monster Farm and be taken to the location you chose.	The page changes to the location your chosen location.

7 Breeding Tests

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-026	FR6	Monster statistics are different for offspring.	User should breed a monster.	A new monster should appear in the Monster Farm.	The new monster should have completely different statistics to the parents.
SE-N06-027	FR6	Breeding with two different monsters.	User should select two different monsters to breed with.	A new monster should be created.	A new monster should be listed in the Monster Farm.
SE-N06-028	FR6	Breeding with two instances of the same monster.	User should select the same monster twice to breed with.	An error message should appear and not allow this.	User should be asked to enter two different monsters.
SE-N06-029	FR6	Ensuring the user has enough money to breed a monster.	User selects a monster to breed with.	If the user has insufficient funds, an error message should appear.	User should not be allowed to breed with a monster.
SE-N06-030	FR6	The user must enter unique name for each monster.	User enters the a name for monster already in use.	An error message should appear.	User should not be allowed to have a monster with the same name.

8 Shop Tests

The following tests will be featuring the Shop aspect of the Monster Mash application. The Shop will only be accessible to registered users. Users will then be presented with a menu which has the following options; Monster Farm, Shop and a Friends list. Within the Shop category the user will be presented with a variety of options. The main purpose of the shop is to allow the user to Buy and Sell monsters using V-Money that they have gained from battles. Users can Buy monsters from the Shop which will then be added to their Monster Farm and the price in which the monster is worth will be deducted from the users V-Money total. The user will also be able to Sell monsters which they own and the price in which the monsters are worth is calculated by taking into account the monsters statistics. If the user sells a monster, the monster will be removed from the users Monster Farm and the correct amount of currency added to the users V-Money total. It is worth mentioning that tests will be carried out using a preloaded database of monsters to avoid the user having no monsters to Buy/Sell. The final feature of the Shop will be the display of the users V-Money which will show how much currency the user has to spend within the Shop.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-031	FR8	Check that the Shop GUI is displayed when the user clicks Shop from the homepage.	User clicks Shop button from the homepage.	The shop GUI should now be displayed to the user, including a list of monsters which the user can Buy and Sell and the ability to View V-Money.	Shop GUI is correctly displayed.
SE-N06-032	FR5	Check that the Sell function works correctly.	User selects what monster they chose to sell in the Shop GUI and presses the Sell button.	The monster should have been removed from the users Monster Farm and the amount of V-Money that the monster is worth based on statistics should be added to the users V-Money Total.	The correct amount of V-Money has been added to the users V-Money total and the Monster sold has been removed from the users Monster Farm.
SE-N06-033	FR8	Check that the view V-Money function works correctly.	Within the Shop GUI, the user should be able to view V-Money on the GUI this and shows how much currency the user has.	The user should be displayed their correct amount of V-Money in the GUI.	The user is successfully displayed their total V-Money amount.

8.1 Shop Test Tables Continued

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-034	FR6	Check that an error message is displayed if the user attempts to purchase a Monster from the Shop but they don't have enough V-Money.	User views the monsters available from the Shop list, selects the Monster desired and presses the Buy button and an error is displayed due to insufficient funds to purchase that monster.	An error message should be displayed informing the user they have Insufficient funds to purchase the monster desired.	An error message is successfully displayed to the user informing them they have insufficient funds to purchase the monster they desire.
SE-N06-035	FR6	Check that the Ok button works on the error message regarding the user having Insufficient Funds when trying to purchase a monster so they can return to the Shop GUI.	The user clicks the Ok button to acknowledge the error message and then returns to the Shop GUI.	Once the user has selected the Ok button, they should be returned to the Shop GUI.	The user presses Ok and they are successfully returned to the Shop GUI.
SE-N06-036	FR6	Check that the preloaded monsters from the database within the Shop are purchasable.	The user selects one of the preloaded monsters from the database and selects Buy.	The user should be able to purchase the monsters from the preloaded database. The correct V-Money from their total should have been removed and the newly acquired monster should now be in the users Monster Farm.	The monster from the preloaded database should now be in the users Monster Farm.

8.2 Shop Test Tables Continued

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria
SE-N06-037	FR6	Check that the preloaded database of Monsters within the Monster Farm are sellable.	The user selects one of the preloaded monsters from the database in the Monster Farm and selects Sell.	The user should be able to sell the monsters from the preloaded database (Monster Farm). The correct V-Money should have been added and to their total and the Monster selected to sell should have been removed from the users Monster Farm.	The monster from the preloaded database has been removed from the users monster farm and the users V-Money total should be correct.

9 References

1. *Software Engineering Group Projects Test Procedure Standards.. C.J.Price and N.W.Hardy. SE.QA.06.Release.*

10 Document History

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2012-11-12	Initial creation	beb12
1.1	N/A	2012-11-15	Adding test tables	arj18
1.2	N/A	2013-01-28	Making changes from hand in feedback	arj18
1.3	N/A	2013-02-11	Making outcomes of tests from coding week.	arj18
1.4	N/A	2013-02-13	Rearranged some tables.	sam39

Group 06 - Design Specification

3rd December 2012

Amy Rebecca James, James Slater, Dan Mcguckin, Samuel Mills, Felix Farquharson,
Ben Brooks, Christopher Krzysztof Ilkow, Aiman Arafat
Project Coordinator: Nigel Hardy

Version: 1.2
Status: Release

Computer Science Department, University of Wales, Aberystwyth

Contents

1 Introduction

1.1 Purpose of the Document

The main purpose of this document is to identify and describe the information in the design specification document produced by CS22120 Group 06. Each major section of the design will be individually described in detail and the document will explain the design theory behind each individual feature. It will also act as a tool for the group to look back on to keep track of current and future developments.

1.2 Scope

The design specification shows the specific features in relation to the Monster Mash game. This document explains the definitions of each component within the design and explains the details of each feature. It is important that this document is read by all members of the project group, especially the Design team.

1.3 Objectives

The main objective of this document is to reinforce the production of a design specification. It has to be complete, up to date and an accurate interpretation of the users requirements. Another objective is to provide an explanation of the design elements necessary for the implementation and development phases. These will generally include the outline structure, decomposition description, dependency description, interface description and a final detailed design.

2 Outline Structure

1 Homepage (Logged Out)
- 1.1 Login link
- 1.2 Homepage link

2 Homepage (Logged In)
- 2.1 Homepage
- 2.2 Fight (Logged In)
  - 2.1.1 Choose two Monsters
  - 2.2.2 Battle Screen

3 Shop (Logged In)
- 3.1 Buy
  - 3.1.1 Choose a monster and click buy.
- 3.2 Sell
  - 3.2.1 Choose a monster and click sell.

4 Farm (Logged In)
- 4.1 Breed - Choose a monster and click buy.

5 About Page (Logged In)

6 Logout (Logged In)
- 6.1 Homepage (Logged Out)

3 Decomposition Description

3.1 Applications in System

In our Monster Mash design, there will be one single application. This handles everything from serving JSPs that a user can see in their browser, to the database connections, inter-server connections and data processing.

4 Significant classes in each package

4.1 Significant classes in the Model package

4.1.1 User

The User class contains fields to hold the user's data for attributes such as Name, Password, Money, Friends, etc. Each user can have between 1 and 6 monsters which can be purchased and sold to the farm. You can also set the user's default battle monster from the changeBattleMonster method in this class. A user is created using the UserFactory class.

4.1.2 Monster

This class contains all of the monster's stats. For example, height, age, strength, sex, etc. A monster is created using the MonsterFactory class.

4.1.3 PersistManager

PersistManager is the class which can create, update or delete a user or monster. It is also used to search for users or monsters and to start or stop the main application. It is also this class which talks to the User database, Monster database and Shop database.

4.2 Significant classes in the Controller package

4.2.1 Login

Login contains methods that will allow a user to login or register themselves as a user.

4.2.2 Shop

Shop contains methods that let a user buy or sell monsters.

4.2.3 FriendsList

FriendsList contains methods for managing a user's friends. E.g. addFriend, removeFriend, getFriend, etc.

4.2.4 Worth

Calculates monster worth based on attributes and age. Also assesses monster age.

4.2.5 Breed

This breeds two monsters and creates an offspring.

4.2.6 makeAttribute Function

Creates new attributes for the new monster based on parents stats.

4.2.7 generalAssess

Assesses a monster

4.2.8 passToPM

Adds new monster to database.

4.2.9 determineWinner

Selects winning monster.

4.2.10 randomizeProbabilities

Adds some randomness to a battle between two monsters.

4.3 compareAttributesRetM1

Compares monsters attributes and returns a percentage of likelihood for monster one to win. (100-m1 is m2 likelihood)

4.3.1 BuyMonster

Purchases the monster, takes monster from shop and adds it to user's farm and updates user's vMoney.

4.3.2 SellMonster

Sells the monster, takes the monster from the user's Farm, adds it to Shop and updates the vMoney.

4.3.3 Logging Checker

Checks how long it has been since user made an action, after a given amount of time it will log the user off if they are not active.

4.3.4 MonsterListCycle

Ages a users primary monster whilst a user is logged on.

4.3.5 SimulateFight

SimulateFight has methods to calculate a winner of a fight between two monsters and calculates the prize amount of a fight, which is given to the winner.

4.4 Mapping from requirements to classes

Requirement	Classes Providing Requirement
FR1 (Server-based authentication)	Login, CreateMonsterAndUser, PersistManager, User and UserFactory
FR2 (Server friends list)	FriendsList and PersistManager
FR3 (Server monster list)	CreateMonsterAndUser, PersistManager, User and Monster
FR4 (Server monster mash management)	SimulateFight, FriendsList, PersistManager, User and Monster
FR5 (Server-server communication)	PersistManager
FR6 (Client options)	Login, Shop, FriendsList, Breed, SimulateFight and PersistManager
FR7 (Startup of software in browser)	Login and PersistManager
FR8 (Game display in browser)	FriendsList, Breed, SimulateFight and PersistManager
FR9 (Friend matching)	FriendsList, PersistManager
FR10 (Fight notifications)	SimulateFight and PersistManager
FR11 (Friends rich list)	FriendsList

5 Dependency Description

5.1 Outline

The following section of the document will explain how the parts of the system integrate together and how they are dependent on each other. All parts of the system will be heavily dependent on the database, as all user details will be stored in it, such as their login details, monsters, V-Money and a list of friends.

5.2 User

In order for the user to play the game and access its internal components, it is a requirement for the user to be logged in. Users will not be able to create a monster, add friends or sell/buy monsters until they are logged in. If this is the case, an error would be returned to the user and the user would be asked to register their username and password to the database, which would then allow access to the application itself.

5.3 Login/Register

Login and register methods will be entirely dependent on the database classes, where the task is to ensure that the details are entered and retrieved correctly allowing user authentication.

5.4 Shop

Buying a monster will not work when the user's V-Money is below the purchase price required. Breed class is dependent on two monsters, when less than two monsters are supplied, the class will return an error to the user and would not proceed further. SimulateFight class also needs two monsters as a minimal requirement for it to be executed, otherwise an error is produced to the user and the user can't proceed further.

5.5 Database

PersistManager is a class that will communicate with the database itself, from which it is going to retrieve the data. The Persist Manager is dependent on the User and Monster classes. The database is set up using Java Persistence API using the open source Hibernate Library. It runs using JavaDB or derby. This changed from HyperSQL due to connection problems. The database tables are created using persistable 'entity' classes through a persistence.xml file. The PersistManager class handles interactions between the database and the server. The database is reliant on the PersistManager and vice versa.

6 Interface Description

6.1 Model Package

The Model package contains the classes that deal with data representation. It will also contain all connections to the database, and methods of using all of the data types the group may need to use in the controller class at a later time.

6.1.1 PersistManager Class

Public class that handles transferring the data currently active in the application to persistent data in a database and it also handles retrieving the data from the database. There are 6 public functions, that are listed below. It uses the Monster class and it works by translating a Monster instance into a set of database friendly fields. It is used by many of the classes in the Controller package, so that they can retain information and process it at a later time.

init Function A public function that when called with initialize the class and instantiates the Persistence class to deal with the database. It takes no arguments and returns no data.

create Function A public function that will create all of the fields in the database for a Monster instance. Takes a Monster instance as an argument.

update Function A public function to update an entry for a Monster or a user in the database, takes a Monster instance or a User instance and returns no data.

search (monster) Function A public function for searching the database to find a list of a user's Monster instances. Takes a User instance and returns an array of Monster instances.

search (user) Function A public function for finding a user in the database. This function takes nothing and returns a User instance and returns nothing.

remove(monster) Function A public function to remove the representations of the Monster instances from the database. Takes a Monster instance and returns nothing.

remove (user) Function A public function to remove the representations of the User instances from the database. Takes a User instance and returns nothing.

isReal Function A public function to check if the user is stored in the database. Takes two Strings, a username and a password. Returns a boolean.

shutdown Function A public function for closing a connection to the database. This function requires nothing and returns nothing.

DropTable Function This function destroys the table i.e. clears it, method is there but is deprecated.

searchGraveYard This method collects dead monsters, this method however is not integrated in the final program.

6.1.2 User Class

A public class to represent a user, it does not inherit from any classes. There are no public variables because everything is dealt with by the PersistManager class.

ChangeBattleMonster Function A public function to change which a user's Monster is set for use in a battle situation. Takes a Monster instance to set as the users new battle Monster.

6.1.3 UserFactory Class

A public class designed to create new User instances when called from somewhere. It has one public function to create the User.

6.1.4 Requests Class

A public class that contains all request attributes to be persistent into the database for request handling.

make it Function This public function will create a User instance, that will later be stored by the PersistManager. Likely called by the Login Class from the Controller package. It requires the following arguments:

- name: String - The name of the new user.
- password: String - The password of the new user.
- vMoney: int - The amount of money the new user has.
- type: Type - The user's type, ie. regular user or superuser.

