**CS22120 Software Development Life Cycle**

**Group 05 Final Report**

**Author:** Chris Savill – chs17

**Config Ref:** SE.05.DS Group 5

**Date:**

**Version:** 1.0

**Status:**

Group 05

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2012

Table of Contents

[1. Project Plan 6](#_Toc348561224)

[1.1. Introduction 7](#_Toc348561225)

[1.1.1. Purpose of this Document 7](#_Toc348561226)

[1.1.2. Scope 7](#_Toc348561227)

[1.1.3. Objectives 7](#_Toc348561228)

[1.2. Overview of Proposed System 7](#_Toc348561229)

[1.2.1. Technology Being Used 8](#_Toc348561230)

[1.2.2. Use-cases 8](#_Toc348561231)

[1.2.2.1. Use-case for Users 8](#_Toc348561232)

[1.2.2.2. Use-case for Server 8](#_Toc348561233)

[1.2.3. User Interface Design 9](#_Toc348561234)

[1.2.3.1. User Interface Layout Designs 10](#_Toc348561235)

[1.2.3.1.1. Menu Bar 10](#_Toc348561236)

[1.2.3.1.2. Cash Pile 10](#_Toc348561237)

[1.2.3.1.3. Banner 10](#_Toc348561238)

[1.2.3.1.4. Create Account 11](#_Toc348561239)

[1.2.3.1.5. Login 12](#_Toc348561240)

[1.2.3.1.6. Homepage 13](#_Toc348561241)

[1.2.3.1.7. Friend’s Page 14](#_Toc348561242)

[1.2.3.1.8. Battle Requests 15](#_Toc348561243)

[1.2.3.1.9. Friend Requests 16](#_Toc348561244)

[1.2.3.1.10. Breed Options 17](#_Toc348561245)

[1.2.3.1.11. Selling Options 18](#_Toc348561246)

[1.2.3.1.12. Battle Report 19](#_Toc348561247)

[1.2.4. Risk Assessment Table 20](#_Toc348561248)

[1.2.5. Gantt Chart 23](#_Toc348561249)

[1.2.5.1. First Draft 23](#_Toc348561250)

[1.2.5.2. Final Draft 25](#_Toc348561251)

[1.3. References 27](#_Toc348561252)

[1.4. Document Change History 27](#_Toc348561253)

[2. Test Specification 28](#_Toc348561254)

[2.1. Introduction 29](#_Toc348561255)

[2.1.1. Purpose of this Document 29](#_Toc348561256)

[2.1.2. Scope 29](#_Toc348561257)

[2.1.3. Objectives 29](#_Toc348561258)

[2.2. General Approach to Testing 29](#_Toc348561259)

[2.3. Test Table 30](#_Toc348561260)

[2.4. References 30](#_Toc348561261)

[2.5. Document Change History 30](#_Toc348561262)

[3. Design Specification 31](#_Toc348561263)

[3.1. Introduction 32](#_Toc348561264)

[3.1.1. Purpose of this Document 32](#_Toc348561265)

[3.1.2. Scope 32](#_Toc348561266)

[3.1.3. Objectives 32](#_Toc348561267)

[3.2. Decomposition Description 32](#_Toc348561268)

[3.2.1. Application in System 32](#_Toc348561269)

[3.2.2. Significant Classes 33](#_Toc348561270)

[3.2.2.1. LoginController 33](#_Toc348561271)

[3.2.2.2. ViewMonsterList 33](#_Toc348561272)

[3.2.2.3. ViewFriends 33](#_Toc348561273)

[3.2.2.4. FriendRequest 33](#_Toc348561274)

[3.2.2.5. MonsterAction 33](#_Toc348561275)

[3.2.3. Requirement-Class Mapping 33](#_Toc348561276)

[3.3. Dependency Description 34](#_Toc348561277)

[3.3.1. Component Diagram 34](#_Toc348561278)

[3.4. Interface Description 34](#_Toc348561279)

[3.4.1. LoginController 34](#_Toc348561280)

[3.4.2. ViewMonsterList 35](#_Toc348561281)

[3.4.3. ViewFriends 36](#_Toc348561282)

[3.4.4. FriendRequest 36](#_Toc348561283)

[3.4.5. MonsterAction 37](#_Toc348561284)

[3.5. Detailed Design 38](#_Toc348561285)

[3.5.1. Sequence Diagram Description 38](#_Toc348561286)

[3.5.1.1. Battle Report 38](#_Toc348561287)

[3.5.1.2. Friend Request 38](#_Toc348561288)

[3.5.1.3. Breeding Offer 39](#_Toc348561289)

[3.5.1.4. Buy/Sell Request 39](#_Toc348561290)

[3.5.2. Pseudo-Code for Significant Algorithms 40](#_Toc348561291)

[3.5.2.1. Breeding 40](#_Toc348561292)

[3.5.2.2. Ageing 41](#_Toc348561293)

[3.5.2.3. Fighting 41](#_Toc348561294)

[3.6. References 41](#_Toc348561295)

[3.7. Document Change History 42](#_Toc348561296)

[4. End-Of-Project Report 43](#_Toc348561297)

[4.1. Management Summary 44](#_Toc348561298)

[4.2. Historical Account of Project 44](#_Toc348561299)

[4.3. Performance of Team 46](#_Toc348561300)

[4.3.1. Chris Savill – chs17 46](#_Toc348561301)

[4.3.2. Richard Gray – rig6 46](#_Toc348561302)

[4.3.4. Edward Davies – edd14 47](#_Toc348561303)

[4.3.5. Sam Morrison – sjm16 47](#_Toc348561304)

[4.3.6. Jacob Smith – jas32 47](#_Toc348561305)

[4.3.7. Ivan Cholakov – ivc 48](#_Toc348561306)

[4.3.8. Katherine Rose Farmer – krf 48](#_Toc348561307)

[4.3.9. Oliver Roe – olr1 48](#_Toc348561308)

[4.4. Critical Evaluation of Project 48](#_Toc348561309)

[5. Appendices 49](#_Toc348561310)

[5.1. Appendix A – Requirements 50](#_Toc348561311)

[5.1.1. Functional Requirements 50](#_Toc348561312)

[5.1.1.1. FR1 Server-based Authentication 50](#_Toc348561313)

[5.1.1.2. FR2 Server Friends List 50](#_Toc348561314)

[5.1.1.3. FR3 Server Monster List 50](#_Toc348561315)

[5.1.1.4. FR4 Server Monster Fights 50](#_Toc348561316)

[5.1.1.5. FR5 Server-server Communication 50](#_Toc348561317)

[5.1.1.6. FR6 Client Options 50](#_Toc348561318)

[5.1.1.7. FR7 Start-up of Software in Browser 50](#_Toc348561319)

[5.1.1.8. FR8 Game Display in Browser 51](#_Toc348561320)

[5.1.1.9. FR9 Friend Matching 51](#_Toc348561321)

[5.1.1.10. FR10 Fight Notifications 51](#_Toc348561322)

[5.1.1.11. FR11 Friends Rich List 51](#_Toc348561323)

[5.1.2. External Interface Requirements 51](#_Toc348561324)

[5.1.2.1. EIR1 Appearance of Interface 51](#_Toc348561325)

[5.1.3. Performance Requirements 51](#_Toc348561326)

[5.1.3.1. PR1 Response of Program to User Input 51](#_Toc348561327)

[5.1.3.2. PR2 Target Computer for System 51](#_Toc348561328)

[5.2. Appendix B – Test Report 52](#_Toc348561329)

[5.2.1. Appendix B1 – Test Table 52](#_Toc348561330)

[5.2.2. Appendix B2 – Failed Tests 60](#_Toc348561331)

[5.3. Appendix C – Sequence Diagrams 61](#_Toc348561332)

[5.3.1. Appendix C1 – Battle Report Seq. Diagram 61](#_Toc348561333)

[5.3.2. Appendix C2 – Friend Request Seq. Diagram 61](#_Toc348561334)

[5.3.3. Appendix C3 – Breeding Offer Seq. Diagram 62](#_Toc348561335)

[5.3.4. Appendix C4 – Buy/Sell Seq. Diagram 63](#_Toc348561336)

[5.4. Appendix D – Maintenance Manual 64](#_Toc348561337)

[5.4.1. Program Description 64](#_Toc348561338)

[5.4.2. Program Structure 64](#_Toc348561339)

[5.4.3. Algorithms 65](#_Toc348561340)

[Singleton Pattern 65](#_Toc348561341)

[5.4.4. Main Data Areas 65](#_Toc348561342)

[5.4.5. Files 65](#_Toc348561343)

[5.4.6. Interfaces 65](#_Toc348561344)

[5.4.7. Suggestions for Improvements 65](#_Toc348561345)

[5.4.8. Things to Watch When Making Changes 67](#_Toc348561346)

[5.4.9. Physical Limitations of Program 68](#_Toc348561347)

[5.4.10. Rebuilding and Testing 68](#_Toc348561348)

[5.5. Personal Reflective Reports 68](#_Toc348561349)

[5.5.1. Chris Savill – chs17 68](#_Toc348561350)

[5.5.2. Richard Gray – rig6 69](#_Toc348561351)

[5.5.3. Edward Davies – edd14 70](#_Toc348561352)

[5.5.4. Sam Morrison – sjm16 70](#_Toc348561353)

[5.5.5. Jacob Smith – jas32 71](#_Toc348561354)

[5.5.6. Ivan Cholakov – ivc 72](#_Toc348561355)

[5.5.7. Katherine Rose Farmer – krf 73](#_Toc348561356)

[5.5.8. Ollie Roe – olr1 73](#_Toc348561357)

# 1. Project Plan

**Author:** Chris Savill – chs17

**Config Ref:** SE.05.DS Group 5

**Date:** 2012/10/28

**Version:** 1.3

**Status:**

## 1.1. Introduction

### 1.1.1. Purpose of this Document

The purpose of this document is to describe how we plan on creating the web application from the design to the technology we plan to use.

### 1.1.2. Scope

This document specifies the user interface designs and what technology will be used.

### 1.1.3. Objectives

The main objective of this project plan is to show to the customer how we plan on creating the system. Showing the design of the UI will give the customer an idea of how Monster Mash will look. Describing what technology that will be used will give the customer an insight of the “back end” side of the application.

## 1.2. Overview of Proposed System

At first we were thinking of using Tomcat server application, because it supports JSP and Java Servlets and are lighter than every other server application. We decided to use Glassfish, as it is being used by all the other groups and is the only software supported by the university. Glassfish provides full Java EE support including JSP and Java Servlets, which we will be using and is easier to navigate through than alternative software.

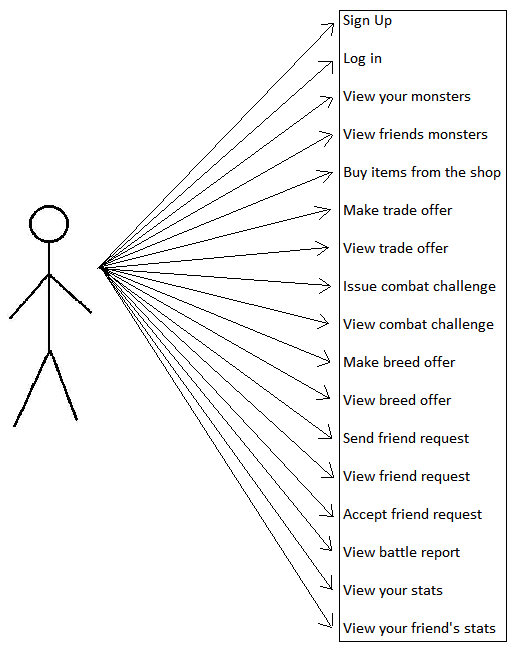
Glassfish is an open – source application server by Sun Microsystems. It provides full support for Java EE, JavaBeans, JPA, JSF etc. Glassfish has many more administration and monitoring tools than alternative software, such as Tomcat. We will be using Java Servlets to deal with requests from the client on our server. Considering we are striving for an MVC design pattern, Servlets are the right choice for control i.e handling requests.