This function will return a User instance.

6.1.5 Monster Class

The Monster class only has public getters and setters, it represents each monster and contains everything that is necessary to represent a monster. The PersistManager uses the Monster class when it converts each monster into a database friendly form.

6.1.6 MonsterFactory Class

A public class to create Monster instances that will later be stored by the PersistManager as part of a User instance. It has one public function.

make it Function This public function will create a Monster instance. It may be called from the UserFactory class when a new user is created to make their first Monster. It might also be called from anywhere else that needs to create a new Monster instance, such as the Breed (servlet) class. It needs to be called with the following arguments:

- name: String - The name of the new monster.
- height: int - The new monster's height.
- age: int - The new monster's age (in days I expect).
- strength: int - The new monster's strength.
- health: int - The new monster's health.
- sex: Boolean - The new monster's sex as an isMale boolean.
- aggression: int - The aggression of the new monster.
- dob: int - The new monster's Date of Birth.
- battleMonster: Boolean - States if the new monster is the monster used for battle.

This function returns a Monster instance.

6.2 Controller Package

The Controller package contains all the classes that deal with the logic behind the back end of the application. It contains all the servlets and uses the Model to represent data for it.

6.2.1 CreateMonsterAndUser Class

A public class used by the Login (servlet) Class to handle creating a user and the user's first monster. It will rely on the MonsterFactory and UserFactory in the Model. There are two public functions.

CreateMonsterAndUser (Constructor) Function This function will set up the class, by creating a User and with it, its first Monster. It takes no arguments and returns no data.

CreateMonster Function A public function to create a Monster instance, it takes no arguments and returns a Monster instance. Presumably used by the constructor.

6.2.2 Login (Servlet) Class

The public class login handles the authentication of the users. It is a servlet class and is used over http. There are a number of public functions to deal with users.

checkPass Function A public function for checking a password is correct and valid. Takes a String and returns a Boolean value that reflects whether the password is correct.

checkUserName Function A public function for checking if a username is correct. It takes a string and returns a boolean value that reflects if the username is correct.

register Function A public function to register a new user into the system. It needs to be passed by two variables; username and password both as strings. It returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.3 Shop (Servlet) Class

This servlet class handles all of the information required to deal with the shop, it allows users to buy and sell Monsters and to find out prices for other listed items.

calcWorth Function A public function that calculates the value of a monster so that it can be displayed in the Shop. Uses a local variable of Monsters in the Shop to find the monsters and returns an integer that represents the price.

buy Function A public function that allows the user to buy the selected monster from the shop if they can afford it. It takes no arguments and returns nothing.

sell Function Public function that allows the user to sell the selected monster from the user's monster list for the calculated worth of the monster. It takes no arguments and returns nothing.

viewVMoney Function A public function for finding out how much money the user has, because this is needed for display in the shop. There are no arguments and nothing is returned.

addMonster Function A public function that adds a monster to the shop, it takes no arguments and returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.4 FriendsList (Servlet) Class

A public class to deal with the Friends List and all the information required, it also allows users to add, view, and remove from their friends list.

addFriend Function Public function that allows the user to send a friend request. If request is accepted then the friend ID will be added to friends array.

removeFriend Function Public function that asks the user to confirm the removal of the specified friend ID, if confirmed, the specified friend ID will be removed from the friend array.

requestFight Function Public function that allows the user to pick a friend from the friend list and to start a battle against the friend's preset battle monster.

getFriend Function A public function that allows the user to pull out a friends profile. Takes no arguments and returns nothing.

findUserMonsters Function A public function to find all of another user's monsters. This function takes no arguments and returns nothing.

updateList Function A public function to update a user's friend list from the server. Takes no arguments and returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.5 Breed (Servlet) Class

A public (servlet) class to handle how you might breed monsters in the game. It allows you to perform all relevant checks to the data. There are a few public functions made available. There are no public variables. The only important things are the two parent Monsters, that are passed in as parameters to the createChild function.

calcCost Function This is a public function to work out what it would cost to buy a particular monster, takes a Monster instance and returns an int that is the price to buy the given monster.

createChild Function A public function that allows the user to breed a monster, takes two Monster instances and returns a Monster instance that is a child of the two given monsters.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.6 SimulateFight (Servlet) Class

A public (servlet) class that represents two monsters battling in the application. There is only one public function available and no public variables.

calcWinner Function A public function to work out which Monster is going to perform better in a battle. Takes two Monster instances, returns the winning Monster instance.

isPrized Function A public function to work out if a match is prized, takes nothing as an argument, but returns a boolean.

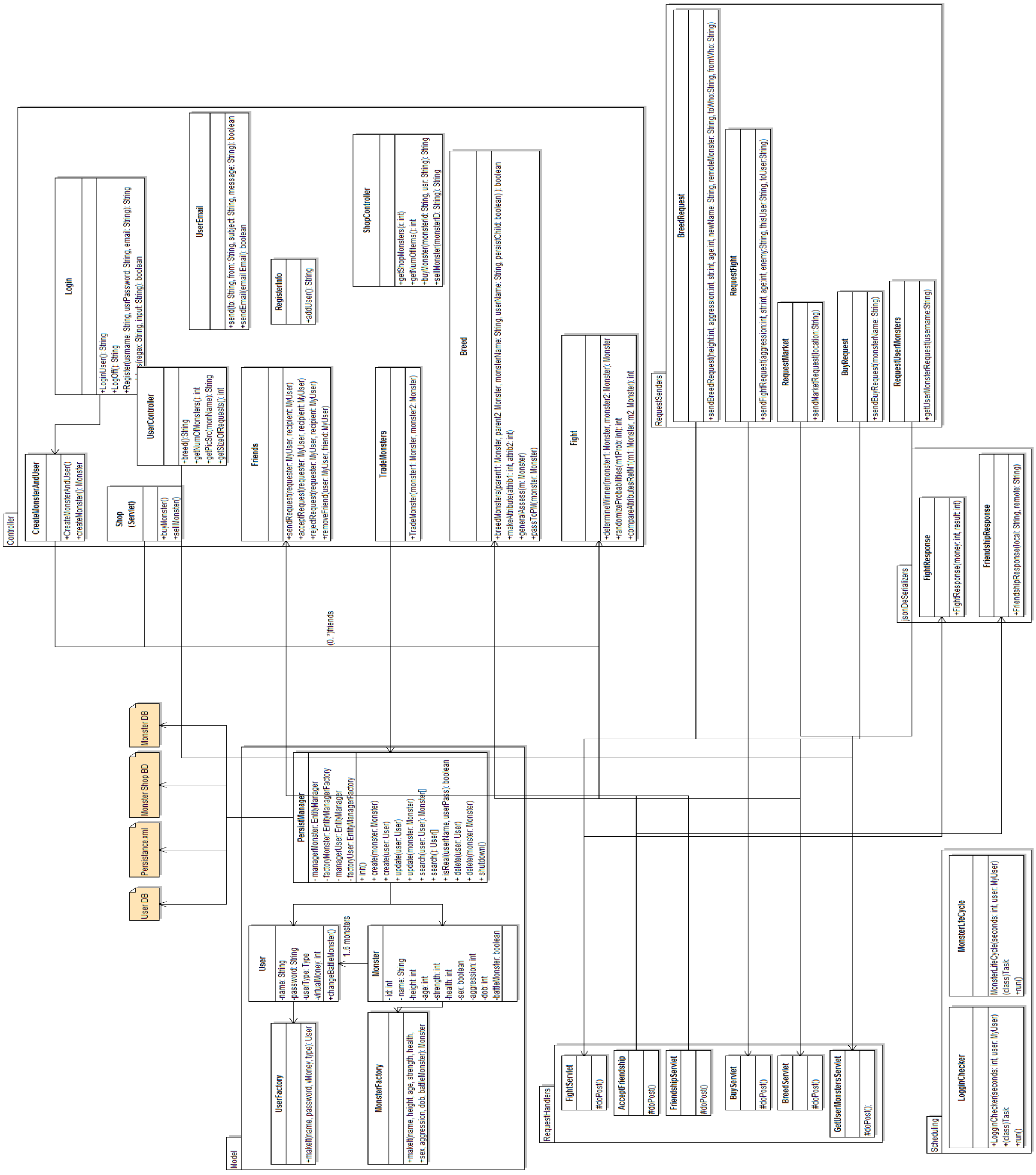
prizeAmount Function A public function to return the amount that is given as a prize, takes nothing as an argument and returns an int that represents the prize money.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

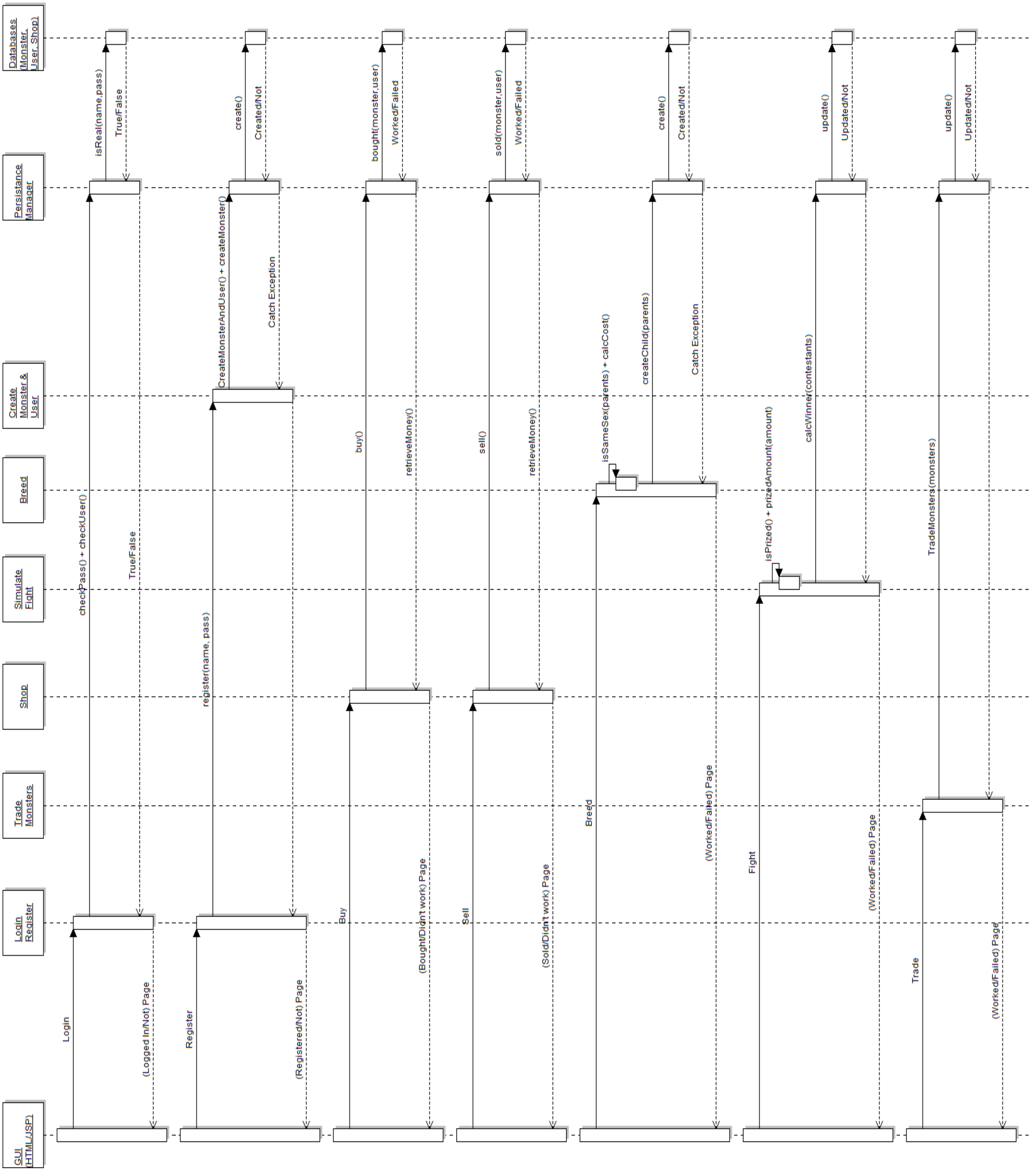
Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