A Java Servlet is a Java class used to extend the capabilities of the server. Although Servlets can respond to any types of requests, they are commonly used to extend the applications hosted by web servers, so they can be thought of as Java Applets that run on servers instead of in web browsers.

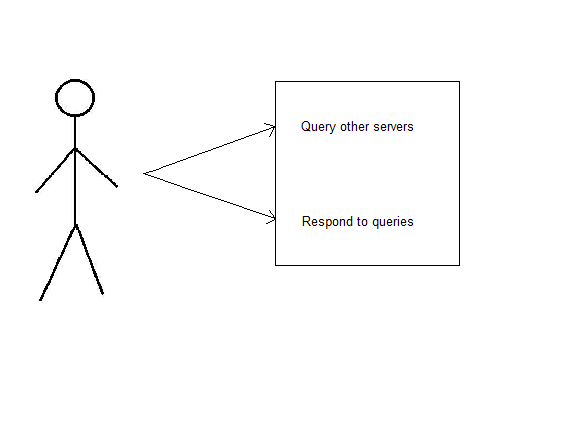
### 1.2.1. Technology Being Used

### 1.2.2. Use-cases

#### 1.2.2.1. Use-case for Users



#### 1.2.2.2. Use-case for Server



### 1.2.3. User Interface Design

To begin the user interface design, we started with a use case diagram which had everything the user needed. From this we created a directed graph which shows all the options the user has for navigation from that page. After agreeing on the pages we will be creating, we started with a basic hand drawn design of what the game will look like. Once we had produced the hand drawn designs, it was necessary to put them in to a digital format as well as annotated versions. There will be seven pages, which include:

* Create Account
* Login
* Homepage
* Friend's Page
* Battle Requests
* Friend Requests
* Breed Options
* Selling Options
* Battle Report

#### 1.2.3.1. User Interface Layout Designs

##### 1.2.3.1.1. Menu Bar

Each page (that the user is signed into (not sign up or log in page)) will have a menu bar. The menu bar will appear below the banner and will be on each page they visit. On this bar, the following will be displayed:

* **Home:** Links to the “Home” page.
* **Friend Requests:** Links to the “Friend Requests” page.
* **Breed Options:** Links to the “Breed Options” page.
* **Battle Requests:** Links to the “Battle Requests” page.
* **Selling Options:** Links to the “Selling Options” page.
* **User Name:** This will display the email of the user which is currently logged in.
* **Cash Pile:** This will display the user’s cash pile.
* **Logout:** This will end the user’s session.

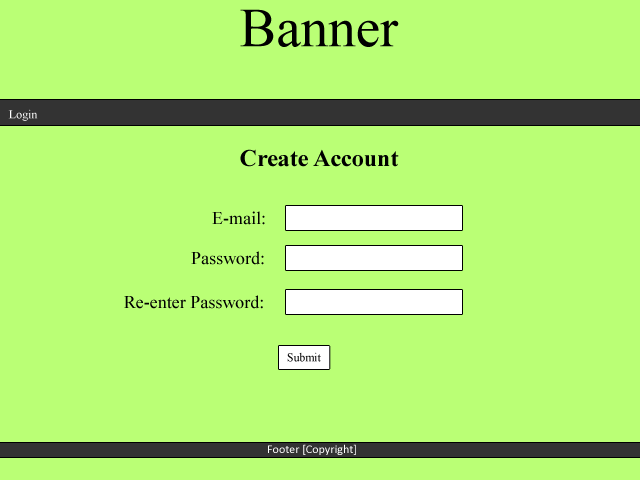
##### 1.2.3.1.2. Cash Pile

It is one of the requirements for the friends list to be ranked and this can be done by the highest amount of money. Each user will be sorted from richest – poorest on the homepage.

##### 1.2.3.1.3. Banner

Each page will have a banner which will be a design (text/image) saying 'Monster Mash'.

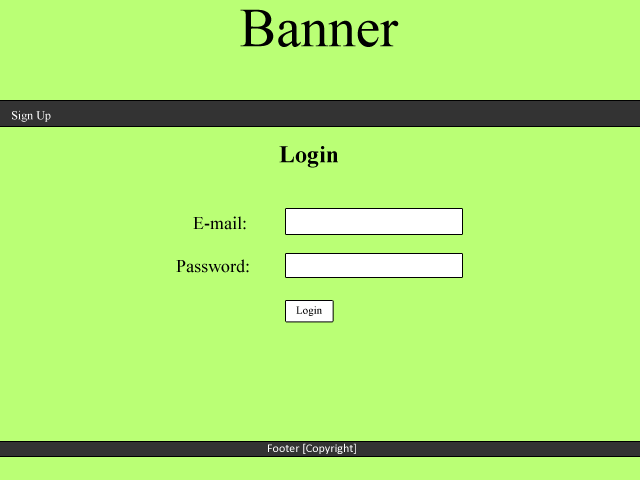
##### 1.2.3.1.4. Create Account



Once the user has entered their email and passwords, it will then be added to the database once the submit button is clicked.

This will direct the user to the “Login” page, to which they can log in.

##### 1.2.3.1.5. Login

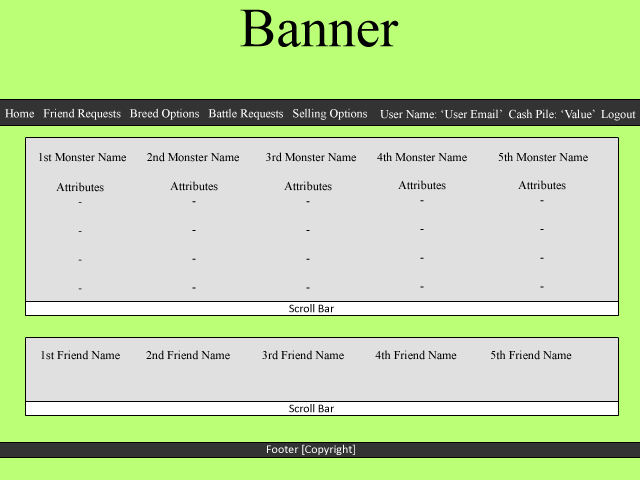


Non-registered users will click this to sign up. This will navigate the user to the “Create Account” page.

If the credentials are correct, they will be directed to their homepage. If the credentials do not match, an error message will appear.

The email and password that the user signed up with will be their credentials to login with. Regular expressions will be used for the email.

##### 1.2.3.1.6. Homepage

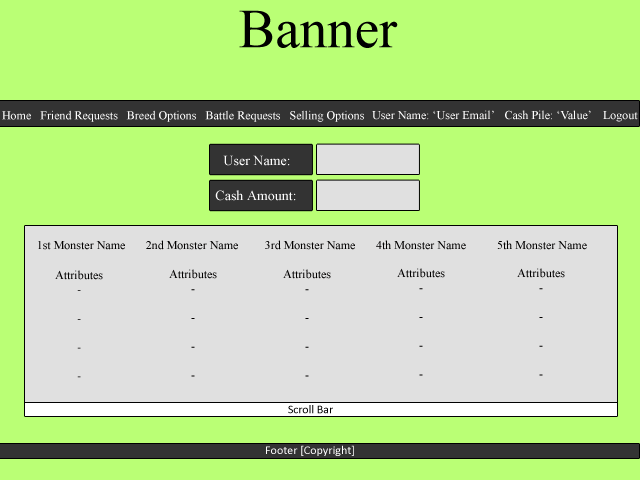


The user’s monsters will appear here and will have the monster’s name and attributes.

The user’s friends will appear here and will be sorted by wealthiest (by “cash pile”) to poorest. The user will be able to click on a friend which will direct them to the selected friend’s page.

For cases where the user has a lot of monsters/friends, a scroll bar is used so they can view the objects that don’t fit on the screen.

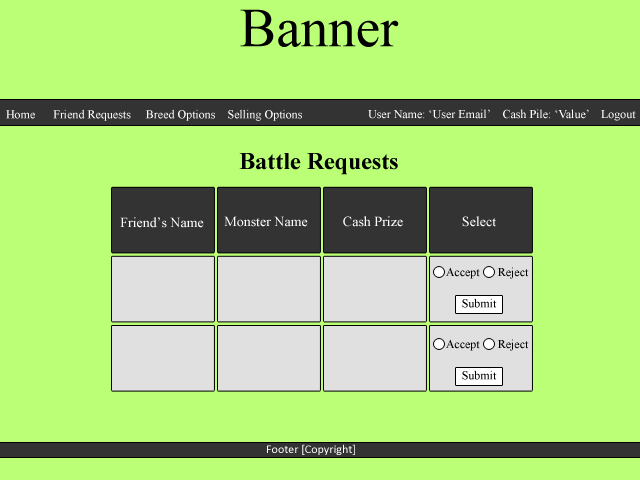
##### 1.2.3.1.7. Friend’s Page



The selected friend’s user name will show here as well as their cash amount.

The monster’s belonging to the friend will appear here with the name and attributes of that monster.

##### 1.2.3.1.8. Battle Requests



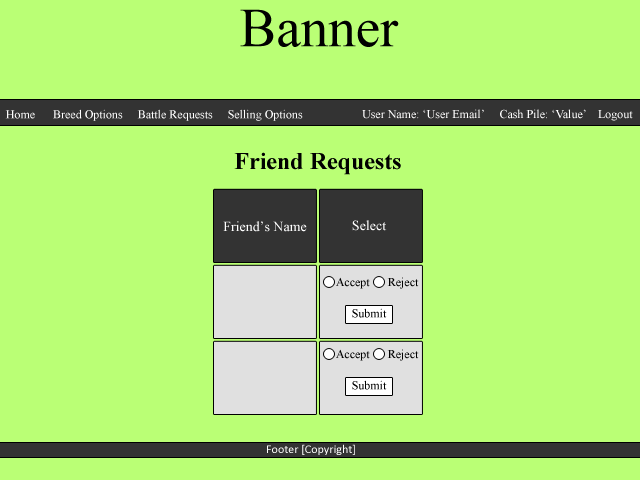
The user will have the option to choose to accept or reject the request.

The name of the friend requesting for battle will appear in this column.

The monster that the friend is requesting to battle with will appear in this column.

Each victor will receive prize money; which will appear in this column.

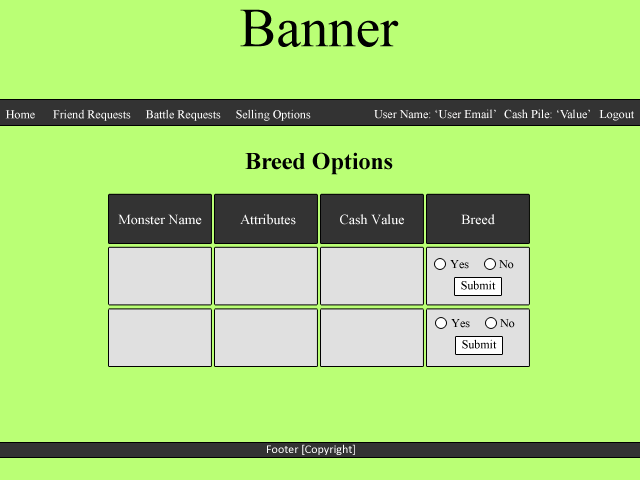
##### 1.2.3.1.9. Friend Requests



Each request to friend will appear here, with that friend’s name.

The user will be able to accept or reject the request to friend. Accepting will add that friend to their friends list.

##### 1.2.3.1.10. Breed Options



The name of the monster that is on offer to be bred will be shown in this column.

The cash value to breed with will appear here. The user will pay this value if they choose to breed with that monster.

The monster’s attributes will appear here.

The user will be able to choose what monster they want to breed with.

##### 1.2.3.1.11. Selling Options



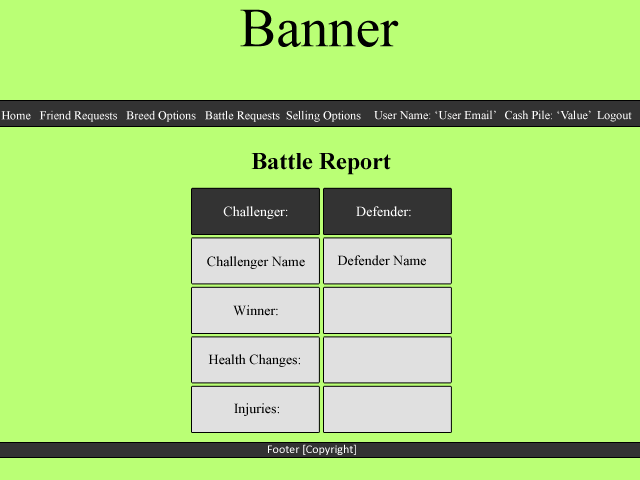
The user can add a cash value to the monster which will represent the sale price.

The user will be able to choose to sell the monster from this field.

The monster’s attributes will appear here.

The monster’s name will appear in this column.

##### 1.2.3.1.12. Battle Report



These headings represent which side the challenger and defender are.

The names of the contestants will appear here.

The winner of the battle will be shown here.

The remaining health of the monster will appear here.

Monster injuries from the battle will appear here.

### 1.2.4. Risk Assessment Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk** | **Risk Severity** | **What Will It Affect?** | **Measures Put In Place** | **Other Notes** |
| Illness | Medium-High | Project milestones and group centralisation. | Alert group as soon as possible so group Could act accordingly. |  |
| Natural Disaster | Low-High | Depending on the natural disaster, progress may be affected in different ways. e.g. flooding may change team members’ priorities. | Contact numbers distributed to group and try to a communication channel open with group to deal with problems. |  |
| Small coding error and unable to track source of error | Low | Ability to proceed with implementation.  Increased stress to | Use version management to roll back code to last working commit of code. |  |
| Deletion of local git repository | Low | Could affect progress of coding/ loss of code. | Commit updates frequently to avoid losing too much code.  Can re-clone online repository to local system. |  |
| Deletion of online git repository | Medium-High | Could lose whole project work. | Make sure each group member has an up-to-date local clone of the git repository to re-upload to the online repository/recreate a new one. | To delete the whole online git repository, the version management controller must delete it and manually and confirm deletion by entering the name of the repository to be deleted. |
| Code incompatibility | Low-High | Interaction of code between group members’ work could be hit and cause program-wide problems. | Make sure coders meet up frequently and work on code together along with QA manager.  Keep all work as centralised as possible by having frequent group meetings in which to do work/assign task at. | If work becomes decentralised, code incompatibility could become a big problem. |
| Server-server interaction problems | Medium-High | Servers may not be able to communicate with each other. | Make sure frequent meetings between allocated members from other groups are arranged to discuss server-server interaction protocols. | Keep the program as simple as possible but making sure that the program meets all of the requirements.  By keeping it as simple as possible, it is less likely that server-server interaction problems will occur. |
| Loss of project direction | Medium | Wrong tasks being allocated so wrong work is produced for delivery. | Frequent group meetings, checking requirements specification and appropriate documents to find check if the right goals are being worked towards at the right time. |  |
| Individual circumstances | Low-High | Could affect work motivation/priorities as well as group dynamic depending on situation. | Group supports each other appropriately having meetings to decide what to do if needed. Handle delicately. | May not become aware of individual’s circumstances straight away but this is expected. |
| Browser compatibility | Low | Client requires program to run on all installed browsers in the Delphinium and Solarium. | Keep interface with browser simple and validate to make sure it is compatible. |  |

### 1.2.5. Gantt Chart

In order to understand the roles from each member in the Gantt Chart, below is a key of the members along with their colours used within the chart.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Whole Team |  | Hand in | | | | |  | | |
| Ed |  | Holiday/Revision | | | | |  | | |
| Sam |  |  |  |  |  |  | |  |  | |  |
| Chris |  |  |  |  |  |  | |  |  | |  |
| Kit |  |  |  |  |  |  | |  |  | |  |
| Rich |  |  |  |  |  |  | |  |  | |  |
| Ivan |  |  |  |  |  |  | |  |  | |  |
| Jacob |  |  |  |  |  |  | |  |  | |  |
| Ollie |  |  |  |  |  |  | |  |  | |  |
| Main Coders |  |  |  |  |  |  | |  |  | |  |

#### 1.2.5.1. First Draft

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Oct.12 | | | | | | | | | | | Nov.12 | | | | | | | | | | | | Dec.12 | | | | | | | | | | | | | | | Jan.13 | | | | | | | | | | | | Feb.13 | | | | | | | | | |
| Task ID | Task Name | 08/10/2012 | | 15/10/2012 | | | 22/10/2012 | | | 29/10/2012 | | | 05/11/2012 | | | 12/11/2012 | | | 19/11/2012 | | | 26/11/2012 | | | 03/12/2012 | | | 10/12/2012 | | | 17/12/2012 | | | 24/12/2012 | | | 31/12/2012 | | | 07/01/2013 | | | 14/01/2013 | | | 21/01/2013 | | | 28/01/2013 | | | 04/02/2013 | | | 11/02/2013 | | | 18/02/2013 | | 25/02/2013 | |
| 1 | SE\_05\_DS\_01 Use Case Diagrams |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 2 | SE\_05\_DS\_02 Page Interaction Design |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 3 | SE\_05\_DS\_03 User Interface Designs |  |  | |  | | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|  | | |
| 4 | SE\_05\_DS\_04 Sequence Diagram for Monster Combat |  |  | |  | | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 5 | SE\_05\_PM\_01 Gantt Chart |  |  | |  | | | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 6 | SE\_05\_PM\_02 Risk Assessment |  |  | | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 7 | SE\_05\_RS\_01 Research Platforms |  |  | |  | | | | | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 9 | SE\_05\_QA\_02 Create Document for Hand-In |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|  | |
| 10 | SE\_05\_Interaction Hand-In |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 11 | SE\_05\_TEST\_01 Test Specification |  |  | |  |  | |  |  | |  | | | | | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 13 | SE\_05\_TEST Test Specification Hand-In |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 14 | SE\_05\_DS\_05 Class Diagrams |  |  | |  |  | |  |  | |  | | | | | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 16 | SE\_05\_DS Design Specification Hand-In |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 17 | SE\_05\_IMP\_01 Server Side Implementation |  |  | |  |  | |  |  | |  |  | |  |  | | | | | | | | | | | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 18 | SE\_05\_TEST\_03 Testing of Server Side |  |  | |  |  | |  |  | |  |  | |  |  | |  | | | | | | | | | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|  | | | | | | | | | |
| 19 | SE\_05\_IMP\_02 Client Side Implementation |  |  | |  |  | |  |  | |  |  | |  |  | | | | | | | | | | | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 20 | SE\_05\_TEST\_04 Testing of Client Side |  |  | |  |  | |  |  | |  |  | |  |  | |  | | | | | | | | | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|  | | | | | | | | | |
| 21 | SE\_05\_IMP\_03 Server to Server Implementation |  |  | |  |  | |  |  | |  |  | |  |  | | | | | | | | | | | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 22 | SE\_05\_TEST\_05 Testing of Server-Server |  |  | |  |  | |  |  | |  |  | |  |  | |  | | | | | | | | | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|  | | | | | | | | | |
| 24 | SE\_05\_IMP Implementation of Prototype |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | | | | | | | | | | | | | | | | | | | | |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 25 | SE\_05\_IMP Fix Bugs and Finish Implementation |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | | |  | |  |  | |  |  |  |  |  |  |
|
| 27 | SE\_05\_Software Delivery for Acceptance Testing |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|
| 28 | SE\_05\_Final Produce Final Document |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | | | | | | | | | | | | | | | | | | | | | | | | | |  |  | |  |  |  |  |  |  |
|  | | | | | | | | | | | | | | | | | | | | | | | | | |
| 30 | SE\_05\_Final Documentation Hand-In |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  |  |  |  |  |
|

#### 1.2.5.2. Final Draft

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Oct.12 | | | | | | | | Nov.12 | | | | | | | | Dec.12 | | | | | | | | | | Jan.13 | | | | | | | | Feb.13 | | | | | | | |
| Task ID | Task Name | 08/10/2012 | | 15/10/2012 | | 22/10/2012 | | 29/10/2012 | | 05/11/2012 | | 12/11/2012 | | 19/11/2012 | | 26/11/2012 | | 03/12/2012 | | 10/12/2012 | | 17/12/2012 | | 24/12/2012 | | 31/12/2012 | | 07/01/2013 | | 14/01/2013 | | 21/01/2013 | | 28/01/2013 | | 04/02/2013 | | 11/02/2013 | | 18/02/2013 | | 25/02/2013 | |
| 1 | SE\_05\_DS\_01 Use Case Diagrams |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 2 | SE\_05\_DS\_02 Page Interaction Design |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 3 | SE\_05\_DS\_03 User Interface Designs |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | |
| 4 | SE\_05\_DS\_04 Sequence Diagram for Monster Combat |  |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 5 | SE\_05\_PM\_01 Gantt Chart |  |  |  | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 6 | SE\_05\_PM\_02 Risk Assessment |  |  | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 7 | SE\_05\_RS\_01 Research Platforms |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 9 | SE\_05\_QA\_02 Create Document for Hand-In |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |
| 10 | SE\_05\_Interaction Hand-In |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 11 | SE\_05\_TEST\_01 Test Specification |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 13 | SE\_05\_TEST Test Specification Hand-In |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 14 | SE\_05\_DS\_05 Class Diagrams |  |  |  |  |  |  |  | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 16 | SE\_05\_DS Design Specification Hand-In |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 17 | SE\_05\_IMP\_01 Server Side Implementation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |
|
| 18 | SE\_05\_TEST\_03 Testing of Server Side |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 19 | SE\_05\_IMP\_02 Client Side Implementation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |
|
| 20 | SE\_05\_TEST\_04 Testing of Client Side |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 21 | SE\_05\_IMP\_03 Server to Server Implementation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |
|
| 22 | SE\_05\_TEST\_05 Testing of Server-Server |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 24 | SE\_05\_IMP Implementation of Prototype |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 25 | SE\_05\_IMP Fix Bugs and Finish Implementation |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 27 | SE\_05\_Software Delivery for Acceptance Testing |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 28 | SE\_05\_Final Produce Final Document |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | | |  |  |  |  |
|  | | | |
| 30 | SE\_05\_Final Documentation Hand-In |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|
| 31 | Weekly meeting |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

### 1.3. References

[1] *QA Document SE.QA.03 – General Documentation Standards. C. J. Price, N. W. Hardy. Release, 1.5.*

[2] *QA Document SE.QA.05B – Project Plan Specification Standards. B. P. Tiddeman. Final, 1.1.*

### 1.4. Document Change History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **CCF No.** | **Date** | **Changes Made to Document** | **Changed By** |
| 1. |  | 2012/10/26 | First Draft | Each member contributed |
| 1.1. |  | 2012/10/26 | General formatting (Table of Contents, Introduction) | Sjm16 |
| 1.2. |  | 2012/10/27 | Added Risk Assessment | Sjm16 |
| 1.3. |  | 2012/10/28 | Added a new use-case for server | Sjm16 |

# 2. Test Specification

**Author:** Katherine Rose Farmer – Krf, Oliver Roe – Olr1

**Config Ref:** SE.05.DS Group 5

**Date:** 2013/01/29

**Version:** 1.4

**Status:**

## 2.1. Introduction

### 2.1.1. Purpose of this Document

The purpose of this document is to highlight any possible tests which will be needed in order to meet the Functional Requirements [2].