7 Detailed Design

7.1 Monster Mash Class Diagram


Controller
CreateMonsterAndUser
+CreateMonsterAndUser()
+createMonster(): Monster
Login
+LoginUser(): String
+LogOff(): String
+Register(usrname: String, usrPassword: String, email: String): String
+s(regex: String, input: String): boolean
UserController
+breed():String
+getNumOfMonsters(): int
+getPicSrc(monName): String
+getSizeOfRequests(): int
Shop (Servlet)
+buyMonster()
+sellMonster()
UserEmail
+send(to: String, from: String, subject: String, message: String): boolean
+sendEmail(email Email): boolean
RegisterInfo
+addUser(): String
Friends
+sendRequest(requester: MyUser, recipient: MyUser)
+acceptRequest(requester: MyUser, recipient: MyUser)
+rejectRequest(requester: MyUser, recipient: MyUser)
+removeFriend(user: MyUser, friend: MyUser)
ShopController
+getShopMonsters(x: int)
+getNumOfItems(): int
+buyMonster(monsterId: String, usr: String): String
+sellMonster(monsterID: String): String
TradeMonsters
+TradeMonster(monster1: Monster, monster2: Monster)
Breed
+breedMonsters(parent1: Monster, parent2: Monster, monsterName: String, userName: String, persistChild: boolean): boolean
+makeAttribute(attrib1: int, attrib2: int)
+generalAssess(m: Monster)
+passToPM(monster: Monster)
Fight
+determineWinner(monster1: Monster, monster2: Monster): Monster
+randomizeProbabilities(m1Prob: int): int
+compareAttributesRetM1(m1: Monster, m2: Monster): int
(0..*)friends
RequestSenders
BreedRequest
+sendBreedRequest(height:int, aggression:int, str:int, age:int, newName: String, remoteMonster: String, toWho:String, fromWho: String)
RequestFight
+sendFightRequest(aggression:int, str:int, age:int, enemy:String, thisUser:String, toUser:String)
RequestMarket
+sendMarketRequest(location:String)
BuyRequest
+sendBuyRequest(monsterName: String)
RequestUserMonsters
+getUserMonsterRequest(username:String)
jsonDeSerializers
FightResponse
+FightResponse(money: int, result: int)
FriendshipResponse
+FriendshipResponse(local: String, remote: String)
Monster DB
Monster Shop BD
Persistance.xml
User DB
Model
PersistManager
- managerMonster: EntityManager
- factoryMonster: EntityManagerFactory
- managerUser: EntityManager
- factoryUser: EntityManagerFactory
+ init()
+ create(monster: Monster)
+ create(user: User)
+ update(user: User)
+ update(monster: Monster)
+ search(user: User): Monster[]
+ search(): User[]
+ isReal(userName, userPass): boolean
+ delete(user: User)
+ delete(monster: Monster)
+ shutdown()
User
-name: String
-password: String
-userType: Type
-virtualMoney: int
+changeBattleMonster()
1..6 monsters
Monster
-id: int
-name: String
-height: int
-age: int
-strength: int
-health: int
-sex: boolean
-aggression: int
-dob: int
-battleMonster: boolean
UserFactory
+makeIt(name, password, vMoney, type): User
MonsterFactory
+makeIt(name, height, age, strength, health,
+sex, aggression, dob, battleMonster): Monster
RequestHandlers
FightServlet
#doPost()
AcceptFriendship
#doPost()
FriendshipServlet
#doPost()
BuyServlet
#doPost()
BreedServlet
#doPost()
GetUserMonstersServlet
#doPost();
Scheduling
LogginChecker
+LogginChecker(seconds: int, user: MyUser)
+(class)Task
+run()
MonsterLifeCycle
MonsterLifeCycle(seconds: int, user: MyUser)
(class)Task
+run()

7.2 Monster Mash Sequence Diagram



8 References

1] Software Engineering Group Project Plan Design Specification Standards. C. J. Price, N. W. Hardy. SE.QA.05a. 1.6
2] Project Plan - CS221 Project Group 6. Version 1.0
3] Test specification - CS221 Project Group 6. Version 1.0
4] Software Engineering Group Projects General Documentation Standards. C. J. Price, N. W. Hardy. SE.QA.03. Version 1.5
5] Software Engineering Group Projects Monster Mash Game Requirements Specification. B. P . Tiddeman. SE.CS.RS. Version 1.2

9 Document History

Table 1: Document History

Version	CCF No.	Date	Changes made to document	Changed by:
1.0	N/A	06/12/2012	Created Design Specification	sam39
1.1	N/A	12/02/2013	Made required changes	sam39
1.2	N/A	13/02/2013	Few sentence changes on some functions	sam39

Group 06 -Final Report

12th February 2013

Amy Rebecca James, James Slater, Dan Mcguckin, Samuel Mills, Felix Farquharson,
Ben Brooks, Christopher Krzysztof Ilkow, Aiman Arafat
Project Coordinator: Nigel Hardy

Release: 1.3
Status: Release

Computer Science Department, University of Wales, Aberystwyth

Contents

1 Introduction

1.1 Purpose of the Document

The main purpose of this document is to identify and describe the information in the final report document produced by CS22120 Group 06. Each section of the final report will be individually reviewed and described in detail and it will contain a conclusion on how our project leader and team members thought the overall project went as a whole, this will look at both the positive and negative parts of the project.

1.2 Scope

The final report will cover a variety of areas of our project such as the end-of-project report, the project test report and the project maintenance manual. All these documents will draw a conclusion and end to the Monster Mash project.

1.3 Objectives

The main objective of this document is to conclude and review the project as a whole. The reason for this is to see where the group performed well, what areas we could improve on and most importantly to see what we have learnt as a group in terms of team work and strategy.

End-of-Project Report

2 Management Summary

The project achieved a working user interface and it was very easy to function as a whole. Our buttons and links were all clear and understandable which allowed the user to navigate around the website effectively. The web application was very easy to navigate with the site ID header, a bar along to the top with all the options available to the user, a friends list at the bottom right hand side of the screen and any interactive buttons at the center / bottom of the screen for fighting or breeding etc. The issues we had with the user interface were that when a user logged in using their username and password, it would take the user to the home page and the options that should be available to the user once logged in were not and the page had to be refreshed first to show these options. The other issue was that when a user was logged in, each page required at least one monster to function and if the user did not have a monster a null pointer was presented. We did have a Boolean value to set any monster in the position of 0 to be invincible to stop the monster from aging and thus solving the problem but for some unknown reason it stopped working in the final version of the application. Money when buying or selling monsters would be updated in the database but would not update visually to the user, this was because it took the value from the database when first loaded in the side bar. If the user was to logout and then log back in, the money would be updated. Given more time this error would have been solved.

For the back end of the project we had everything in place for the algorithms E.g. Breeding, fighting, trading and all the persistent class to work with the database. The only issue the group had was with the database as the database we actually had tested did not work in the web application. This resulted in the group creating a new one in the integration and testing week and this caused the group issues with sending requests and creating friends and thus stopped the user from being able to fight with friends and breed etc. So towards the end the group had to set it up internally to fight/breed/buy etc your own monsters to allow us to show its functionality. For the log in feature we set up regular expressions for the username and password. We had an expression for the email feature which would send the user an email welcoming them to the game. However unfortunately the regular expression for this did function as intended and neither did the sending of the email so the group had to comment it out. Given a bit more time we would had fixed this issue most definitely. The Friends List works as long as the friends are in the database but we could not get the request actions to work before the software delivery date and this applies for all the request actions also. In regards to server to server communication, be cause we had delays with the database, however we implemented as much as we could into our code. All the group needed was to set up another computer and have their IP address and it would send a request with all the information needed successfully. However we did not have time to set it up with another group but the code is in and place ready to use.

The documentation has been checked over by Amy James and Samuel Mills and is all at a very consistent standard, using LaTeX. The only document we experienced some problems with was the Maintenance Manual and this was due to a lack of communication and lack of time allocation to properly check over the document before we had to hand it in.

A major difficulty we had was at the start of the integration and testing week. On the Tuesday we got told that one of our group members Aiman had dropped out and this ultimately resulted in us having a pair of hands less. This meant we had to share out the jobs Aiman had to do to other members but everyone was very willing to help and we shared the jobs out evenly. However this still put extra strain on the group as they had to do all their own jobs as well the extra tasks on top. Through out the project it was very hard to get in contact with certain members of the group. The group attempted to resolve this issue by setting up a group on Facebook to allow group members to communicate and ask questions. This was also in place for the project leader to delegate tasks to members. However the problem was still present as some members did not look at it or just ignored the comments and posts. This delayed certain work from being developed and we had issues with this from the outset and throughout of the entire project. During integration and testing week we had difficulties combining the groups code together as each segment of the codes logic was different. We overcame this by pair programming and the group made controller classes that dealt with all queries the user could have. The group found this very effective and it worked really well and saved a lot of time.

Overall the team worked very well together in terms of commitment, communication and willingness to do the tasks and complete them to the best of their ability. Commitment on the whole from everybody was very good having majority of members at every meeting and completing the work and keeping to submission deadlines. Communication in the group at times was lacking at time however the group always managed in the end to make sure work was complete. Everybody voiced their well in group meetings with ideas and general discussions, no one argued about differences in opinions or ideas confidently and the group all worked together to complete the project. One of the main reasons the team did so well was the willingness to help each other and willingness to offer to put in hard work to achieve success.

3 Historical Account of the Project

The first milestone within the group project was to assign different roles to the relevant group members. This was discussed in the first meeting we attended. Three members of the team put themselves forward to take on the role of Project Leader:

- Dan McGuckin
- Chris Krzysztof Ilkow
- James Slater

As other members were unsure who would best fit the role we decided that each person should prepare a pitch to deliver to the group to aid their decision on who would suit the group leader role best.
QA Manager was assigned to Ben Brooks and deputy QA Manager to Amy Rebecca James.

3.1 Project Plan

The first document that the group needed to produce was the Project Plan. The aim of the Project Plan was to outline the plan for the entire project. To meet the deadline for this, each member of the group was assigned a section of documentation to complete. We decided that each part needed to be finished a week before the deadline so that we could meet as a group and conduct a review, this allowed all members to see each section and raise any worries or concerns with certain parts of the document. During this review Ben Brooks was also able to compare all sections against the relevant QA document to ensure the group were completing documents correctly.

3.2 Test Specification

When producing the test document the group decided it would be most beneficial if we split into two groups. James Slater, Dan McGuckin and Chris Krzysztof Ilkow would be in charge of unit testing using Junit. Then Samuel Mills, Ben Brooks, Amy Rebecca James and Felix Farquharson were to produce test tables. Each person was in charge of creating test tables for a particular feature of the Monster Mash application e.g. - login/register, Monster Shop etc.
Once again we ensured that the test tables were completed well in advance so that all group members had a chance to review them and add suggestions or remove irrelevant tests.

3.3 Design Specification

Once again when completing the Design Specification we discussed as a group which members felt most comfortable completing particular sections of the document. Some group members worked together as some weren't 100% sure what that section required and troubleshooted the task as a team.
Again, a review of the completed document was put forward and any required changes were made well before the hand in date was due.

3.4 Prototype Demonstration

Prior to this demonstration we discussed as a group what it was we would like to show of our Monster Mash application so far.
We decided that we would produce a GUI so that we could demonstrate how the different pages would interact with each other. Our project manager Nigel would also be able to see the general look and feel of the application in regards to colour and images etc.
Dan McGuckin talked Nigel through some of the algorithms that he had already written also so that he understood how we intended to do carry out different features of the Monster Mash application.

3.5 Coding and Integration Week

In the week leading up to coding and integration week we had a group meeting to discuss what needed to be done and to ensure that all group members understood what tasks needed to be completed. This ensured that minimal time was lost and we made the most of our week of development.
During this week each group member was in charge of a specific area of development:

- Chris Krzysztof Ilkow was in charge of the server server interaction.
- Ben Brooks and Felix Farquharson designed and created the JSP pages.
- Dan Mcguckin concentrated on programming the database and the back end of the application.
- James Slater integrated the database and back end with the GUI.
- Sam Mills and Amy Rebecca James continued with the documentation. This included making relevant changes to existing documentation and beginning the final documentation alongside extensive testing.

3.6 Final Documentation

To complete our final documentation, once again each group member was given an individual task to complete. The final report was to include personal reflections from all members of the group and also a review of how each group member worked during the project written by the group leader (James Slater). Relevant changes to previous documentation also needed to completed ready for final hand in. A review meeting was held a week before the deadline so that any relevant changes could be made and the documentation could be bound in time to be submitted. The roles were assigned as follows:

James Slater was to write the Management Summary, critical evaluation and the reviews of each group member. He also had small changes to make to previous work so that other documents could be edited. Ben Brooks was designated project management and was to delegate roles to Dan Mcguckin, Chris Krzysztof Ilkow and Felix Felix Farquharson. All small changes had to be made to previous documentation. Amy James was in charge of writing the project time line, creating a template for the personal reflections and adding in the changes to previous documents together with Samuel Mills. This includes the project plan, design specification and test report. It was also their responsibilities to put all the documentation together into one .pdf document.

4 Final State of Project

4.0.1 Server Authentication

Users can log-in or register their details and they as stored in a database (Derby) and are retrieved for log-in. If a user is not in the database and tries to login, the user is given an error and instructed to register.

4.0.2 Friends List

The friends list did work however somewhat sporadically. Friends would appear in a Friends List. Requests for fighting or breeding could be sent to friends successfully. When the user's friend logs in they can see requests and confirm or decline them. (This action itself was not implemented). Friends are added by name but not email. Friend name is a primary key in the database so each name is unique.

4.0.3 Server Monster List

The server successfully keeps a list of monsters in a table in the database. Each Monster can have an owner (A foreign key of username). If a monster has an owner then that Monster is part of that users list. If the user doesn't own the monster, it is part of the Shop. Attributes for Monster include Age, Height, Strength, Aggression, worth, their maximum age and whether they are dead and if they are immortal. (Not affected by age and not killed in fights).

The server handles the Monster lifecycle by using a thread that activates when a user logs in and cancels when they log out. This means that monsters will only age when a user is logged in and the account is active. When the user is logged out the Monster will not age and this was a design decision made by the group. There is another thread that logs a user out if a user has not made an action for a certain period of time (E.g. Breeding a Monster). This was primarily built to stop monsters aging when a user closes the application without pressing log out button. There can be a memory issue problem with the threads in that they can multiply up and cause the application to crash (This has only happened twice).

Monsters die if they lose a battle with another monster and the server removes the defeated monster from the database. The group did want them to be put into a graveyard so they could still be seen and potentially revived but we did not have time to implement this feature. New users are allocated a Monster; however the Monster is not random and is always the same. All child monsters created through breeding are random however with boundaries determined by the parents statistics.

4.0.4 Monster Fights

Cash prize feature is not implemented. Users can challenge friends but cannot fight friends. Fighting does however work and the winner is based on Monster statistics with a bit of randomness combined in also. A losing Monster is classified as dead.

4.0.5 Server-Server Communication

Server-server does work; the ode is written but has not been fully integrated with the GUI so users cannot use it properly. The application can communicate with hard-coded servers on other computers. The application can only communicate with servers running our selected software and it cannot communicate with other groups as we did not have time to implement this properly as other groups did not follow standards that specifically (That includes our group also).