### 2.1.2. Scope

This Test Specification document will show what tests will be needed before the actual testing of the code, and how we plan to test these functions. Additional tests may be added under the Software Delivery for Acceptance Testing task.

### 2.1.3. Objectives

The main objective of this document is to show all the tests which will cover each function in its functional requirement category. The tests will consist of a test reference, any inputs and outputs, what functional requirement category belongs with it, and the expected outcome. The test reference will be used in future testing.

## 2.2. General Approach to Testing

Our module/unit testing will be performed using JUnit tests for each class, designed and implemented by the programmer who is working on that class. This will be verified by the tester during the implementation phase as well as the integration testing phase of the project.

Our general system testing will take place in three different environments:

The first environment will be contained within our group and will use pre-generated test data, no real-time tests for time-based events and server interaction will take place between members of the group. The test data will be created based on the inputs required to fully test the functionality of the program.

The second environment will be contained within two to three groups and will use specific test data, some real-time tests for time-based events and server interaction will take place between members of the groups. The test data will be created based on the inputs required to fully test the functionality of the program.

The third environment will be contained within two to three groups, more if possible, will use specific test data and real-time tests and events will occur. Server interaction will take place between members of all groups. The test data will be created based on the inputs required to fully test the functionality of the program. This may occur during the Acceptance Testing phase.

## 2.3. Test Table

See Appendix B for Test Report which includes Test Table and Failed Tests table.

## 2.4. References

[1] *QA Document SE.QA.03 – General Documentation Standards. C. J. Price, N. W. Hardy. Release, 1.5.*

[2] *QA Document SE.CS.RS – Requirements Specification. B.P. Tiddeman. Second Draft, 1.1.*

[3] *QA Document SE.QA.06 –Test Procedure Standards. C. J. Price, N. W. Hardy. Release, 1.5.*

## 2.5. Document Change History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **CCF No.** | **Date** | **Changes Made to Document** | **Changed By** |
| 1. |  | 2012/11/08 | First Draft | Krf |
| 1.1. |  | 2012/11/15 | Changes to the test table | Olr1 |
| 1.2. |  | 2012/12/08 | Additional changes to the test table and adding of ‘General Approach to Testing’ | Olr1, Krf |
| 1.3. |  | 2013/01/28 | Additional changes to the test table | Krf |
| 1.4. |  | 2013/01/29 | Ensured that there were enough tests to cover all bases. | Chs17 |

# 3. Design Specification

**Author:** Katherine Rose Farmer – Krf, Richard Gray – Rig6, Ivan Cholakov – Ivc, Jacob Smith – Jas32, Oliver Roe – Olr1, Sam Morrison – Sjm16

**Config Ref:** SE.05.DS Group 5

**Date:** 2013/01/30

**Version:** 1.1

**Status:**

## 3.1. Introduction

### 3.1.1. Purpose of this Document

The purpose of this document is to outline what application we are using and give a description of each significant feature.

### 3.1.2. Scope

This document describes the application that has been chosen, significant classes and algorithms used along with a pseudo-code.

### 3.1.3. Objectives

The purpose of this document is to show the structure of the software to allow the user to understand any significant classes or algorithms with relevant diagrams to help give a clear visual of it. The customer will be able to gather what the application will have to offer and how it has been designed to meet the requirements.

Sequence diagrams are used to visually describe what a user can do when on a specific page.

## 3.2. Decomposition Description

### 3.2.1. Application in System

The application allows users from multiple servers to connect to each other. Each user will only be able to interact with another user if they are on each other’s friends list. Users will be able to accept friend requests which will enable them to then agree to: fight, breed and buy other user's monsters. Users will be able to send friend requests and change their monster's status regarding breeding, sale and battle for other users to select and continue the result of the action.

New users will require a registration with an email, which will be used as a username and as a personal identifier used for when other users wish to add them to their friend list.

A user can have more than one monster by either trading with other users or winning ownership of a monster after winning a battle.

In order to breed, a user must label their monster available for breeding. Another user will then see that monster is available to breed with and choose whether they want to accept this. Users wanting to breed will have to pay the owner the value of the monster they want to breed with, in return of a newly bred monster.

Each monster will have characteristics (Age, Health, Strength, Defence and Aggression). These will help judge the outcome of a fight with the calculation of monster damage (shown in 5.2.3. Fighting). The monster will eventually die of old age after a certain, predetermined, amount of time. The age of the monster will be determined by comparing the system date (and time) with the particular monster’s creation date (and time). As each day passes, the monster's health will decrease. Once the monster’s health reaches 0, it will be removed from the user's monster list.

After a fight has been arranged, it will take the user to the battle report page upon which it will show the winner, any health deductions (due to injuries) from monsters which took part, if a monster died in battle and prize money given to the winner. Each battle will be a fight to the death. The monster that reaches 0 Health during the battle will be classed as the losing participant and removed from the list of monsters. After the fight, Health will be deducted from the winning monster making it possible for the winner to die after the fight. The monster will have a value which will vary depending on the attributes that monster has. Each victorious user will receive the value of the opponent's monster in virtual money, which will be sent to their cash pile.

### 3.2.2. Significant Classes

#### 3.2.2.1. LoginController

This class authenticates users when logging onto (connecting) to the server. This will also authenticate users on other servers which are trying to connect to this server.

#### 3.2.2.2. ViewMonsterList

Each user will have a list of monsters. This class will allow them to view their, and their friend's monsters. The server will send requests and receive responses to view other user's monsters.

#### 3.2.2.3. ViewFriends

This class will contain the relevant friend details for the user to view. This will show all the friends the user has. Friends which have been accepted will be added to the list of friends.

#### 3.2.2.4. FriendRequest

In order to be friends, each request must be accepted. This class will deal with the requests and once accepted will add them to the friends list array for the user to see.

#### 3.2.2.5. MonsterAction

This class allows actions to be made that the user can do with the monster – breed, battle and buy.

### 3.2.3. Requirement-Class Mapping

|  |  |
| --- | --- |
| **Requirement** | **Classes providing requirement** |
| FR1 - Server-based authentication | LoginController |
| FR2 - Server friends list | ViewFriends |
| FR3 - Server monster list | ViewMonsterList |
| FR4 - Server monster mash management | MonsterAction, ViewFriends |
| FR5 - Server-server communication | MonsterAction, FriendRequest |
| FR6 - Client options | LoginController, ViewMonsterList, MonsterAction, FriendRequest |
| FR7 - Start-up of software in browser | LoginController |
| FR8 - Game display in browser | ViewMonsterList, ViewFriends, MonsterAction |
| FR9 - Friend matching | ViewFriends |
| FR10 - Fight notifications | ViewMonsterList |
| FR11 - Friends rich list | ViewFriends |

## 3.3. Dependency Description

### 3.3.1. Component Diagram



## 3.4. Interface Description

### 3.4.1. LoginController

This class extends HttpServlet so we can use it to send requests and receive responses from the relevant servers. It handles the operations relating to a user logging in.

**public** **class** LoginController **extends** HttpServlet{

/\*

\* Checks information from the database against the data entered by the user and

\* redirects to another page according to the outcome of the check.

\*/

**protected** **void** doGet (HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException{

}

/\*

\* Returns an error if data entered is not valid.

\*/

**protected** String loginError() {

}

/\*

\* Logs the user in to their account.

\*/

**protected** **void** login() {

}

}

### 3.4.2. ViewMonsterList

This class extends HttpServlet so we can use it to send requests and receive responses from the relevant servers.

**public** **class** ViewMonsterList **extends** HttpServlet {

/\*

\* This method retrieves the monster list for the user that has logged in and displays it

\* in the view.

\*/

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException{

}

}

### 3.4.3. ViewFriends

This class extends HttpServlet so we can use it to send requests and receive responses from the relevant servers.

**public** **class** ViewFriends **extends** HttpServlet {

/\*

\* This method retrieves the friend list for the user that has logged in and displays it

\* in the view.

\*/

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException{

}

}

### 3.4.4. FriendRequest

This class extends HttpServlet so we can use it to send requests and receive responses from the relevant servers. It deals with operations involving both sending friend requests and accepting friend requests.

**public** **class** FriendRequest {

/\*

\* This method sends the friend request to the relevant server.

\*/

**protected** **void** requestFriend(**enum** action, **int** groupID, String userID, String friendID) **throws** ServletException, IOException{

}

/\*

\* This method sends a key to the corresponding server and stores the new friend

\* in the friend list if the user decides to accept a friend request.

\*/

**protected** **void** acceptFriend(**enum** response, String userID, String friendID, **boolean** result) {

}

/\*

\* This method returns the address of the server required for a friend request.

\*/

**protected** String getServerAddress(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException{

}

}

### 3.4.5. MonsterAction

This class extends HttpServlet so we can use it to send requests and receive responses from the relevant servers. This deals with sending monster related requests and responses (breed, battle, buy). This also generates battle results.

**public** **class** MonsterAction **extends** HttpServlet {

/\*

\* This method sends a monster related request to the relevant server.

\*/

**protected** **void** monsterRequest(**int** groupID, String friendID, **int** friendMonsterID, String monsterName, **enum** action) {

}

/\*

\* This method responds to a monster related request, sending data to the relevant server.

\* It also generates a result for a battle if the request is for a battle.

\*/

**protected** **void** monsterResponse(**int** groupID, String ownerID, **int** monsterID, String friendID, **int** friendMonsterID, **boolean** response) {

}

## 3.5. Detailed Design

### 3.5.1. Sequence Diagram Description

#### 3.5.1.1. Battle Report

(See Appendix C1 for the sequence diagram matching this description.)