4.0.6 Client Options

Our client has issues but many parts work and some do not.
Users can register an account but can't unregister an account. They can successfully add friends but cannot remove them (The required code is there but not integrated). Users can offer Monsters for sale and can buy new Monsters. Users do not set prices for Monsters but the Monsters worth is calculated by an algorithm that takes into account their stats and their age (i.e. peak age = more expensive). User pricing is implemented in code but not fully integrated into the application. V-Money will not be shown subtracted in the GUI/JSPs, however it does work in the database. Users have a minimum bank amount of 100 V-Money. The breeding feature works but not with other users. Breeding is calculated using an algorithm that takes the statistics of the parents, finds the middle ground between them both and then chooses stats randomly within a boundary based on this middle ground.
Friends can't successfully view other friends Monsters or breed with them. Again, the code has been developed but is not integrated into the application, therefore users can only breed with themselves. The Shop feature works and any user can see Monsters in the Shop that other users have sold and can purchase them. Purchased Monsters are transferred to the user and taken from the Shop and sold Monsters are transferred from the user Farm to the Shop. Shop Monsters are Monsters with no owner so selling or buying a Monster is a case of taking or adding a username (owner) to the Monster. If a client does not have sufficient funds, the transaction will still take place however. This is not a design decision and the group did not intend for this to happen. However due to time constraints the group did not fix this issue.

4.0.7 Startup in Browser

This works successfully and users can login and register (which creates a new account). Users can then log-out once they are logged in. This takes them back to the log-in screen.

4.0.8 Game Display

When logged in users can switch to the Farm feature which shows them their Monster list with Monster statistics and allows them to breed Monsters. Friends can displayed on all pages in a friends list that expands at the bottom right of the selected page. Friends names are displayed but Monsters and general options are not available. Monster fights are handled on another page which like the other pages can be accessed through the top navigation bar which is clearly shown to the user. Monsters chosen fight can be selected using radio buttons and the winner is then revealed. The loser's Monster is killed and removed from the users monster list.

4.0.9 Friend Matching

Friend requests can be sent to other users which uses their name as identification. Other users can accept and reject friend requests. The user who sends the friend request is not told the outcome but if the invitation is successful, the friend will appear in the user's friends list.

Fight notifications This has issues in regards to prize V-Money.

The Monsters list is updated when a fight ends and the losing monster is removed from the user's Farm. Prize V-Money code has been developed but is not integrated with the application so it can't be used. The winner is calculated through fight algorithms which takes both battling Monster's statistics and calculates percentage likelihood of winning/losing for one monster and a random wheel then selects a Monster taking percentage into account.

4.0.10 Friends Rich List

Friends list can be viewed by a user successfully and friends can be seen. However they are not ordered in wealth and wealth cannot be seen for each friend.

4.0.11 Appearance

The appearance of the final application utilises the Twitter Bootstrap CSS framework. This lets the user have a consistent experience throughout the application and allows for standardisation of certain elements such as buttons, menus and modal dialogs. As well as the CSS framework, jQuery was also used to animate things such as friends lists and provide additional functionality to elements on the page. These frameworks bring together the design which lets the user have a great and familiar experience.

4.0.12 Server Response

The system application can be quite slow at times and this is likely because of database connection code that was not optimized as well as it could have been.

4.0.13 Target Computer for System

The software is tested to run on Firefox, IE and Google Chrome. There are sometimes database problems that cause the user to not be able to log in or register but the group have tried to correct most of these issues.

4.0.14 Use of Java Correct

We used Java with Glassfish to build the project. This includes setting the database up where we used the Java Persistence API.

4.0.15 Reuse of Existing Software

In this project, we made good use of existing software and this both saved time and effort. In order to be able to do this, the existing software had to be open source or freely available to the public.

For example, Glassfish is an open source application server written in Java by Oracle. This enables us to easily serve web pages via JSPs and servlets to create a smooth experience for the end user.

Hibernate was used to create a persistence manager in order to connect to the database. This saves managing the connections manually and it is easy to implement. It is open sourced under the GNU Lesser public license.

JSON is used to collect data in a structured manner, similar to XML. We used GSON in order to implement it into our project. It's an open source library to serialise and deserialise JSON objects written by Google.

The user interface made use of the Twitter Bootstrap framework. This provides pre-made CSS styles and JavaScript snippets which allow for fast deployment of web pages that look visually appealing. It is open sourced under the Apache License 2.0.

Finally the images used in the user interface were all either open source or in the public domain. Because of this, there would be no problem in releasing this as a product, or an open source project.

5 Performance of each Team Member

5.1 Amy James

An excellent team member and it was a pleasure working with her over the course of this module. Amy was set as QA deputy manager and she made sure all the documentation were up to a high standard before they were submitted for a hand in. Amy attended every meeting if she was available and if she was unavailable to make the meeting either with Nigel or a group meeting, Amy would always inform everyone either by email or on the Facebook page. Amy's attendance for the meetings that were setup outside of University hours was very good and only missed a couple of meetings because of commitments in other modules and always the group if this was the case.

In integration and testing week Amy was always on time and was there each day of the week and worked 9am-5pm. Semester 1 - In the meetings we had every week with Nigel, Amy was very vocal with good comments, ideas and always had a very positive attitude. Amy from the start stated she did not want to do any programming and would rather focus on the documentation and testing side of the project. When any documentation needed creation, Amy was the first to say she would do it alongside Samuel Mills.

Semester 2 - Amy continued at the same performance standard and was able to notify the group of the errors found when carrying out testing with Samuel Mills back to the programmers to allow them to make the suitable changes. Amy was also very good at keeping everybody up to date with what work she had done and what work she planned on doing. This made the general project process a lot easier as team leader.

Tasks: Amy was very helpful with every aspect of the project from testing to documentation creation and editing. If Amy finished one task she would take the initiative and find another task or work to do which again hugely benefited the group. Amy was always productive and her work was at a very high standard.

5.2 Samuel Mills

An excellent team member who was a really hard and enthusiastic worked who contributed ideas to the project. I really enjoyed working with Samuel over this project as he performed well in all his tasks. Samuel stated from the start that he did not want to program and wanted to focus on the documentation and testing aspects of the project. Samuel was always at every meeting and volunteered to do the minuets for each meeting which helped the group keep organised. Samuels attendance of weekly meetings with Nigel or unofficial meetings was 100% In the integration and testing week Sam was working with the team from Monday to Friday 9AM-5PM.

In the weekly meetings with Nigel, Samuel always came with a good attitude and contributed by making some very good points and ideas to the project. As Samuel was writing the minutes for each meeting, he word processed and emailed them to the group and put them on Github within 24 hours of the prior meeting. For the meetings we had outside of University hours, Samuel was always giving positive feedback and he would check and make sure that work was correct and up to a good standard. He also had good communication and written skills in general.

Samuel was very helpful for doing all the minutes, creating documentation, combining documents and testing. Samuel was also working with Amy Rebecca James in the testing in the integration and coding week. Samuel also helped with the javadoc in the Monster Mash application and once Samuel had completed a task he would inform me straight away and be willing to do more work.

Talking with Samuel about what I have written about his performance, Samuel added some more tasks that he carried out and agreed with my description of his performance. We are both happy with the project and the contribution of work that Samuel did. Overall Samuel was an important team member and contributed well to the project and I would happily work with Samuel again in the future.

5.3 Dan McGuckin

Dan Mcguckin was very hard working and he completed what ever tasks I asked of him and it would be complete by the next time of seeing him. I didn't have to keep checking or asking for updates, Dan just did the work at a high standard. Dan was one of the main programmers of the group who focuses on the database and the algorithms for this project.

Dan was at all the weekly meetings with Nigel and only missed a couple of meetings that were setup. Dan informed the group before hand if he was unable any of the meetings. In integration and coding week Dan worked from 10AM-6PM Monday to Friday and also did segments of work back at home.

Dan was tasked with creating the UML design, database and the algorithms. Dan was tasked with the database of our project and he had created and tested a HyperSQL database and showed it to me fully functioning. However when the group tried to implement it with the JSPs, the database would not work in the web application environment so the group had to create a new database source. Myself and Dan were able to use the built-in database inside NetBeans and still use all of his existing persistent classes. The errorwas unforeseen and should have been tested in a web application beforehand and because of this the jUnit tests created did not work correctly through the web application environment. Dan was only able to create tests for methods that didnt interact with the database but helped Amy and Samuel create some more tests for the testing to ensure the code was handling everything correctly. Dan also created the algorithms for Breeding, Fighting and had already developed the Buying and Selling algorithms completed prior to implementation and testing week starting.

Talking with Dan about what I have written, Dan was happy with what I had put and was happy with the tasks discussed and how way they went in general. Overall Dan was a big part of the programming side of the project. He successfully made the algorithms for the breeding, fighting and creating the persistent classes. I feel that if we did not have as many setbacks with the database which was an integral part of the project we would have achieved much more development in general. itself.

5.4 Christopher Krzysztof Ilkow

A really hard working team member and was one of our main programmers. Chris was tasked with the server to server communications.

Chris missed a few of the weekly meetings with Nigel and was also absent from some of the other meetings that were setup outside University hours. Chris did not always inform the group that he was not able to attend a meeting however with that said he was did complete the tasks that were delegated to him to a good standard. In the integration and testing week Chriss attendance was very good working 9/10PM - 6PM throughout the week and helped out later one night to get certain aspects of the application completed.

When Chris was in the group he was very vocal and made very good suggestions towards the planning and creation of the project. He made it very clear that he was very good programming in general and wanted to mainly focus on that aspect. Chris suggested some ideas for theGUI and server server communications.

Chris was delegated the task of setting up the server server connections and had it all working and demonstrated this to me. Unfortunately by the time we had implemented Chriss work there were no other groups available to test it with. However we made sure that the code was functioning and was ready to be used if we had the opportunity to implement it. Chris also contributed to parts of the documentation that was on time to a good standard.

Talking with Chris about what I have written, Chris was happy with the way I described his performance and was happy with the tasks and the way they went overall. Overall Chris did a major part of the project and helped with many different aspects of the application. Unfortunately by the time we had sorted out the database issues and implemented his code we did not have time to find another group to test it with.

5.5 Ben Brooks

An excellent team member who was the QA Manager for the group. Ben was in charge of the GUI, from design to creation along side Felix and Aiman.

Ben was at all the weekly meetings with Nigel and did not miss any meetings that were setup unofficially. In integration and coding week Ben worked from 9PM-6PM Monday to Friday and also helped out later one night to get certain aspects of the application complete.

Ben always said what he could and could not develop and was more than happy to meet at any time to discuss project progression. In all meetings Ben was always very vocal and made some very good suggestions, ideas and points about the GUI and our version control platform. Ben was also more than happy to be in charge of the GUI and delegate certain tasks to particular individuals. Ben also made contributions to documentation alongside Amy and Samuel. As QA manager Ben also did a very good job at making sure the source code was at a good standard and worked well with Amy to make sure the documents were at a high standard.

I was very impressed with Ben with regards to how well the GUI looked and how easy it was to be able to change and manipulate pages to make them look as the group intended. Ben was also the person assisted members of the group who needed help with version control and Github.

Talking with Ben about what I have written, Ben was pleased with how I described his performance and was happy with the tasks and the way they went. Ben also added some more task/roles he carried out. Overall Ben played a major part in the project and helped with all aspects of the project from documentation to the GUI and I would happily work with Ben again.

5.6 Felix Farquharson

Worked along side Aiman in the first semester and Ben throughout the project on the Documentation and GUI design and creation of it.

With regards to the weekly meetings with Nigel, Felix was there more times than not and if he was not able to make the meeting he did not always notify the group. This also applies for the meetings we set up outside University hours. This is mainly because the groups main gateway of communication was through Facebook and Felix did not check his Facebook frequently enough to keep up with updates. I did start texting Felix some information about what was needed but there was no consistency in his replies.

In integration and testing week Felix missed Monday and Friday and worked 12PM-6PM for the other three days.

However he stayed behind one night for a while to help to get certain aspects of the application completed and also worked the next Monday night to make up for the fact he missed Monday of integration and testing week.

Felix when with us was very vocal and made very good suggestions towards the planning and creation of the project. Felix helped with sorting out Github and the CSS used for JSPs templates. Felix also worked on the login checks for the correct username and password and the feature to send an email to the user welcoming and informing them of the registration.

5.7 Aiman Arafat

Dropped Out.

5.8 James Slater

I was Group leader and was one of the programmers for the group. My task was to keep the group focused and ensure progress was present from the outset and throughout. I also had to delegate tasks to group members and had to integrate the GUI with the Database and assist Chris integrate his code into the application also.

I was present at all meetings with Nigel and as I setup all the meetings for the group. I made sure everybody had a task to complete and that everybody was happy and felt they all had the same amount of work in comparison to other group members. I had to help out with the creation of the GUI as Aiman departed from the group and I was in charge of linking the Database and algorithms with the GUI by making controllers for the different aspects of the application (E.g. Fights controller, Users controller and a Shop controller. Overall I feel the team worked very well together from start to finish and everyone was willing to help each other. The group had set backs and we worked through them together by helping each other with each problem.

6 Critical evaluation of the team and the project

The team on the whole worked very well together from communication to attendance. The group worked well when working on the documentation and also pair programming in the integration and testing week.

The attendance and communication of the team was very good in regards to the weekly meetings with Nigel. I feel that we gained something from each meeting and they generally went very well and many points were always made by all group members. I believe the group always left those meetings with ideas and plans for the future. The team in the weekly meeting worked very well and had many points and ideas from each individual, this made writing the documentation, assigning tasks to individuals and designing the project easier and a simpler process. There was not a single member of the group who made it hard for the group to progress and if we ever had a conflict of ideas we managed resolve the issue by discussing it in a very professional and concise manner. I do not believe the team could improve on how we worked in the meetings as attendance was generally good and all the group communicated well with each other.

The Group in the integration and testing week worked fantastically well with each other and individually. Each person worked on the task delegated to them and if any help was required all group members were willing to help.

Overall I think the team worked extremely hard and well together. I believe we could have improved on certain aspects of attendance where all team members turn up instead of having one or two members missing. This would have helped with setting tasks earlier on in the process so we are all working together and in synchronization. The main recurring problem we had with the project was our database. The database we had originally was tried and tested but we could not get it fully working in the web application environment. We had to abandon the original database and create a new one that would work with the web application, this set the project back a day and resulted in the group being on the back foot. We could have prevented this by making sure the original database worked in the web application by carrying out more spike work and being a bit more diligent with testing the database. The group could get it successfully working on a computer then once it was synced with Github it would no longer work. We found the problem here was with our persistent.xml file which would change depending on the computer. We solved this eventually in the week however we should of had this set up before the integration and testing week but we did not foresee the problem and this held us back again.

The greatest lesson learnt from this project was to communicate between all group members and to make sure that all members were working from the same page. To be able to gets everybody's opinion and identify suitable times for meetings was rather difficult as not everyone would their emails or Facebook notifications. It was also the same when it came to handing out tasks as each person would work on the tasks at different times and need questions answered. However this resulted in members having to wait for answers and this made communication for some tasks very hard and this result in slowing down project progress. The other lesson learnt was to make sure all group members we doing the job and not other tasks. (E.g. Setting a task to make a function work on the project and a member creates a function that does even more than it should, which is positive but when other tasks still need to be completed, the extra work is not necessary until later date. By the end of project we had run out of time for certain functionally requirements that were in the design specification.

7 Project Test Report

The test tables below are the tests that currently Pass/Fail in regards to Login/Logout and Account creation.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-001	FR7	Creating User Account	The user will enter desired username and password.	The details should be saved and added to the database.	The user can use these details to access the Monster Mash application.	Pass	N/A
SE-N06-002	FR7	Creating an account with characters other than A-Za-z0-9 or an underscore in the username.	User should create a username containing an exclamation mark for example.	An error message should appear saying that the username cannot be created and explaining why.	The username will only be created if it contains accepted characters.	Pass	Failed during first tests as there was no validation.
SE-N06-003	FR7	Testing that the account has been created successfully	The user will enter their Login details and click Login button.	User should be taken to the application homepage.	The user should be logged in and be able to start playing Monster Mash application.	Pass	N/A
SE-N06-004	FR7	Creating an account using a user name that has already been taken.	User attempts to create an account using a user name already taken.	An error message should appear saying that the user name cannot be created and it should explain the reason for this error to the user.	The user should not be able create an account.	Pass	During the first set of tests this failed as user was able to create a test but was then corrected.
SE-N06-005	FR7	Creating an account with an email address that is already registered to a user.	The user will attempt to create an account using an email address already registered.	An error message should appear alerting the user.	The user should not be able to create an account.	Fail	User is able to create more than one account with same email address.
SE-N06-006	FR7	Case sensitive login.	User will attempt to login using a capital or lower case letters incorrectly.	An error message will appear.	The user should not be allowed to log in.	Pass	N/A

7.1 Testing User Login Information

The following tests Pass/Fail in regards to the current User Login information.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-007	FR7	Login using correct details.	User will enter correct login details and click Login button.	The correct details will be accepted by the server.	User will be taken to Homepage and able to access the features of the Monster Mash application.	Pass	N/A
SE-N06-008	FR7	Login using incorrect password but correct username.	User will enter correct user name with incorrect Password and click Login button.	An error message will appear informing the user that either the user name or Password is incorrect.	The user won't be logged into their account until correct details are entered.	Pass	N/A
SE-N06-009	FR7	Ensure that both username and password have been entered into the login field.	User will attempt to login with one of the fields left empty.	An error message alerting user of the problem should appear.	User shouldn't be allowed to log in until both fields and filled out correctly.	Pass	During the first set of tests this test failed as user was able to login, however has now been fixed.

7.2 Homepage/Friend List Tests

The following tests Pass/Fail in regards to the current functionality of the homepage and the ability for the user to interact with the friends list.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-010	FR2	Locate all available links to the user and interact with them and test their functionality.	All of the link locations within the application.	The link goes to the correct page..	If the link is linking to the correct location and the page is active.	Pass	N/A
SE-N06-011	FR1	Test if the user can logout successfully.	Select the logout link.	The page template/source.	The page should identify the fact the user is logged out and is unavailable to interact with the application.	Pass	During the first test the logout feature failed but was fixed.
SE-N06-012	PR2	Test that the graphical user interface and graphics show correctly to the user.	Test that the graphical user interface and graphics show correctly to the user.	Pages should be displayed correctly.	The images should be consistent with the original design.	Pass	N/A

7.3 Battle Screen Tests

The following tests Pass/Fail in regards to the current functionality of the Battle Screen.

.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-013	FR11	Check that the friends list displays correctly to the user when it is not sorted by any custom settings.	The page containing the friends list should be loaded and there should be no specific sorts or filters specified.	The output page source for this page with no friends, lots of friends, and with the maximum amount of friends.	The pages should return standards compliant HTML with no errors encountered. The sorts should be correct and a visual inspection should also be carried out.	Fail	There was no server-server interaction so no friends list could be displayed.
SE-N06-014	FR11	Check the friends list displays correctly when it is sorted by recent activity.	The page containing the friends list should be loaded and there should be only recent activity showed.	The output page source for this page with no friends, lots of friends, and with the maximum amount of friends.	The pages should return standards compliant HTML with no errors encountered. The sorts should be correct and a visual inspection should also be carried out.	Fail	There was no server-server interaction so no friends list could be displayed.
SE-N06-015	FR9	Check the add new friend function works correctly.	A friends should be added using the friends list form.	The updated friends list containing all the friends, or the database.	The friends list after the friend has been added should now contain the newly added friend.	Fail	Due to no server-server interaction no new friends could be added. However two instances of the same server could send and receive requests.
SE-N06-016	FR6	Check the delete friend function.	One of the friends selected should be deleted using the friends list delete function.	The friends list.	The new friends list should not contain the deleted friend.	Pass	N/A

7.4 Battle Screen Tests Continued

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-020	FR8	Check that Friend button has been pressed from the side of the battle screen and display a message.	User clicks on Friend button from battle screen.	An error message is displayed to the user. User cannot leave the battle screen.	Display error message.	Fail	N/A
SE-N06-021	FR8	Check that Shop button has been pressed from the battle screen.	User clicks on Shop button from battle screen.	An error message is displayed to the user. User cannot leave the battle screen.	Display error message.	Fail	N/A
SE-N06-022	FR9	Check that Add friend button has been pressed from the battle screen.	User clicks on Add Friend button from battle screen.	An error message is displayed to the user. User cannot leave the battle screen.	Display error message.	Fail	User couldn't add friend due to no server-server interaction.

7.5 Farm Tests

The following tests Pass/Fail in regards to the current functionality of the Monster Farm. (E.g viewing monsters, their stats and picking a primary battle monster.)

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-023	FR1	Obtain your monster's statistics	Go to the "Monster Farm" page"	A list of your monsters with associated statistics should be displayed	The page correctly displays your monsters and their statistics.	Pass	N/A
SE-N06-024	FR2	Choose your primary battle monster (i.e. the monster you use to fight)	Click on the check box/button next to the monster you wish to make primary battle monster	That monster should now be selected to be your primary battle monster	The chosen monster is now your primary battle monster.	Fail	User was unable to select a primary battle monster.
SE-N06-025	FR3	Exit the Monster Farm	Click on a link that returns you back to a previous screen	You should exit the Monster Farm and be taken to the location you chose	The page changes to the location your chosen location.	Pass	N/A

7.6 Breeding

The following tests Pass/Fail in regards to the current functionality of the breeding feature of the application.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-026	FR6	Monster statistics are different for offspring.	User should breed a monster.	A new monster should appear in the Monster Farm.	The new monster should have completely different statistics to the parents.	Pass	N/A
SE-N06-027	FR6	Breeding with two different monsters.	User should select two different monsters to breed with.	A new monster should be created.	A new monster should be listed in the Monster Farm.	Pass	N/A
SE-N06-028	FR6	Breeding with two instances of the same monster.	User should select the same monster twice to breed with.	An error message should appear and not allow this.	User should be asked to enter two different monsters.	Fail	User is allowed to breed with two instances of same monster.
SE-N06-029	FR6	Ensuring the user has enough money to breed with your monster.	User selects a monster to breed with.	If insufficient funds an error message should appear.	User should not be allowed to breed with your monster.	Fail	Allowed user to breed regardless of funds.
SE-N06-030	FR6	Must enter unique name for each monster.	User enters the a name for monster already in use.	An error message should appear.	User should not be allowed to have a monster with the same name.	Fail	If same name is entered or if no value is entered in monster name then the monster created is given the same as previous name.

7.7 Shop Tests

The following tests Pass/Fail in regards to the current functionality of the Shop within the application.

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-031	FR8	Check that the Shop GUI is displayed when the user clicks Shop from the homepage.	User clicks Shop button from the homepage.	The shop GUI should now be displayed to the user, including a list of monsters which the user can Buy and Sell and the ability to View V-Money.	Shop GUI is correctly displayed.	Pass	N/A
SE-N06-032	FR5	Check that the Sell function works correctly.	User selects what monster they chose to sell in the Shop GUI and presses the Sell button.	The monster should have been removed from the users Monster Farm and the amount of V-Money that the monster is worth based on statistics should be added to the users V-Money Total.	The correct amount of V-Money has been added to the users V-Money total and the Monster sold has been removed from the users Monster Farm.	Fail	Removed from farmer but money isn't updated unless user logs out and logs in again.
SE-N06-033	FR8	Check that the view V-Money function works correctly.	Within the Shop GUI, the user should be able to view V-Money on the GUI this and shows how much currency the user has.	The user should be displayed their correct amount of V-Money in the GUI.	The user is successfully displayed their total V-Money amount.	Fail	Displays money but doesn't update figure.

7.8 Shop Test Tables Continued

Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-034	FR6	Check that an error message is displayed if the user attempts to purchase a Monster from the Shop but they don't have enough V-Money.	User views the monsters available from the Shop list, selects the Monster desired and presses the Buy button and an error is displayed due to insufficient funds to purchase that monster.	An error message should be displayed informing the user they have Insufficient funds to purchase the monster desired.	An error message is successfully displayed to the user informing them they have insufficient funds to purchase the monster they desire.	Fail	No error message appears.
SE-N06-035	FR6	Check that the Ok button works on the error message regarding the user having Insufficient Funds when trying to purchase a monster so they can return to the Shop GUI.	The user clicks the Ok button to acknowledge the error message and then returns to the Shop GUI.	Once the user has selected the Ok button, they should be returned to the Shop GUI.	The user presses Ok and they are successfully returned to the Shop GUI.	Fail	No error message appears.
SE-N06-036	FR6	Check that the preloaded database of Monsters within the Shop are purchasable.	The user selects one of the preloaded monsters from the database and selects Buy.	The user should be able to purchase the monsters from the preloaded database. The correct V-Money from their total should have been removed and the newly acquired monster should now be in the users Monster Farm.	The monster from the preloaded database should now be in the users Monster Farm.	Fail	Only Gary stored in database.

7.9 Shop Test Tables Continued

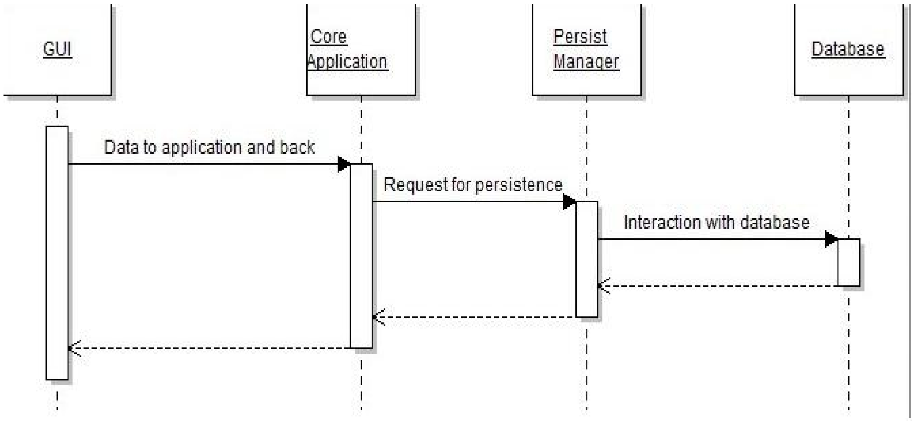
Test Ref	Req being tested	Test Content	Input	Output	Pass Criteria	Pass/Fail	Comment
SE-N06-037	FR6	Check that the preloaded database of Monsters within the Monster Farm are sellable.	The user selects one of the preloaded monsters from the database in the Monster Farm and selects Sell.	The user should be able to sell the monsters from the preloaded database (Monster Farm). The correct V-Money should have been added and to their total and the Monster selected to sell should have been removed from the users Monster Farm.	The monster from the preloaded database has been removed from the users monster farm and the users V-Money total should be correct.	Fail	Can sell monster but money doesn't update until user logs in again.

8 The Project Maintenance Manual

8.1 Program Description

Our database is set up with Java Persistence API which uses the open source Hibernate Library. It runs via JavaDB (or derby). This is changed from HyperSQLwhich we were originally going to use due to connection problems. The database tables are created using persistable 'entity' classes through a persistence.xml file. The PersistManager class handles interactions between the database and the server. The database is reliant on the PersistManager and vice versa.

8.2 Program Structure



As can be seen from the image above, the GUI interacts with the core application i.e. the login, fight and breed methods etc. These methods then interact with the persistManager by sending entity objects to it so that it can send SQL statements to the database to create, remove and update records (and more) in the database. These objects can then be retrieved and checked with the database.

8.3 Description of Sub-Parts

Friends class - A persistable or entity class used to relay information from the database table friends.