This diagram is used to show the dynamic interaction between the different objects included in creating a battle report. The *Home* participant refers to the *Home* page of the website. The diagram assumes that the user has already logged in and reached the *Home* page. The *Request Page* participant refers to the *Request* page which is linked from the *Home* page. The *Request* page contains battle requests, friend requests, trade requests and breeding requests. For this diagram we are only interested in the battle requests. The user can return to the *Home* page from the *Request* page. The *Battle Participant* represents battle taking place. The battle will not be shown but will be calculated using each monster's characteristics and a random value. The *Battle Report* participant represents the *Battle Report* page. This will contain various statistics, such as prize money, injuries etc.

#### 3.5.1.2. Friend Request

(See Appendix C2 for the sequence diagram matching this description.)

This diagram is used to show the dynamic interaction between the different objects used when executing a friend request. There are two actors and two objects involved in the friend request. The *User* is the actor which sends the request. The *Other User* is the actor which receives the request. The *User's Request Page* is where the *User* will send the request from. The *Other User's Request Page* is where the *Other User* will receive the request.

The User will go to their *Request Page* and create a request. The request will use the *Other User's* email address, which is used as a user name for the website as a unique identifier. The request will be sent to the *Other User*. It will appear on the *Other User's* *Request Page.* The *Other User* can choose to accept or decline the request. If the request is declined, the request is removed from their page. If the request is accepted, the *User* is added to The *Other User'*s friends list and the *Other User* is added to the *User*'s friends list.

#### 3.5.1.3. Breeding Offer

(See Appendix C3 for the sequence diagram matching this description.)

This diagram is used to show the dynamic interaction between the different objects used when executing a breeding request. There are two actors and two objects involved in the breeding request. The *User* represents the person placing one of their monsters up for breeding. The *User's Friend* represents a valid friend, another user of the game, who wishes to use the user's monster for breeding with one of their monsters. The *Breeding Options Page* is the place where the *User's* monster is set available to breeding. This process must go through an error check, possibly using JavaScript validation, which is represented by the *Error Check Object*. The *User's Friend's* monster list represents the store of monsters the *User's Friend* has.

The *User* accesses his *Breeding Options* page and chooses a monster and it's cash value. An error check then occurs to ensure that the cash value is valid before the monster is put up for breeding. If it fails the monster will not be put up for breeding and an error message displayed. Otherwise the monster will be put up for breeding. The *User's Friend* will be able to see that the monster is up for breeding, and can select it from the *User's* monster list. After selecting a monster, the *User's Friend* must select a monster from their list for the selected monster to breed with. The number of children is calculated based on an algorithm which is included in the *Significant Algorithm* section. These children are then added to the *User's Friend's* *Monster List.* The *User's Friend* gives money to the *User*, which is added to the *User's* cash pile. Throughout this time, the breeding monster never actually goes over to the *User's Friend*; only the monster's attributes are needed to perform the breeding algorithm.

#### 3.5.1.4. Buy/Sell Request

(See Appendix C4 for the sequence diagram matching this description.)

This diagram is used to show the dynamic interaction between the different objects used when executing a *Buy/Sell Offer*. There are two actors and two objects involved in the *Buy/Sell Offer*. The *User* represents the person placing their monster up for sale. The *User's Friend* represents a valid friend, another user of the game, who wishes to buy a monster from the User. The *Selling Offers* page is where the user can select a monster and place them up for sale. The *User's Friend's Monster List* is found on the *User's Friend's* *Home Page*.

The *User* will be able to move from their homepage to their *Selling Offer* page. Here they will be able to set the cash value of the monster and set it available to sell. If the cash value is not valid, the error check will return an error message and the monster will not be set available. The *User's Friend* will be able to see that the monster is up for sale and can select it from the *User's Monster List*. After the *User's Friend* has selected and bought the monster, it will be added to the *User's Friend's Monster List*. This will only occur after the *User's Friend* has paid the *Monster's* value into the *User's* cash pile.

### 3.5.2. Pseudo-Code for Significant Algorithms

#### 3.5.2.1. Breeding

1. Receive father as parameter (who to breed the female monster with).
2. Multiply the fertility rate of the mother and father monsters and then square root the result in order to generate the number of children bred.
3. Initialise array of monster children of a size equal to the number of children bred as provided by the step above.
4. While number of new added children < number of children bred:
   * Create a new monster.
   * Randomise whether monster is a male or female.
   * Initialise age rate by using crossover and mutation (see below for detailed explanation).
   * Initialise strength by using crossover and mutation (see below for detailed explanation).
   * Initialise evade by using crossover and mutation (see below for detailed explanation).
   * Initialise toughness by using crossover and mutation (see below for detailed explanation).
   * Initialise fertility by using crossover and mutation (see below for detailed explanation).
   * Initialise injury chance by using crossover and mutation (see below for detailed explanation).
   * Initialise date of birth with system date.
   * Initialise monster ID with an integer value.
   * Initialise owned ID with integer value of owner.
   * Add new monster to array of monster children.

5. Return array of monster children.

Where:

Crossover is the genetic crossover operator which has a 50% chance of returning the first argument and a 50% chance of returning the second argument.

Mutation is a random number between 0 and 1 inclusive which is described by a binomial distribution with an arbitrary number of trials and ½ probability for each trial.

#### 3.5.2.2. Ageing

Every time a monster is evaluated (e.g. every time the stable is viewed):

1. Compare the monster's date of birth with the system date.
2. Decrease the health of the monster based on the difference between its date of birth and the system date.
3. If after the health deduction the monster's health is <= 0 the monster will be classed as dead and deleted.

#### 3.5.2.3. Fighting

Once battle request has been accepted:

1. Add both monsters to a new array of battle monsters.
2. Make sure both monsters' details are up-to-date.
3. While there are still 2 monsters in the array of battle monsters (meaning both monsters are still alive):
   * Randomly select which monster attacks this go.
   * Calculate damage to other monster using:
     + Monster aggression \* monster strength \* (1 – defence of other monster) \* a random value.
   * Update injury to other monster with damage calculated above.
   * If the injury to the other monster is > the base health of the other monster:
     + The other monster dies and is removed/deleted from the array of battle monsters.
4. Return result of battle.

## 3.6. References

[1] *QA Document SE.QA.03 – General Documentation Standards. C. J. Price, N. W. Hardy. Release, 1.5.*

[2] *QA Document SE.QA.05A – Design Specification Standards. C. J. Price, N. W. Hardy. Release, 1.6.*

[3] *QA Document SE.QA.09 – Java Coding Standards. C. J. Price, A. McManus, N. W. Hardy. Release, 1.6.*

## 3.7. Document Change History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **CCF No.** | **Date** | **Changes Made to Document** | **Changed By** |
| 1. |  | 2012/12/07 | First Draft | Krf, Olr1, Rig6, Jas32, Ivc, Chs17, Sjm16 |
| 1.1. |  | 2013/01/30 | Updates to Breeding Offer and Buy/Sell sequence diagrams with descriptions | Krf |

# 4. End-Of-Project Report

## 4.1. Management Summary

The project team aimed to have a fully functional ‘Monster Mash’ game produced and tested by the deadline provided. Unfortunately this was not the case due to a few factors. However, before mentioning why the team did not manage to produce a fully working and tested product, it must be said that the whole team worked incredibly well together and supported each other where required. The problems did not lie within the team’s dynamics but within the initial plan and optimisms.

In terms of the overall functionality of the program, the entirety of the back-end code is present and ready to be integrated with the servlets/front-end. However, this is where we encountered difficulty. We underestimated the time needed to implement and integrate the servlets with the back-end code and its difficulty. Although we had most of the back-end including the JUnit tests to go with it quite early, we should have got producing the servlets and front-end earlier alongside the back-end and attempt to integrate throughout the process.

Also waiting on a standard for server-server communication held us back significantly and despite finding two other groups to pair up with to sort out a standard between us, it was still a bit too late to allow our coders to implement the system. When we realised that it would be too hard to get server-server functionality implemented we decided as a whole to focus on getting as much local functionality finished as possible and focus all our resources on that rather than splitting with the server-server implementations.

By the time acceptance testing came along we managed to get the game working with some integration with the servlets. The sign-up servlet allowing new user to create an account worked, along with the login servlet and the home page servlet. Users were able to create an account, login and view their home page (showing their monsters, initially starting with one) as well as viewing the battle requests, breeding options and selling options pages (however, the last three pages listed here were not fully integrated).

After realising that we might not be able to get a fully working game, we tried to shift resources to help aid the implementation however, although more got done, the actual integration of those parts completed by the other members still couldn’t be implemented in the time left. In the end we managed to get a lot of JavaScript for input validations completed but not integrated into the relevant servlets, the HTML integrated into the relevant servlets, but still missing the main integration with the back-end code.

This meant that other than having a user being able to sign-up, login and view some pages, nothing else could be really tested other than the user’s monsters experiencing their full life-cycles and dying (along with replacing the monster with a new one as they only had the one monster). The back-end code was however done along with the JUnit tests to prove the functionalities are there, they just couldn’t be integrated in time.

In terms of the documents, all documents were completed and in a good state, updated with all relevant feedback provided. The maintenance guide and test report within this document will be completed to the level we can with the program we managed to produce.

## 4.2. Historical Account of Project

The first thing we did as a group was allocate the team roles based on our abilities and preferences, although this was always subject to change as needed. We had no problems allocating roles although changes did occur shortly after, again with no problems. The first deadline was for the project plan so we got started straight away on that by listing out the requirements and thinking about what looked easy and hard as well as following the project plan structure provided by the relevant QA document.

We produced use case diagrams along with user interface designs for the time; although designs have been updated since implementation began as instructed. It was not hard coming up with the UI designs as we decided to follow the generic ‘facebook’ game layout, nice, simple and intuitive. The risk assessment table was also completed along with the Gantt chart and task allocations.

Once the project plan was submitted, we got started on the next deadline/deliverable, the test specification. We started by deciding how we would test the system, agreeing that JUnit testing the back-end as it was being implemented was a great approach along with integration testing as the system was to be put together. However, due to how the reality of things happened, integration testing went as far as it could with the state of the program at the end. When designing the tests we took the requirements specification and based the tests on the functional requirements trying to cover the obvious bases first then any not so obvious ones afterwards thus producing our testing table.

After submitting the test specification, the harder phase began; the design specification. This is when the team really had to start thinking as up to this point, the work load was not too bad. We started by discussing certain algorithms that would require some thinking such as how would the monster fights be processed? How would the breeding process occur and what would result? We looked at what was provided in the requirements specification and came up with pseudo code for the more significant algorithms of the program to aid with the design and implementation. Alongside that class diagrams were produced to show a more detailed view what classes would be built and what they would represent, what they depend on etc. Sequence diagrams were also produced to help show how the classes would interact in certain scenarios, these also helped us to visualise how to go about implementing certain algorithms being shown.

A very high level component diagram was produced to show how the HTML servlets simply interacted with the back-end and JSON servlets although not much was known about how these would fully interact at the time of design. At this stage of development, the design phase we hit our first stump, the establishment of server-server standards. This was a major hit to our progress as the coders were not comfortable designing a program that might have to be totally reworked to conform to any new standard established. No progress was really being made between groups so we decided to try and get together with another two three groups but unfortunately that came a little late and we ended up conforming with a bigger standard during implementation week, but even then couldn’t get to the server-server phase due to all the other stumps mentioned below.