FriendsFactory class - Contains a constructor method to create a new Friend object. Entity classes cannot have constructors. However new entity objects can be created quickly, efficiently and neatly.

Monster class - A persistable or entity class is used to relay information from the database table Monsters. This contains getters and setters for all attributes of a Monster. (i.e: Name, Height, Age, Strength, etc.)

MonsterFactory class - Contains a constructor method to create a new Monster object. Entity classes can't have constructors to quickly, efficiently and neatly create new entity objects and a factory class is used for that object.

MyUser class - A persistable or entity class is used to relay information from the database table myusers. Contains getters and setters which relate to the management of users. User is an entity class which is persisted to the database.

RequestFactory class - Contains a constructor method to create a new Request object. Entity classes can't have constructors, therefore can quickly, efficiently and neatly create new entity objects a factory class is used for that object.

PersistManager class - Handles transactions with the database through SQL commands. Contains create, update, remove and search methods for each object type, and misc methods for dropping tables, initilising and shutting down the persistence manager.

- void create(Monster monster) - Adds the given monster to the database.

- void create(Requests request) - Adds the given request to the database.
- void void create(Friends friend) - Adds the given friend to the database.
- void create(MyUser user) - Adds the given user to the database.
- boolean update(Monster monster) - Updates a given monster in the database.
- boolean update(Requests request) - Updates a given request in the database.
- boolean update(Friends friend) - Updates a given friend in the database.
- boolean update(MyUser user) - Updates a given user in the database.
- boolean remove(Monster monster) - Removes a given monster from the database.
- boolean remove(Requests request) - Removes a given request from the database.
- boolean remove(Friends friend) - Removes a given friend from the database.
- boolean remove(MyUser user) - Removes a given user from the database.
- boolean dropTable(String table) - Drops a given table from the database.
- MyUser getUpdatedUser(MyUser user) - Returns the most recent version of a user from the database.
- Monster getUpdatedMonster(MyUser user, Monster monster) - Returns the most recent version of a monster from the database.
- ListMonster>searchMonsters() - Returns all monsters.
- List Monster>searchMonsters(String username) - Returns all monsters for a given user.
- List<Requests>searchRequests() - Returns all requests.
- List<Requests>searchRequests(String username) - Returns all requests for a given user.
- List<Monster>searchGraveYard(String username) - Returns all monsters that are dead (Not used).
- List<MyUser>searchUsers() - Returns all users
- List<Monster>searchShopMonsters() - Returns all monsters in the shop (Monsters that do not have an owner.)
- List<Friends>searchFriends(String theUser) - Returns all friends for a user.
- void in it() - Initialises the persistence manager and connects to the database.
- void shutdown() - Shuts down the persistence manager and exits from the database.

Requests class - Contains getters and setters for the fields contained in Requests.

UserFactory class - Contains constructor methods to create a new MyUser object.

8.4 Recognition of Potential Errors

There is no sanity checking of variables used in the database. It is probably possible to inject SQL commands into the program. All of the variables used in the SQL queries would need to be escaped or checked before running the command. Due to the use of the Persistence API there is likely not much danger of SQL injections however the group has not explicitly tried to sanitise our input.

8.5 Upgrades

We connect and shutdown the database connection very regularly and our program can be quite slow. We believe that this is because of regular connecting and shutting down of the database. If we connected and shutdown the database on the start and end of the application i.e. only the one time each then it would probably make the program faster.

8.6 Algorithms Used

8.6.1 Fighting

This algorithm takes two Monsters and compares their statistics. There are three methods. One that compares their attributes which returns a percentage likelihood of the Monster in the first parameter position of how likely that Monster is to win. (Obviously the second can thus be worked out by doing 100 minus the first Monsters percentage.

Next a method that randomizes these probabilities is used. This allows for a slight chance for the less able monster to win if the odds are severely in the more powerful Monsters favour. Lastly a random wheel is used to select a Monster which picks a number from 1 to 100 and if the first monster has say a 70% chance of winning, then the wheel picks from 1-70 and the first Monster will win or the less powerful Monster will. Another part of the application then handles the deletion of the losing Monster from the database.

8.6.2 Breeding

This algorithm looks at the stats of both parents. It then takes the middle ground of the parent's attributes and selects using a random wheel. E.g. a value between 1/8 less and 1/4 more than the value of the stat that the less powerful monster has centered around the middle ground.

I.e. If one parent has 40 strength and another had 80. The middle ground would be 60. Then the child strength could be between 60-(40/8) and 60+(40/4) so the child could have a stat of between 55 and 70. This algorithm encourages a slow increase in the average monster stat over generations of breeding as they are more likely to be stronger than weaker than the middle ground.

The new monster is then persisted to the database with the username of the owner passed to the algorithm.

8.6.3 Trading

This algorithm is very simple. It takes the two Monster entities and swaps their owners with one another. V-Money was not implemented into the algorithm and the algorithm was not integrated into the system as the group ran out of time. To implement it further the group thought that we would have had to be apply it to getting users to accept trades in real time. However we could have probably managed it using requests.

8.6.4 Worth

This algorithm is used to decide on the worth of the monster. The group decided that it would be better to have a Monster get a set worth based on its stats and age rather than for a user to just set a price on it.

The algorithm takes the stats and the Monsters age and decides a cost based on these attributes.

The age is assessed using another method. The closer the value is to halve the Monster's max age the higher its age assessment will be (out of 10). A Monster's age at half its max age is therefore its most powerful age.

8.6.5 Data Areas

The data-structure of the database is obviously the database itself. The database holds tables built using a series of entity classes. These include friends, requests, users and Monsters. These classes are the backend data and hold everything that users need to play the Monster Mash application. Details of their Monsters, friends and the requests for fights, breeding, trading and friends that they have.

The attributes for each entity class are the columns in the table, each entity object is thus a record in the database. For one-to-many relationships like friends (where each user has many friends) a list is persisted into the database.

8.7 Files and Directories Used

For the databases tables to be created and a connection to the database from the application to be applied a persistence.xml file is created. This holds the link to a data-source (the connection to the database) a list of entity classes that the database is based on and a few other things. The database itself is also contained in a file on the system.

9 Interfaces Used

The databases interface/constraint is whatever goes into the writing of the entity classes. This has some constraints however. There can be no constructor in the entity classes (The group got around this by creating factory classes) also only certain data types can be persisted. This includes strings, ints and booleans. For one-to-many relationships, only abstract lists can be persisted (i.e. a list).

9.1 Suggestions for Improvements

The group did not manage to get the email feature working. If this wasn't the case, the group could have stored and accessed the user emails in the MyUser table and potentially send account confirmation emails and updates on monster statistics etc.

As mentioned the application is fairly slow. Most likely due to repeated connections to the database. If we did this only once then the application would probably be a lot faster.

Validation and sanitisation of user input would have been good. This could have been done further up in the applications structure.

Better persistence annotations could have been implemented to create a better and more fitting database. I.e. rather users having a list of friends names (Strings/VarChars) they could have had a list of friends records. The group tried to implement this but we did not have enough JPA experience to manage it.

Concurrency control would have been a good feature. This could have been implemented using a queue of commands to the database, probably with an optimistic locking mechanism on the database. If we did the application again this is definitely one of the first things we would have tried to implement.

9.2 Things to watch for when making changes

Watch out for inconsistencies in class names. For example, Friends, Monster, MyUser and Requests.
Be careful when adding or removing attributes in the entity classes. They are used throughout the program, but also are a part of the persistent data which will have an affect on the database i.e. could corrupt an existing one, make the database incompatible or corrupt individual records in the database that are there before the altering of the entity classes.

9.3 Physical limitations of the program

There are no notable limitations. The application does have persistent data so the more users that are registered the more disk space the application (database) will take up. This will be fairly small however unless the amount of users reached a ridiculous/professional amount. The larger the amount of users logged in at one time the more memory the server will user trying to process requests. So many users could cause the server to slow down or potentially crash.
The server uses threads and too many users could cause too many threads to be active. With some more engineering and development this issue could have been solved likely by either re-thinking the use of threads or by merging them.

9.4 Rebuilding and Testing

When rebuilding the system the entity classes must be the same. The hibernate and derby libraries must also be in the system (amongst others no significant to the datatbase). The persistence.xml file must also be set up correctly. This is done via the following:

- Create new Netbeans Java Web project from existing sources
- Set Location to N06/src/Main
- Make sure project name is Main
- Set Web Pages to N06/src/Main/web
- Set WEB-INF to N06/src/Main/web/WEB-INF
- Set Libraries to N06/lib
- Finish
- Switch to services tab, right click JavaDB and create database
- Call it monsters with username/password APP
- Switch back to Project tab
- Delete META-INF folder in Source Packages
- Right click project, create New ¿ Persistence ¿ Persistence Unit
- Change name to monsters
- Change Provider to Hibernate (JPA 1.0)
- Click arrow on right hand side for new data source, name to MMDB5
- Finish
- Edit the persistence.xml with transaction-type=”RESOURCE_LOCAL”
- Add underneath jta-data-source: (<class>databaseManagement.MyUser</class>/ <>databaseManagement.Monster</cl <>databaseManagement.Friends</class>/ <>databaseManagement.Requests</class>
- Run Project

9.5 Any known bugs

There is one known bug with the database:

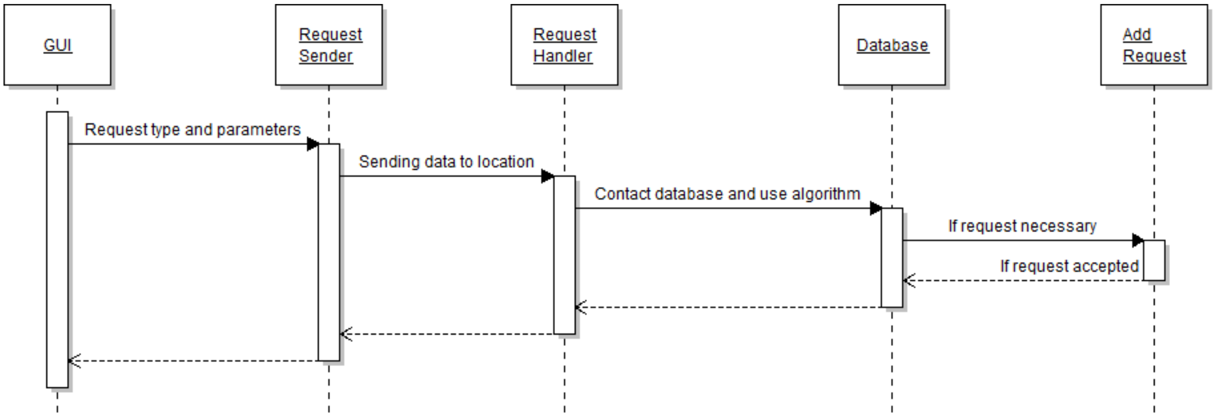
- The server is slow. This is likely because of repeated connecting and shutting down of the database. This could be solved by connecting to the database when the application starts and shutting down when it ends. This would have been simpler to code most probably as well however with inexperienced with JPA this is what was implemented to start with and was not modified.
- The use of threads in the application (which use and persist with the database) stack with users. These threads need to be merged with extra users rather than just cascade. If there are too many users there will be too many threads which can cause a memory error.
  There needs to be proper server-side validation after data has been passed into the program. This will avoid problems with SQL injection, and also solve the HTML/JS injection mentioned in the User Interface section. There is no validation or sanitisation against injections. We originally did implement some regular expression checking validation however this was removed for access to the code and was never integrated back into the system. Thus no validation or sanitisation was present in the final system.

10 Server Server Interaction

10.1 Outline

Server to server interaction has been developed using POST requests being sent using methods contained in RequestSenders package. The handlers would then get the data, and if resulted in having to return something - it would be done by sending back a JSON object back.The JSON serialization and deserialization is done by using a GSON library, open source from Google.

10.2 Program Structure



The user uses the GUI to send a request and has to select which location it is to be sent and the request is sent to a servlet handler on local or remote server. It is then handled appropriately - The request for Monsters for sale happens instantly, the receiving servlet needs to contact the database and send back the data in a JSON object which is then deserialized on the server. For fight, friend and breed requests the servlet needs to add a request to the user database and wait till it is accepted to send the data back.

10.3 Description of Sub-Parts

10.3.1 BreedRequest class

sendBreedRequest(int height, int aggression, int str, int age, String newName, String remoteMonster, String toWho, String fromWho) - Sends a request to the server with all the necessary information needed to breed with another monster.

10.3.2 BuyRequest class

sendBuyRequest(String monsterName) - Sends a request to the server with the name of the monster to be bought.

10.4 RequestFight class

sendFightRequest(int aggression, int str, int age, String enemy, String thisUser, String toUser) - Sends a request to the server with the necessary details for the fight to happen, details are mostly required by the fight algorithm implemented on the target servlet.

10.5 RequestMarket class

sendMarketRequest(String location) - Sends a request to the server asking for all the monsters that are currently for sale in order to display it to the user.

10.6 RequestUserMonsters class

getUserMonstersRequest(String username) - Sends a request to the server asking for all the monsters of the specified user(Has to be friend).

10.7 AcceptFriendship servlet

doPost() - Takes the information from the user input and accepts a friendship request, adds it to the friends database and removes the request from the list. It should also send back the information to the server from which the request came from informing about the acceptance.

10.8 BreedServlet servlet

doPost() - This servlet takes all the necessary parameters from the POST request received and carries out the breed algorithm, returning a new monster serialized into a JSON object.

10.9 BuyServlet servlet

doPost() - Sells the monster to the user if available - Serializes target monster as JSON, sends it back to the request server and deletes the monster from the local database.

10.10 FightServlet servlet

doPost() - Takes necessary parameters from the POST request, carries out a fight algorithm, depending on who has won, the response is sent back.

10.11 FriendshipServlet servlet

doPost() - Receives a POST request from the server and takes an username attribute, queries for the user in local database and adds a friend request.

10.12 FightResponse class

Used to deserialize a JSON object returned after the fight has happened.

10.13 FriendshipResponse class

Used to be addressed when the user accepts a friend request. Will handle it and add the user to the database as a friend.

10.14 Recognition of Potential Errors

The crash may occur when the remote server is offline, it would likely return an IO error in this case, to fix this issue the location to which the method is calling needs to be checked. Once this is done, the method can be simply ran again and should work.

Problems may also occur when requesting data that has no validation - Although all care has been taken to minimise those, some may remain.

10.15 Upgrades

The design can be altered without entire modification, the structure of request and handlers is rather simple and there may not be many ways to speed it up as I tried to keep it minimal.

10.16 Algorithms Used

Algorithms used for all requests has been described in the same section above.

10.17 Data Areas

All the data in the request senders/handlers are temporary and it is not stored anywhere, as it is only serialized when requested from the database. Then it is deserialized and either added to the database, or displayed and would be destroyed once a user exists the page.

For example, a list of monsters for sale would be sent back to the request location within a JSON Array of objects.

10.18 Files and directories used

Request handlers and senders would not create any additional files, however, it would assume that the classes that they address will be available on the server, as those are hard typed and a part of the API. If the location is unavailable, an IO error will be returned and the operation halted.

10.19 Interfaces used

The request handler classes cannot address a variable that is not being sent in a request and then attempt to use them in a certain way. Those will be NULL and return an error. The receiving server needs to comply with the API for communication.

10.20 Suggestions for improvements

While we did not manage to get all of server to server code to work properly with the GUI, it would need to be introduced, as mentioned before. The database has not been complete for a very long time and the group did not manage to get everything together on time, therefore the improvement here would be to actually finish the integration of the code.

The validation on the handler's side would need to be introduced to reduce the possibility of an error, as well as additional classes would need to be designed in order to handle the responses to requests.

10.21 Things to watch for when making changes

If a change is made to the request sender class, for example when adding another parameter to be sent to the handler, make sure that the handler expects that parameter because otherwise it would not be recognised and will be lost. Also, deserializers need to contain appropriate variables and getter/setter methods as being received.

10.22 Physical limitations of the program

Most of the data sent/received is relatively small. The only place where an issue of space and bandwidth may occur is when requesting a monsters for sale, as the data that is being sent back may be very big for a larger application with more users. For example, a system that will return an array of 5 monster would be very quick to handle, however, one with more than 200 units would likely take longer and may be an issue.

10.23 Rebuilding and Testing

The senders and handlers are located in appropriate packages, the way to test the functionality of server - server classes on both sides are to send appropriate data and then deserialize what has been sent back - If it is what the tested expected, it works perfectly fine. For example, a breed request would send back a new monster with new attributes and new name.

10.24 Any known bugs

We did not get the testing stage of the program, therefore we were unable to test all functionality. However from the command line, the only bug that may occur is the IO exception when the location to which the request has been sent is not available or doesnt exist, the exception would need to be caught and dealt with appropriately.

Exceptions may also occur on the handler side, it would likely relate to having too few parameters in the request or the database returning unexpected values, which should not happen. The functionality will work well when the expected data is present on the handler side.

11 User Interface

11.1 Outline

The user interface is the part of the program that displays the information from the backend to the user. There is a web based user interface for this project. The user interface was constructed using the Twitter Bootstrap framework to keep development time to a minimum. You can find more information on Bootstrap on the Bootstrap website http://twitter.github.com/bootstrap/index.html.

The users are presented with a clean page that offers them options of what to do next as links on their current web page. There are different pages for each of the different aspects of the game. For example The Shop, The Fights, Logging in/out etc.

There is no administration interface. To administer the Database you need to directly manipulate it with other tools. For example PHPMyAdmin allows you to administer a MySQL database online.

11.2 Program structure

The majority of the user interface code is split into .jsp files. You can find these files in the web directory on the project root. You can find the parts for the header, the footer and the sidebar in the parts folder which is located in the web folder.
There are several different templates that make up the user interface, the header.jsp and footer.jsp parts are included on every page, this is for convenience. You may edit these pages/parts and every page that includes them will change with it. The rest of the pages are also .jsp files that can be directly edited if necessary.

11.3 Description of Sub-Parts

- Includes Folder - This contains parts that are in existence for compatibility with a type of java server page that we are not using for the rest of the UI, unless you want to start using this again, you should ignore this folder and everything inside it.
- Resources Folder - The resources folder contains all of the static files that need to be served by the web server in order for the website to display correctly, the consist of things like images, JavaScript files and CSS Stylesheets.
- WEB-INF Folder - This contains an XML file to tell glassfish how to handle the web folder.
- Parts Folder - The parts folder contains all of the .jsp pieces that are repeated over a number of pages, for example there is one for the header that contains all of the code that should go at the top of every page.
- about.jsp - This jsp renders the about page of the website.
- Bread.jsp - This is a loading screen while it takes the two choices the user has and sends it the UserController class to make a new monster and add it to the database. Once done the Farm.jsp will reload with your new monster.
- failedLogin.jsp - This is the page that is rendered if the user fails to log in correctly from the login page.

- logout.jsp - This is the page that is rendered when the user is logged out of the system.
- processRequest.jsp - This page is to process a friend request.
- sellMonster.jsp - This page is rendered when a user wishes to sell one of their monsters.
- shop_sell.jsp - This page is rendered when a users is trying to sell a monster in to the shop.
- addFriend.jsp - This page is rendered to present all of the options to a user when they want to add a new friend.
- buyMonster.jsp - This page is rendered when someone is trying to buy a monster off another user.
- farm.jsp - This page is rendered when a user tries to use the farm.
- index.jsp - This is the default page, it is the first one rendered.
- Outcome.jsp - This page is rendered to display the outcome of a fight that they have just been involved in.
- SendFriend.jsp - This page is rendered when a user if trying to send a friend request to a user on this server or on another one.
- battle.jsp - This page is rendered to display the various different aspects of the monsters fighting each other. It uses the outcome.jsp to display the outcome.
- CreateUser.jsp - This page is rendered to allow new users to create an account in the system that they can later log into.
- Fight.jsp - This page is the loading page to work out which monster has won the fight once done it loads the Outcome.jsp with the winning monster.
- ogin.jsp - This page is rendered to display the login page to a returning user.
- SaveName.jsp - This page is rendered to allow a user to save a name change in the system.
- shop_buy.jsp - This page is rendered to allow someone to buy a monster from the in game shop.
- welcome.jsp - This page is rendered to welcome the returning user that has logged in again.

11.4 Recognition of Potential Errors

Potential errors that might occur in the User Interface are things like spelling mistakes and grammar issues, you may find these errors while using the web interface. To fix an issue like this you would first have to find the .jsp file that the error is in. This should be a fairly easy process. The .jsp files have names that reflect the part of the website that the relevant page. For example welcome.jsp is the welcome page at the beginning. Once you have figured out which file the changes need to be made in, you can simply make the change using your preferred editor and save the file. The next time the page is rendered in a browser it will show your changes. If you have a web cache then it may require you to reload the page before you see the changes.

Another issue you may have is with regards to the website rendering incorrectly in a particular browser. The fact that we are using Bootstrap helps to avoid these issues because the framework is fairly well tested. If you have an error, you need to again figure own where the error located. The first step is to try the .jsp file that renders incorrectly. If the error is not fixable from that location, you may have to look through the stylesheets, and the other static files that are included into the page. There are a number of included files. Again you simply need to make the changes and save the file. The static content will be in a directory on your webserver somewhere. Like with fixing grammar errors you may have an issue with web caches.

As the UI is almost completely limited to the .jsp files and the static files mentioned above, most errors that can occur, you should be able to fix by following the instructions for either potential error above.

11.5 Upgrades

You may wish to upgrade the pages style as it goes out of fashion, this would be as simple as upgrading all of the relevant jsp files CSS stylesheets and Javascript files in the web folder and then saving them. It may be a good idea to make a backup of your current files unless you need to revert to them in the case of an unforeseen error.

You may also need to upgrade the versions of some of the libraries used to display the UI in order to get better performance, security updates, style updates or new features from in the new versions. The libraries used are served by your webserver, probably from the resources folder in the web folder of the project root. You need to make sure that the new versions support all of the old features otherwise you may have to upgrade the JSP files also. This can be done by simply editing the jsp files to work in accordance with the new libraries. Watch out for upgrades in the Javascript library because there have been modifications to make the friends list drop upward.

11.6 Files and Directories Used

The files for the UI are all located in the web folder inside the project folder. In this folder there are 3 directories described above under the parts section. there is a simple diagram below to illustrate the file structure.

Main - Project directory
- parts Folder - Contains the parts that are included
- header.jsp - included in the web page
- footer.jsp - included in the webpage etc
- includes Folder - For compatibility only etc
- WEB-INF Folder - Glassfish configuration etc
- The .jsp files that represent the websites are in this directory etc

11.7 Suggestions for improvements

11.7.1 New user Tutorial

When users are new to the system they will have no experience with using the various parts. This means that they will lack the experience to interact with the game properly until they have spent a significant amount of time working out the different functions. They may never discover some of the more obscure features.
This could be easily remedied by providing users with an optional tutorial at the beginning of the game, just after they have registered. This would also allow the website to showcase some of its best features and allow users to get a more full experience from the application sooner.

11.7.2 A new set of Graphics

The graphics provided may need to be replaced anyway as time goes on in order to appeal to the audience that are using the application. You may however want to brand the game with either a company logo or with a new logo to represent Monster Mash. This is as simple as replacing the graphics with your new ones in the resources folder inside the web folder.

11.8 Things to watch for when making changes

When making changes you need to make sure that all of your code is compliant with the standards, this helps to ensure that the page is rendered correctly in most browsers. You can validate code online at the validator.w2.org website.
Because of the nature of web design there are many includes in the templates, you need to make sure when moving them that the references to them remain correct in the other files. For example if you rename or move the header.jsp file you will need to change the reference in every file that links to it, which could be a big task.

11.9 Physical limitations of the Program

The load time on some computers may become an issue. Especially if the computer has a very slow network connection or connection to the internet. This may cause a delay in the page load for some users.

The servers connection to the internet is also a bottleneck here, if there are many concurrent connections to the server and there is not enough bandwidth to keep all of the connections running at full speed users may experience a Denial of Service.

In some cases malicious users may use something called a Denial of Service attack (DOS) during this kind of attack they deliberately invoke the symptoms described above in order to cause real users a service outage. There is little you can do to avoid these attacks, but you can try to cache as much of the information as possible so that the server is doing less work and serving less information on the whole. The server can if caching is implemented therefore cope with more users (legitimate or not).

11.10 Rebuilding and Testing

When you have modified or updated any of the UI files, you need to check that they are accurate and that they conform to the web standards. You can use the online validator at validator.w3.org to do this. You also need to do a visual inspection and link checking to make sure that all links work as they should and that the changes have taken effect as they should. To perform a visual inspection and link checking you can simply start the server on your local computer and browse though the website checking each page. Its possible that some errors that show are not caused by the UI code.

11.11 Any known bugs

11.11.1 Some aspects of the site are vulnerable to HTML/Javascript injection

This is a major security issue as it potentially allows malicious users to put any code into our website and have it execute on our users computers wherever the information is displayed by the template. This can cause malware to be installed on users computers and will be extremely detrimental to the brand of this website. Because we were not tasked to ensure that the website is secure, the group left this issue for another team to sort out at a later date. There are two steps to fix this issue. First you will need to add validation, so that inputted information can only contain the intended characters. This validation should take place twice, once on the users pc for convenience, and once on the server to make sure that information is accurate before storing it. The next step is to make sure that all fields that arent trusted have their data escaped before they are rendered. escaping data before it gets into the database is a good idea. There are some aspects of validation in the application already and you should be able to modify those bits of code to cover the rest of the inputs on the website. For example some of the other inputs on the shop.

12 Personal Reflective Report

12.1 Felix Farquharson

Name: Felix Farquharson
User Name: fef
Group Number: N06
Role within the group: Version Control/UI Design

The group I was assigned to was very mixed, everyone had a willingness to complete the task at hand, but it was clear that everyone had very varied skills in a project like this. It quickly became apparent that in order to get the best possible outcome you need to play to everyone's strengths. Sometimes however, you aren't doing something you have done before and there is a steep learning curve and a looming deadline. I think the best way to avoid this is to be vocal (in the right situations) about the things that you feel less comfortable doing. While I found it more productive working on my own, I also realised how important face to face collaboration can be when working with a group to complete a team goal.

I think if we were given the chance to complete a similar project as a group, we would be much more prepared for it. Problems we encountered that will certainly be more prepared for in future are as follows: Importance of good planning from the beginning, just because it's not something you're told to do doesn't mean that you won't be able to spot something useful that might affect you later. Using the same tools, in the past I have stuck to the same tools when using the computer. I learned that the tools that suit you may not be the best choice for the group, and on the whole everyone may spend more time working them out than you gain by learning how to use them. Conforming to everyone's schedule. When working with other people, it is important that you have time to speak to them face to face, and that finding time when you are all available to meet is more difficult than it seems.

12.2 Samuel Mills

Name: Samuel Mills
User Name: sam39
Group Number: N06
Role within the group: Documentation/ Minutes Author/ Javadoc

Looking back to the start of the semester where we were put into groups in comparison to now, I personally believe our group has really progressed and bonded as a team. As a whole, we communicated well, had good determination for the development of the project and overall, we enjoyed while we took part in this project.
When we initially started our meetings for the project we were unsure who to designate as project manager as a few members of the group. As majority of members had not worked with each other before, we thought it would be a good idea for each aspiring project leader candidate to tell us their experience and what they could bring to the group. In reflection I think this was a really good idea as it gave us a chance for members to show what they could offer to the group and to show what areas they are generally specialised in. Conclusively we decided James Slater would be project manager. James really did bring the team together; designated tasks assertively, follow up absenteeism of members and generally ensured the group ran smoothly.