The design specification was then submitted allowing us to start on producing the prototype for the demo. This is where things started to go wrong. We had one person, Ivan working/learning how to implement servlets and if we could go back, we probably would have had another doing the same as well as it proved harder than anticipated. We were not able to produce a working prototype but simply produced static HTML with CSS to show how the final product would look and how things would transition, not functionality at all. If we would have had some servlet functionality implemented by the time the prototype was due for demoing, the rest of the implementation and integration may have gone a lot smoother.

Christmas holidays came along and we did plan to have the coders work on things over the holiday, but it was then decided that exam stress was building and it would be best if everyone focused on their exams so that is what we did, we postponed working until after the exams, leaving the bulk till integration and testing week.

The way we structured/planned the build was to have Richard to generate the CSS and HTML for the servlets, Jacob to build the back-end (dealing with the serialising of the data/database and all the main functionalities) as well as tackle the server-server interaction from a back-end standpoint, and Ivan to build the servlets and take Richard’s CSS, HTML and JavaScript, along with Jacob’s back-end code and integrate everything together. Richard managed to get all of the CSS, HTML and most of the JavaScript (for validation purposes such as the sign-up validation) completed and Jacob managed to get most of the back-end implementation done and even tackled the server-server implementation. However, the servlet production and integration were the problem and Glassfish was also causing problems. As Glassfish only managed to work properly on Ivan’s laptop for some reason it made things harder.

## 4.3. Performance of Team

### 4.3.1. Chris Savill – chs17

My role as team leader involved keeping the team focused, motivated and working with good cohesion. I tried to keep the team centralised so our direction was always clear and most of the time we were fine. My main downfall was underestimating the difficulties of integration. If I could do the project again, I would assign an extra coder to help with the servlets and have me help with the coding as well. I don’t think I was proactive enough with the coding, although I helped with the design, the test design and the project plan, I should have put my coding skills to use rather than focusing purely on leadership.

As I leader I think I did well making sure to organise meetings, locations, topics to be covered, and running the meetings, however I should of got involved me with the coding. Also during integration and testing week I got ill and my blood condition prevented me from leading properly so I stayed bed bound one day so I could get well for acceptance testing, luckily Edward took charge, however as I leader I should have been present throughout the whole of integration and testing week. On the topic of being ill, I was ill many times throughout the project, which although I know couldn’t be helped; maybe the role of project leader wasn’t best suited for me due to the illness. My gauge of my performance is good as a leader by should of engaged with the coding more especially during coding week and planned for integration more.

### 4.3.2. Richard Gray – rig6

Richard was an invaluable member of our team, always making sure that he was doing something, contributing in some productive way. Richard even wanted to pursue helping Ivan with the servlet production but it was too late by the time we made the decision to assign Richard to helping Ivan. All of the program’s CSS, HTML and JavaScript were produced by Richard and to a good standard in my opinion.

A vital thing to mention about Richard was the amount of support he gave to people when they needed it, although to be honest, every team member supported each other in one way or another. Also Richard was never afraid to ask questions which really helped reduce setbacks as he never waited till the next meeting to ask questions that may have hindered his ability to finish the tasks assigned to him; he would just contact someone who could help straight away. Great performance all round.

### 4.3.4. Edward Davies – edd14

Edward was a key member in helping the team stay on track with our targets and especially good at pointing out things most of the team may have forgot or missed out. Not only he one of the main members to design how the system would look and work but he also was a great deputy project leader when I was not around due to being ill. Edward never hesitated to take over when I had to take leave due to my illness even if it was short notice and he never failed to produce some sort of weekly plan for the team in those events. He kept me up-to-date as well which was really helpful and helped me run our team meetings.

Both Edward and Sam were the two main members that focused on our documentation and they did a good job always making sure that we didn’t miss out on anything or clarifying topics as we went along. They both updated the documentation and designs when feedback was given and amended details as we implemented he actual program.

Edward helped me keep the team organised and has great organisation abilities, even questioning my decisions sometimes (good questions an reasons as well). Great performance all round.

### 4.3.5. Sam Morrison – sjm16

Sam was extremely proficient at keeping a record of the team’s minutes and actions as well as working with Edward on the documentation. He kept the documentations in the right format specified by the Quality Assurance documents and ensured that all documentation was produced to a high standard (also double checked by Ollie Roe the QA manager). It was very easy to give Sam a task and just leave him to get to it, if he had any problem he would find someone to help him straight away, he worked non-stop over integration and testing week, revamping our documentation with Edward and was quite independent. Great performance all round.

### 4.3.6. Jacob Smith – jas32

Jacob was one of biggest contributors, if anything the biggest contributor to the team. He is an excellent coder, very independent, and driven to get the tasks assigned to him done. Jacob was in charge of coding the back-end database and main functionalities to be integrated with the front-end servlets as well as producing the relevant JUnit tests to go with his code. Jacob event tackled the server-server interaction but as time grew close to acceptance testing it was apparent we would not get the product up to that level so we left that implementation to what was there and focused our resources on the integration.

Jacob was the member that suggested using serialisation as a form of managing a ‘database’ instead of using MySQL. At first the team did not understand the reasons behind that decision but once Jacob had implemented his solution and it worked, there was no reason to change the scheme and it proved to cause no problems anywhere else in the system. Great performance all round.

### 4.3.7. Ivan Cholakov – ivc

Ivan along with Richard and Jacob was one of three main coders and was in charge of developing the servlets for the system as well as integrating the whole system together. To be honest the integration lying solely on any one person’s shoulders would be too much now it is apparent how hard the integration of different components of systems can be. However, Ivan did extremely well and was very driven to get as much done as possible even though he kept hitting obstacles in his path.

Ivan was also a very independent coder, I could give him a task and he would be of doing it no questions asked (unless there was a problem) and would come in during integration d nesting week the next day and just pick up from where he left off. Ivan wanted to tackle the servlets by himself and he did well, but if we had a second chance, we would have assigned Richard or someone else such as me to aid him with the servlets. We just all underestimated the difficulties of integration. Overall great performance all round.

### 4.3.8. Katherine Rose Farmer – krf

Katherine was the team’s main tester, involved in designing the tests of the system, keeping an eye on the testing throughout the implementation as well as being part of the design decisions for the system. Katherine played a key role with me in supporting wherever we were needed as well such as when integration of the servlets came along. Katherine was very good at evaluating Jacob’s JUnit tests and was very hands on during coding week. Great performance all round.

### 4.3.9. Oliver Roe – olr1

As QA manager Oliver was in charge of making sure everyone stuck to standards and all of our documents and programs were produced correctly. Oliver was also one of the team members that kept asking questions to make sure that we were not missing anything. Oliver was the member that mainly kept the team on track, if we diverged from the main focus, he would remind the team of our current position and focus. A lot of quality assurance checking was needed for the project and both Sam and Oliver worked well together to split the tasks, with Sam managing document formatting and skimming over documents where Oliver would deal with the thorough checking and critical evaluating of the documents. Great performance all round.

## 4.4. Critical Evaluation of Project

# 5. Appendices

## 5.1. Appendix A – Requirements

### 5.1.1. Functional Requirements

#### 5.1.1.1. FR1 Server-based Authentication

The server will be used to authenticate a user, allowing them to log-in or register from their browser.

#### 5.1.1.2. FR2 Server Friends List

The server will maintain a list of friends for each user. Users will only be able to interact directly with their friends. Friends will be identified by their email address and added by a request-confirm mechanism.

#### 5.1.1.3. FR3 Server Monster List

The server will maintain a list of the monsters owned by each player and their attributes. These include genetic attributes and phenotypic attributes (such as age, health etc). The server will manage the monster lifecycle i.e. mating, birth, ageing, illness, injury and death. New users should be allocated a basic (random) monster and a small pot of virtual money.

#### 5.1.1.4. FR4 Server Monster Fights

The server will handle monster fights with a (virtual) cash prize available. The system will provide a fixed value prize to the winner. Users can select one of their monsters and challenge one of their friend’s monsters to a match. The friend can accept or decline the challenge. If they accept, the server will decide the winner based on the characteristics of the monsters along with an element of random chance (see Appendix A for an outline suggested algorithm). The server “pays” the winner the prize value and the loser’s monster should die.

#### 5.1.1.5. FR5 Server-server Communication

The server should be able to communicate with other servers using a standard protocol (agreed between groups) in order to play the game (add friends, buy/sell monsters, arrange monster breeding, manage fights, etc).

#### 5.1.1.6. FR6 Client Options

The client will allow users to interact with the system i.e. register/unregister, add/remove friends, offer for sale/buy monsters, offer for breeding / purchase breeding, etc. The sale and breeding of monsters will be managed in a similar way. If a user wishes to offer a monster for sale or breeding they can assign a value to the monster.

Any of their friends can view the monster’s price and purchase it or hire it for breeding. When purchasing the monster is transferred to the purchaser, when breeding the offspring are transferred to the purchaser. In both cases the sale price is transferred from the buyer to the seller. If the buyer does not have sufficient funds the transaction should not take place.

#### 5.1.1.7. FR7 Start-up of Software in Browser

When the software first starts, it will display a set of choices for the user as follows:

• Log in

• Create new account

Once logged in the system should provide an option to log-out. This will take the user back to the initial log-in/register screen.

#### 5.1.1.8. FR8 Game Display in Browser

When the player has logged in they should be able to see a list of their monsters (with status info), their friends (with offers of monsters for sale and for breeding), challenge requests (with prize money etc.) and have options to interact with these options as described in FR6.

#### 5.1.1.9. FR9 Friend Matching

The system should allow users to send a friend request to other users of the system (identified e.g. by their email) and to accept or reject requests sent to them. On accept the friend would be added to the friend list.

#### 5.1.1.10. FR10 Fight Notifications

Following a fight that the user has entered, the monster lists off all competitors should be updated. Loser’s monsters should be removed from their list, the winner will have the prize money added to his account and the monster’s status will be updated (accounting for injuries etc.).

#### 5.1.1.11. FR11 Friends Rich List

A user should be able to see a list of his friends (including himself) and the wealth of each, ordered by wealth.

### 5.1.2. External Interface Requirements

#### 5.1.2.1. EIR1 Appearance of Interface

The program should conform to usual look and feel guidelines for web-based applications.

### 5.1.3. Performance Requirements

#### 5.1.3.1. PR1 Response of Program to User Input

The user should feel like the system is responding to them at all times during game play. There should not be any perceptible lag between attempting a game action and the system responding.

#### 5.1.3.2. PR2 Target Computer for System

The client software produced should run correctly on standard browsers (i.e. one of the browsers installed on the IS desktop). The servers should also run either on the Department’s or University’s systems or a third-party system, but should be accessible from the department for testing.

## 5.2. Appendix B – Test Report

### 5.2.1. Appendix B1 – Test Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Reference** | **Functional Requirement** | **Test Content** | **Input** | **Output** | **Pass Criteria** |
| SE-TS-001 | FR1/FR6 | Check that if valid data is entered, log-in is possible. | Valid email address and password. | User's Home Page should be displayed. | After log-in details are entered and log-in button is selected, user will be directed to their home page. |
| SE-TS-002 | FR1/FR6 | Check that if invalid data is entered, log-in is not possible and error message shown. | Invalid data – either email address and/or password not used or data that does not resemble an email address. | General Error Message should be displayed i.e. “Invalid Username and/or Password”. | After invalid details are entered, and log-in button pressed, user will see an error message and no forward to home page will occur. |
| SE-TS-003 | FR6 | Check that if sign-up button is selected, user is directed to sign up page. | User selects sign-up button. | Sign-up page displayed. | After clicking sign-up button, the user should be directed to the sign up page. |
| SE-TS-004 | FR1 | Check that sign-up details are valid at registration. | User enters valid sign-up details and then tries to log in with the new user details. | User should to be directed to log-in page, then to home page after log-in. | If output is correct depending on what data has been entered, then test has passed. |
| SE-TS-005 | FR1/FR6 | Check that after SE-TS-004 has passed with valid data and valid sign up data has been added, that the user can now log-in. This tests whether the sign-up data has been processed properly. | Sign up data at registration. | User should be directed to their new home page. | After registration, user should be able to access the log-in page, log-in with their details and be re-directed to their home-page. |
| SE-TS-006 | FR2/FR8 | Check that a user's home page displays friends list with correct friends. | Initial test would use pre-generated test data, actual system test will use data obtained via interaction between users. | Friends list will be displayed on homepage with correct friends. | If the friends match up to the data within the server, then the friends list is being maintained correctly. |
| SE-TS-007 | FR3/FR8 | Check that user's home page displays a list of monsters with the correct monsters. | Initial test would use pre-generated test data, actual system test will use data obtained via interaction with the system and other users. | Monster list will be displayed on homepage with correct friends. | If the monsters match up with the data within the server, then the monster list is being maintained correctly. |
| SE-TS-008 | FR3 | Check that when user is first registered they are provided with a new basic monster and a small cash value. | User would sign up, then proceed to log-in. | At homepage, the monster list should display their new monster and their cash pile should display the amount given. | After log-in, the monster list and cash pile should be displayed with the required items. |
| SE-TS-009 | FR3 | Check that the server manages monster life-cycle, with regards to ageing then death of monsters. | The initial test will shorten the amount of time that ageing and death can occur in. The actual system test will take place in real-time using the algorithm contained within the program. | Monsters age, value and eventual death will be noted and displayed on web-page. | After a pre-determined amount of time, the monster will die. Between its creation and death the monster should also signs of ageing, such as change of age, value and characteristics. |
| SE-TS-010 | FR4 | Check that the monsters have a prize value assigned to them based on their characteristics. | Initial test would use pre-generated test data, actual system test will use data obtained via interaction with the system and other users. | Monster list will show monsters with their prize value. | The prize value of a monster should be equal to either the pre-generated values or determined correctly based on their characteristics. |
| SE-TS-011 | FR4/FR5/FR6 | Check that a user can select a monster and therefore issue a challenge with that monster against a friend. | Initial test would use pre-generated test data, actual system test will use data obtained via interaction with the system and other users. User should be able to select one of their own monsters, then select a friend and issue a challenge. | A challenge should appear in the friend's request list, with prize value of monster selected. | If the user is able to select a monster, issue a challenge to a friend on their friends list and that challenge appears on the selected friend's challenge list, this test has passed. |
| SE-TS-012 | FR4 | Check that after challenge is issued and accepted, the battle will take place and a winner will be decided. | Initial test would use pre-generated test data, actual system test will use data obtained via interaction with the system and other users. | A battle report page will be displayed showing the outcome of the battle. | A battle report page appears confirming a winner has been declared and it displays the chosen winner. If this occurs, the test is passed. |
| SE-TS-013 | FR4/FR10 | After battle, server should “pay” the winner the prize value of the losing monster. | Use the data of the monster used in  SE-TS-011 | Cash pile of winner should change. | If the cash pile increases by the prize value of the monster used, the test shall be passed. |
| SE-TS-014 | FR5/FR6/FR9 | Server should handle friend requests so users can see requests and accept or decline them. | User will enter email address of another user. Other user will be sent a friend request. | On accept, user's friend list should display the other user as a friend. | If the request is displayed in the other user's requests, it is accepted and the user is displayed, the test is passed. It is also passed if the user sends a request and upon reject the friend does not appear in the friend list. |
| SE-TS-015 | FR5/FR6 | User should be able to buy monsters from and sell monsters to their friends. | User will place a monster for sale and will attempt to buy a friend's monster which is displayed as being for sale. | Users will lose monster from list when sold, and gain monster from friend when bought. | If when the monster is bought it disappears from the seller's list and appears in buyer's list, then the test is passed. |
| SE-TS-016 | FR5/FR6 | User should be able to put a monster up for breeding and have another user choose that monster to buy for breeding. | User will place a monster up for breeding. One of the user's friends will try to buy that monster for breeding. | Users should keep both monsters. User who sold male would gain money equivalent to the sale. Other user would gain new baby monster. | If the users gain the appropriate items from the communication, the test is passed. |
| SE-TS-017 | FR6 | User should have the ability to unregister. | User will select the unregister button. Password confirmation could be required. | User should be removed from user data making them unable to log-in. | If user's details have been removed from the database and the user can no longer log-in, the test is passed. |
| SE-TS-018 | FR7 | Start-up page should give user the ability to register/ login. | The start-up page will be loaded. | The start-up page should be displayed. | If the start-up page contains area that would give the user the ability to register or log-in, the test is passed. |
| SE-TS-019 | FR7 | Throughout the site, the user should be able to logout and be returned to the start-up page. | On each page there should be an area which the user can select to logout. | User should be returned to the start-up page. | If the user selects the log-out link, the user should be logged out and the user should be returned to the start page. |
| SE-TS-020 | FR8 | Whilst using the game, the user should be able to see a friends list, a monster list and any requests and challenges they may have. | User will log-in to game. | The website. | If the website contains items that match with the requirements, then the test is passed. |
| SE-TS-021 | FR10 | After a battle, the loser's monster should be removed from their monster list. The winner's monster should be updated with any injuries that may have occurred. | After SE-TS-011, user and friend will check their monster lists. | Each of their monster lists should be updated. | If the lists have updated as required, the test is passed. |
| SE-TS-022 | FR11 | Friends list should be ordered by wealth and there should be a space that states where the user is within the list. | Initial test would use pre-generated test data, actual system test will use data obtained via interaction with the system and other users. | Friends list should be displayed in order. | If the order of the friends list is correct to the order the test data requires, the test is passed. |
| SE-TS-023 | PR1 | User should not be able to detect any noticeable lag while using the website. | User will browse website and interact with the website. | Outcomes of actions should be displayed as required. | If there is no noticeable lag between the user interaction and the required outcome, the test is passed. |
| SE-TS-024 | PR2 | Website should run on all standard browsers. | Website will be run on all standard browsers i.e. Browsers which are found on all departmental machines. | Website. | If the website displays correctly and runs as expected, the test is passed. |
| SE-TS-025 | FR1 | Check that sign-up details are valid at registration. | User enters invalid sign-up details. | Invalid data is added, an error message should pop up. This error message should state the problem that has occurred. | If the error message appears correctly, then the test has passed. |
| SE-TS-026 | FR5/FR6 | Check that if an invalid cash value (negative figure, ‘0’, or non-numerical value) is entered for a monster when ‘apply’ is hit to put the monster up for selling; an error message pops up prompting the user and the selling application is rejected. | User enters in ‘0’, ‘-27’ and ‘45afs5’ over 3 separate tests into the selling value field for a monster when putting them up for sale. | Error message should pop up when ‘apply’ is hit to put the monster up for sale pompting the user that an invalid selling value has been entered and the monster will not be put up for sale. | If error message appears correctly, then tests have passed. |
| SE-TS-027 | FR5/FR6 | Check that only monsters that have been put up for selling are available to buy for other users. | User selects a friend who has both a monster that is available for buying and one not available for buying and is taken to their page with the list of monsters. | A list of monsters should appear on the friend’s page with one monster which has a ‘buy’ button next to it and one monster without the ‘buy’ button next to it. | If the right monsters that are set to ‘buy’ are available to buy and the others are correctly not, then test has passed. |
| SE-TS-028 | FR5/FR6 | Check that a monster can only be bought if the user has enough cash to buy the monster. | User tries to buy a monster they cannot afford. | Error message should pop up prompting the user that they cannot afford the monster and the transaction is rejected. | If error message appears correctly, then tests have passed. |
| SE-TS-029 | FR5/FR6 | Check that if an invalid cash value (negative figure, ‘0’, or non-numerical value) is entered for a monster when ‘apply’ is hit to put the monster up for breeding; an error message pops up prompting the user and the breeding application is rejected. | User enters in ‘0’, ‘-2’ and ‘7ha1’ over 3 separate tests into the breeding value field for a monster when putting them up for breeding. | Error message should pop up when ‘apply’ is hit to put the monster up for breeding prompting the user that an invalid breeding value has been entered and the monster will not be put up for breeding. | If error message appears correctly, then tests have passed. |
| SE-TS-030 | FR5/FR6 | Check that only monsters that have been put up for breeding are available to ‘rent’ for other users. | User selects a friend who has both a monster that is available for breeding and one not available for breeding and is taken to their page with the list of monsters. | A list of monsters should appear on the friend’s page with one monster which has a ‘breed’ button next to it and one monster without the ‘breed’ button next to it. | If the right monsters that are set to ‘breed’ are available to breed and the others are correctly not, then test has passed. |
| SE-TS-031 | FR5/FR6 | Check that a monster can only be ‘rented’ for breeding if the user has enough cash to ‘rent’ the monster for breeding. | User tries to ‘rent’ a monster for breeding they cannot afford. | Error message should pop up prompting the user that they cannot afford the monster and the transaction is rejected. | If error message appears correctly, then tests have passed. |

### 5.2.2. Appendix B2 – Failed Tests

## 5.3. Appendix C – Sequence Diagrams

### 5.3.1. Appendix C1 – Battle Report Seq. Diagram



### 5.3.2. Appendix C2 – Friend Request Seq. Diagram



### 5.3.3. Appendix C3 – Breeding Offer Seq. Diagram



### 5.3.4. Appendix C4 – Buy/Sell Seq. Diagram



## 5.4. Appendix D – Maintenance Manual

### 5.4.1. Program Description

This is a browser based game about fighting monsters, with the aim to educate people on the evolution of breeding. Each player has their own monster(s) that they can use to fight, breed or sell with other users. Each monster has some generic attributes these are: strength, toughness and evasion. It accomplishes this by using a web based user interface that uses a supporting server program to store user information and to handle the server to server interaction.

An OO/Thread-safe Java subsystem for the storage of data, both in memory and disk, without resorting to an off the shelf database solution.

The piece of software we were supposed to produce was web based social game. The main functionality of the program is supposed to allow the users to register, sign in/log in, add/remove friends, fight, breed sell monsters, create new monsters. IVAN

### 5.4.2. Program Structure

[UML Diagram goes here]

The product consists of three main parts – server application (Description by Jacob), Database (Description by Jacob) and visual interface in the form of dynamically interactive websites. The visual interface is written in Java using the servlet and jsp frameworks. It is direct interaction with the server applications capabilities. IVAN

The DataSingleton class provides an entry point that allows access to persistent data and neatly encapsulates the entire sub-system. It uses a Singleton design pattern to, on first access, initialise a global TableOfAccounts by reading data from disk (more on this later), into memory and returning a handle to this global. A task is scheduled so that every 60 seconds the global TableOfAccounts, as it is in memory, is written to disk.  
  