I believe each member and I all brought different qualities and contributions to the group. Being a Business Information Technology I believe my coding and game creation ability was severely restricted in comparison to Computer Science, Artificial Intelligence, Computer Graphics, Vision Games and Software Engineering students. This did cause me to have a lower amount of contribution hours in the first semester in comparison to other group members. However this was to be expected but I believe I made up in contribution in other areas of the project. I was the main documenter of each meeting which we had on a weekly basis; I had to ensure that our future plans were clear and understandable to allow all group members to follow. I and Amy James were also designated document designers, editors and creators alongside testing the actual application. These documents would influence all group members marks or grades so I had a great amount of responsibility and it was imperative to ensure I performed to the best of my ability. Reflectively, I participated as much I physically could in terms of skills and knowledge. I attended a large majority of formal and informal meetings and always arrived in a timely manner and only missed 2 meetings due to illness.

Other members of the group also performed well. I think Ben, Chris, Dan Felix and James all carried out the tasks designated to them correctly and effectively. Unfortunately at the start of semester 2 one of our original members Aiman left the group. This ultimately resulted in having less manpower for the second semester which resulted in us having to reconsider and revaluate the distribution of work and contribute more hours in general.

In semester 1, there were periods of patterned absences between a few members of the group which did cause delay and lack of communication. Even though majority of us ensured everyone understood when the next meeting was by notifying them through email and social media, some members sometimes didnt turn up and not explain their absence. Again this slowed down our progress as sometimes we couldnt make decisions for the future as we were lacking information or input from other members. However, the spell of absences soon subsided and normal routine resumed.

In conclusion, I am very happy and pleased with how the group has worked out. We pulled together, worked hard and most importantly produced a project I'm happy with which was the aim at the end of the day. I have often heard from 3rd year students and other 2nd year students say that they disliked their group and no one did any work apart from them. This is most definitely not the case for CS Group 06 and we all did our bit effectively and always helped each other if someone was struggling. I would take up the chance to work with any members of the group again and I hope this is a possibility in the future.

12.3 Ben Brooks

Name: Ben Brooks
User Name: beb12
Group Number: N06
Role within the group: QA Manager/Coder

During this project, as QA manager, I was responsible for ensuring consistent and high quality work produced by the group. This involved reading through all of the QA documents, understanding what the group had to do and also reviewing code, documentation and any other things the group produced. I feel as though overall the group has put in a decent amount of effort and as a result has produced a reasonable attempt at completing the objective of the assignment. We hit a few pretty major snags on the way, which impacted on the overall final result, however I do believe what we did get done was up to a high standard and had we not hit those snags, we would've had a very good product at the end.
Throughout the project I worked in small teams, usually between 2 and 3 people on things. This was mostly pair programming with James and Felix to integrate the GUI with the backend of the program. One of my main contributions to the coding side of the project was the GUI. Felix had set out a base using Twitter Bootstrap and I then enhanced it with modal dialogs, friend menus, and getting the GET/POST functionality working.

I feel as though this project has given me a very good grasp on how to use Git as a version control system, as I had not used it before this however I feel very comfortable using it now.

I believe that James has been a very good group leader, everything was always clear when instructions were given and very reasonable time frames were set for work. Problems were quickly resolved and he was very approachable with any issues that people had.

12.4 James Slater

Name: James Slater
User Name: jas38
Group Number: N06
Role within the group: Group Leader, Programmer

My Attendance for the unofficial group project meetings was 100% and for the weekly meetings with Nigel was also100 %. I worked Monday to Friday 9am-6pm during the integration and coding week. I feel that the teams attendance was very good and I was very happy with the general commitment from all members of the group.
I feel my communication for this project was excellent as I had to listen at the start to find out and know what all the members were able to do, what they wanted to do and what had to be achieved in the project. I then had to talk to each member either as a group or individually, and express what was expected and needed of them as a contributor. I feel I did these tasks very well as I only had a couple of people coming to me saying they had to much work to do or they were unable to work because they did not know what they were doing but we managed to sort it out and get the job done in the end. I also had to be able to keep everyones personality clashes to a minimum and always be approachable for anybody to talk to me about any problems they may have and for us to solve the issues.

As group leader I had to have very good organizational skills as I had to know what was needed and the time scale it needed to be done by and if it needed anything finished before any task can be started. I had to organize all the meetings and make sure everybody could make it. I found this very challenging to keep everybody happy and as I found that everybody works at different times and do not always communicate with each other very well. With that said all the tasks were still completed and the team did very well when it came to completing the tasks assigned to them.

I had to create the Gantt Chart for all members to work off and I found this rather simple. I only had to carry out some minor changes for different individuals and as Aiman left I had to redo the Gantt Chart to show this. I also had to help Sam with a small segment of work in the first semester because they had other commitments. I also had to help out Ben and Felix with creating the GUI for the prototype hand in on the 10th of December and I had to piece together their work to make each page was correct. For example I made the header, footer and the side bars and left the body of each page free available for each member of the GUI team to create and edit. These were later tidied up by Felix and Ben.

My main tasks were to bring the GUI and the database together and to create controller classes for the UI to function properly. I found this very challenging as I needed to understand the GUI as I am not as skilled in HTML and CSS. I made a new database in the persistent class for the requests.

Overall I feel the team did all their tasks very well and I was happy with how we all worked together. Considering the set back we received we got quite a lot done in the project and that is thanks to everyone pulling together and working effectively on the issues we were having.

12.5 Dan McGuckin

Name: Dan McGuckin
User Name: dam44
Group Number: N06
Role within the group: Programmer
I found the project ran very smoothly. Most people turned up to each meeting and we understood what we each had to do.

My jobs were spike testing of the database, UML and implementation of many parts of the system, specifically handling persistent data and writing the shop, monster worth, fighting, breeding, monster aging, logging in/out, registering and monster/user design.

The spike testing for the database connection had a few problems. I set it up in a regular java project and it worked perfectly. I thought, mistakenly, that this would therefore work for the entire project however when applying it to the web - based java project we found connection problems which continued throughout the week. Looking back I realize that I should have tried connecting it up with a web java project, but in my mind at the time I thought that the difference between the web java project and the regular ar one was just on the front end. i.e. the output using the server with JSP's rather than using a main method. I was evidently mistaken for problems occurred and possibly I should have been slightly more thorough at this angle.

My writing of the persistence class to connect the database to the java code was very successful, largely thanks to the spike testing. Although connection through the persistence.xml had problems, the persistence manager class worked well and we could easily persist, update, remove and search for monsters in the database from the beginning of the implementation and testing week assuming the database connection was working. .

I was also given the task of creating the UML for the project. This included the use - case, class and sequence d diagrams. I felt this went well; we used the use - case diagram to aid us in deciding and implementing our user requirements. The class diagram helped us in putting the system together and the sequence diagram aided in our understanding of how the system was meant to run. I had never created a sequence diagram prior to the project so this was a nice learning curve.

In terms of coding I was tasked with writing a large amount of the core system. I prepared a lot of it before coding week and some I wrote in the first two days. This included the fighting algorithm that determined a winning monster when passed two as parameters. It calculated the winning monster by comparing stats and using a random wheel. I also wrote the breeding algorithm which used much of the same techniques but generated a new monster by taking the parent stats and using these with a random wheel.

I also wrote the shop (buy and sell) algorithms. This involved changing/updating monster and user details in the database. I also wrote the algorithms for determining a monster worth, this was used by the shop and looked at the monsters stats and decided a worth based on those and it's age. I assessed the monsters age with another algorithm which outputted a number between 0 - 10 depending how f ar the monster was from its peak age. This number would then change the monster's effectiveness in battle and alter its worth.

I wrote the logging in/out and registering algorithms. These accessed the database to check if user names and passwords were c correct and created new entities in the database when registering. I initially added some regular expressions for sanitizing input but we commented this out for ease of access and I did not have time to implement it at the end.

I also created the monster a going algorithm. I did this by using a separate time thread when the user was logged in. The theory of this was that when the user logged off the monster would stop aging and would only age so long as the user was active. The thread worked but was not perfect ct and did occasionally create some memory issues. I created the algorithm to check if users were logged in, this was for if a user had closed their window without pressing the log- out button or if the user had gone away from their computer for a long time . This was another thread that activated (like the aging thread) when the user logged in and died when the user was deemed as logged out. When users did things like breeding monsters it would change their state to active, if after so many minutes they did not do anything to change their state to active they were deemed as logged out.

If I could do the project again I would make sure the database was 100% working on a Java Web application before coding week, I made the mistake of not testing it on this platform and it cost us time. I would also make our time management better, especially in terms of having more code completed by coding week (Although most of it was).

I teamed up with Ben during coding week to aid in integrating the back code with the jsp's and to attempt to get our friends list working. I found this pair coding s style fun, productive and I thoroughly enjoyed it.

I found the team dynamic flowed with no personality clash or in - house fighting. We were directed well by our project leader and we all got on very well to get the job done. Members worked very well together r with a united persistence and determination to get the job done to the best standard we could. I found that by the end, we had all handled the stress well and I found that I was proud to be a member of this team. I would work with them all again and enjoy yed almost every second of it. I wish I could write more on this subject but there is little drama to write about, we worked efficiently and I enjoyed it.

Our team leader (James Slater) was very organized and le d the team well. We, as a group worked well under his easy, relaxed yet determined leadership. He motivated and inspired the group to work efficiently even under a lot of pressure. Although almost all the members consistently performed, when one or two slacked for certain periods of time e he dealt with them fairly and ultimately I believe the reason almost everybody in the team worked so consistently was influenced a lot by him.

I always knew my task throughout the entire project, and if I ever had a question James was ready to answer it . We had group meetings every week outside of our compulsory meeting where tasks would be set and questions could be answered by a member of the team. There was not a dictator - like policy in our team, everybody could speak their mind and James would listen.

12.6 Christopher Krzysztof Ilkow

Name: Christopher Krzysztof Ilkow
User Name: cki
Group Number: N06
Role within the group: Main Programmer

The group project has been a great experience for me, I have learnt few things that would be important for future projects like this. I have learned how to properly use Github within a group for version control and how important comments are for other people and myself to understand the code that someone else has written. I have learned the importance of time management and communication within the group, as well as the fact that sometimes it is necessary to change things at last minute in order to deliver the project on time.
In my opinion the groups were too large for this kind of a project, and sometimes there was no work to do for certain individuals, I usually prefer to work on my own in university projects, and I am not a fan of documentation which is why there is not much of input there from myself.

I think that if we were given the chance, the project would be done much better especially in terms of time management and making sure that fundamental elements of our software are being delivered as expected. As a group, this was a new experience for us all, at least in such a large format which is the reason why certain aspects were left in an unfinished state.

My role in the team was to ensure that our project is able to communicate within itself and to a standard with other groups. I have attended few meetings about server-server communication, however me and few other people decided to write our own API consisting of POST requests and JSON data, which has been quickly developed in our own time and incorporated into the software.

I have also recommended many ideas and changes to the project, such as a choice of IDE and server engine software to be used in the development phase.

I finished my task rather quickly, it was possible to query our own server to fetch data about users, monsters, request fight, breed and sell/buy. I have worked with Dans algorithms to achieve few of those tasks. I have shown our group leader (James Slater) that my code works perfectly.

I am slightly disappointed in the fact that our software did not meet our expectations, due to unforeseen circumstances our database has not been functional until the last day of Coding Week, and without this, I was not able to incorporate my own code into functional GUI/Database program, I would say that this was the only reason for which the program lacks few important features.

The group has been great and I am thankful to everyone for making this a great experience.

13 Amy Rebecca James

Amy Rebecca James Name: Amy Rebecca James User name: arj18 Group Number: 06 Role within the group: Deputy QA Manager. Put documentation together ready for deadlines throughout the project. Reflection
During the Group Project I feel that our group has worked really well together. For each deadline our group leader assigned each member to complete a specific part of the document. As a rule all group members had met the specified deadlines set and there has been no last minute panicking to complete a document and to and put it together. If there was even a slight chance that a group member wouldn't have completed the work by the deadline then James was quite happy to contact them and ensure they knew what they were doing and when it had to be complete.

As I'm not a particularly confident coder I have been working on the documentation for the project and concentrated on piecing together the different parts of the documents. I feel that I have played a key part in the organisation of the group. Due to the fact that I cannot code, I have made an extra effort to pull my weight and to contribute to the work by trying to take a larger responsibility for the documentation. During this time I have learnt how to use LaTeX, so that documents were well presented and had a professional look.

This project has given me a good insight into how it would be to work in industry and how projects like this work. i.e QA documents, relating back to the requirements specification. I feel that I have gained valuable skills and improved my team skills in general.

14 References

1. *1] Software Engineering Group Projects Test Procedure Standards.. C.J.Price and N.W.Hardy. SE.QA.06.Release.*
2. *2] Software Engineering Group Project Plan Design Specification Standards. C. J. Price, N. W. Hardy. SE.QA.05a. 1.6*
3. *3] Project Plan - CS221 Project Group 6. Version 1.5*
4. *4] Test specification - CS221 Project Group 6. Version 1.4*
5. *5] Software Engineering Group Projects General Documentation Standards. C. J. Price, N. W. Hardy. SE.QA.03. Version 1.5*
6. *6] Software Engineering Group Projects Monster Mash Game Requirements Specification. B. P . Tiddeman. SE.CS.RS. Version 1.2*
7. *7] Software Engineering Group Projects - Producing a Final Report. C. J. Price, N.W. Hardy and B.P.Tiddeman SE.QA.11. Version 1.7*

15 Document History

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2012-11-10	Initial creation	sam39
1.1	N/A	2012-11-12	Added personal reflections	sam39
1.2	N/A	2012-11-13	Added critical evaluation	sam39