Serialisation is performed by traversing the TableOfAccounts and constructing a json expression describing each user and any monsters they own. The list of these expressions is attached to the ‘users’ field in a root object that is written to disk under .monstermash\_persistence\_data.json in the user’s home directory. Deserialisation performs this task in reverse, expanding the json expression stored in .monstermash\_persistence\_data.json into its component sub json-expressions and populating the TableOfAccounts. Any relationships between objects, that cannot be expressed in a hierachical fashion (ie has-a) are stored by writing an integer that identifies the target object.  
  
Each user is represented by a UserAccount object, which contains Monster objects along with its own data. A log in request is handled by looking up a user by their email address in the global TableOfAccounts (obtained using DataSingleton) and then checking if their password matches a given string.  
  
User accounts can be looked up by either email address (user friendly name) or identifier number (guaranteed persistence) according to precise need. Email address is recommended.

### 5.4.3. Algorithms

### Singleton Pattern

def getSingleton():  
 global singleton  
 if singleton == None:  
 singleton = new object()  
 return singleton

### 5.4.4. Main Data Areas

### 5.4.5. Files

~/.monstermash\_persistence\_data.json is used to provide persistence. Read/write permissions are needed.

### 5.4.6. Interfaces

### 5.4.7. Suggestions for Improvements

As with most software, on reflection there are many features and processes that can be improved, such as:

Turn-based Combat – turn-based combat would allow for more user interaction with monster to monster combat. The user would be able to select between either a free, basic attack or a ‘special attack’ which would have a specific type (dependant on the monster’s type – see below). Any special attack would use up special points. A monster’s number of special points would be calculated based on their intelligence level. A stronger special attack would use up more special points. Any monster would only have a certain amount of special points that it could use per battle. Special points would recharge over time.

Attributes – by giving the monsters a larger variety of attributes the game would be more complex and so would appeal to more of users especially those that enjoy turn based, combat games. This is a list of attributes that could be used:

* Strength – the number of battles a monster can be in is dependent on how much strength the monster has. A monster’s strength will go down after each battle until it reaches 0 which means it will be unable to fight. This value will regenerate over time, maybe 1 point every 3 minutes.
* Attack – damage a monster’s basic attack does would be based on the monster’s attack and the opponent monster’s defence.
* Defence – damage a monster would take would be based on the monster’s defence and the attacking monster’s attack.
* Armour – armour would be the amount of damage a monster could either absorb from each attack in which case the value would be a percentage or the amount of damage a monster could absorb before taking damage in which case the value would be an integer.
* Intelligence – a monster would be able to use more powerful ‘special attacks’ based on the size of its intelligence.
* Speed – a higher speed would mean a monster could gain some advantage over any monster it was attacking or defending from and attack before the other monster.
* Type – every monster would have a specific type that would define which kinds of moves any specific monster could perform.

Shop – adding a shop to the game would enable users to purchase items that could be used to improve their monsters’ attributes, the monsters themselves or even in game currency:

* Health potions – used to increase the health of a monster in order to increase the chances of a successful win.
* Stim. Packs – used to increase the defence, intelligence or speed of a monster in order to increase the chances of a successful win.
* Weapons – used to increase the attack of a monster in order to increase the chances of a successful win. Equip-able items such as this could be damaged and eventually broken.
* Armour – used to increase the armour of a monster in order to increase the chances of a successful win. Equip-able items such as this could be damaged and eventually broken.
* Monsters – Users could be able to buy a new monster or even an additional monster for more fights or breeding. Monsters bought could either be basic monsters with basic attributes or could be ‘premium’ monsters (at extra cost) with more impressive attributes.

Monster evolution – by winning a fight each monster could earn a number of points that could be exchanged for health, weapon or amour points. This would make the monster more valuable and a better pedigree for breeding.

Graphics – by enhancing the graphics of the game in general it would make for a better user experience as a whole. Adding more interactive graphics or maybe adding animations in the place of images would be a definite improvement.

Gender/breeding – If every monster had a gender it would give the game a better way of educating people on how real-world evolution and natural selection works but on a small scale; the monsters can breed to create better monsters with higher attributes. It also makes the game more immersive and it gives players another way to play i.e. where some players may just want to fight, others may want to solely breed to make money or even just keep breeding to create a Mega-monster!

Mobile Friendly Gaming – the site could be edited in order to work as efficiently on mobile devices as on desktop computers and laptops. This would allow users to log onto and play the game at all times (assuming internet connection). This would improve general site traffic and therefore the general gaming experience.

Colour Blind Accessibility – people who are colour blind can only see certain colours, may merge text together when reading black text on white and may not be able to distinguish between some colours altogether (e.g. red and green). In order to make the site more accessible we would either need to add an extra feature to edit the site colours or to completely edit the site so it complies with these needs anyway.

The Remote sub-system was never implemented. .monstermash\_persistence\_data.json is not presently written at program exit due to integration issues with glassfish. There is a fair amount of dead code (ie commented out) produced during the latter stages of integration and testing week that need to be removed.

### 5.4.8. Things to Watch When Making Changes

Seeing as there are many different ways of coding the same thing and every programmer (though effort is taken to keep to the same coding standard) may programme the same thing differently. This could cause confusion between programmers if more than one programmer is working on the same piece of code and it is also possible that one programmer may delete something. It is therefore important that a good version control system is used to minimise the amount of code that is wrongly deleted.

If the software were to be written using a standard set of methods and those methods were change or discarded and not all the programmers were told, then the software (where it is vital that they integrate together with very little or no problems) would not work and would have to be ether re written or a new piece of code would have to be written like a bridge to the other codes.

Through using ‘Git’ programmers are able to all work on the software and able to have up to date copies of all the documents, however if the data is not committed, pushed or pulled correctly then problems can occur and it can be difficult to uncover the mistake and sometimes data can be lost and have to be re written.

The readJSON and buildJSON methods of any objects need to be updated to reflect the new schema.

### 5.4.9. Physical Limitations of Program

The data subsystem does not scale well. There is no option to distribute and load balance data. Serialising the data will cause latency in other requests to increase as a mutex on the entire data subsystem is obtained. Java’s garbage collection has similar behaviour.

### 5.4.10. Rebuilding and Testing

## 5.5. Personal Reflective Reports

### 5.5.1. Chris Savill – chs17

My role in the team was project team leader, my duties were to keep the team focused on the task at hand, plan and manage the team’s direction and resources and keep the team motivated, as well as dealing with reviews and problems within the team. I think I did well as a leader, especially in keeping the team motivated, however there are many areas where I let the team down.

Firstly I have to say that our team was fantastic, we worked really well together, everyone got along, there were no fights or real arguments (except over design, but that was bound to happen) and everyone supported each other and filled their roles well. The main reason our team didn’t manage to get a fully working and fully integrated product was down to me. The reason I say this is that it was my responsibility to make sure we had an accurate and reasonable project time plan/scale, and should have analysed how to implement the servlets earlier. I should have noticed after our demo that we needed a second person, either myself or someone else, to help Ivan implement and integrate the servlets. Although Ivan said he would be fine and wanted to take it on by himself, I as a leader should have realised that it may have been too much of a task for one person alone and assigned someone else to help regardless of his enthusiasm.

I also feel that I should have played a more proactive role in the team such as a coder. I could have brought more the team if I had contributed more to the coding, but instead I decided to focus more on managing. I have learned that to be a good leader and to manage, you can’t just look over everyone; you have to get involved and see what is really happening. Although I did help where needed and managed the team well, motivated them etc., I could have been an even better asset to the team as a coder and maybe the problems we encountered would have been mitigated better if I was coding as well as leading.

Another let down, especially as a general team member, is that I have been ill a lot. I have had to miss meetings that I had arranged and had other people catch me up or even do my tasks due to my illness. As a leader I see this as a big failure. How am I meant to lead a team properly if I am ill all the time? Even if I wasn’t ill enough to miss meetings, I was generally fatigued and less productive due to my blood condition. I know that the university and Computer Science department are aware of my on-going blood condition and continued treatment, but it doesn’t change the fact that my team may have suffered a lot due to it.

On the plus side, I always made sure that the group was on track and working on the right tasks so that the group stayed moving in one direction and didn’t diverge too much. Also, I made sure to keep the team morale up and always tried to organise and/or run the meetings.

On a final note I have to say that not one team member in my opinion did badly, they all did great, no-one let anyone else down (apart from me) and I wouldn’t change the group I had, just changed our approach to the project.

### 5.5.2. Richard Gray – rig6

I thought our group worked very well together and throughout most of the project there was good communication between team members. The project was managed very well by Chris considering it is the first major group project we have been assigned. Every time we had a meeting Chris had a clear agenda of the items that needed to be discussed and he ran the meetings confidently. I also thought he was very good at getting the team motivated and dealing with any problems that we came across. The work was divided up very well between the team members and pretty much all of the coding for the back end was completed well in advance by Jacob.

In the end it was decided that Jacob would deal with the backend, Ivan would deal with the servlets and I would write all the HTML, CSS and JavaScript to create the look of the website and handle client side validation of input fields. As well as writing the HTML code I was involved with some of the documentation such as writing the overview of how the system would work and the different classes involved with the servlets. I also did the use case diagram to show the interaction between the user and the system. By the end of coding week we had all of the functionality of the web application in place. The problems came with integrating the servlets with the backend. We kept getting HTTP 404 and HTTP 500 errors when trying to run the servlets. In hindsight we should have either had more people working on the servlets, which dealt with the main functionality of the application, or we should have got the servlets working much more in advance so we could have used coding week for final integration and testing rather than actual coding of the application.

It might have also been better to use a proper RDBMS for the database such as MySQL rather than storing data in files. This could have made integration easier and might have worked better with Glassfish, although the database we had was very robust, so if we had tested servlets earlier, we might have been able to debug the issues we had. I did complete the HTML, CSS and JavaScript in enough time and once that was done I tried to help out with the servlets, but because I didn't think I was going to be as involved with the servlet functionality I didn't learn enough about them in time to help with them effectively.

Overall, I think that we did well as a team and everyone contributed to the work load. The fact that we didn't complete all of the requirements of the project was down to the team as a whole and there isn't any one individual that can be held responsible. I have learnt a lot from this project. I have learnt that it is wise to take time to decide on how you will implement a system and the importance of comparing the different technologies available to ensure that you select the most appropriate software or hardware for a system. I have realised that it is much easier to change the software you will use in the design phase of the project rather than going ahead with implementation without verifying all of the decisions made. If I were to do this project again I would have made sure that there were at least two people working on servlets way before integration week to get the functionality of the application working well.

### 5.5.3. Edward Davies – edd14

Even though we did not meet all of the project requirements, I feel we have worked very hard on what we have delivered and I am proud that we have delivered them without any infighting or bickering. Chris was an excellent team leader who knew exactly what he wanted done and when it needed doing. Chris led all the meetings in a clear and objective way, as deputy project leader I took control of any meetings if Chris was unable to attend or ill.

I feel that every member of the team preformed really well together with no fighting or arguing, and if ever anyone did not understand a certain aspect of the project, the whole team would help to make that aspect clear: an example would be when Jacob suggested we use serialisation instead of SQL; most of us were unsure why we should do this, as no one else to our knowledge was using this method, but after Jacob explained what it was he was suggesting, we all understood what he meant and decided to go with that idea. It’s this kind of team work that was present every time we met for a weekly meeting.

As far as the allocation of work goes, I feel that Jacob, as the group member with most coding experience, was maybe given quite a lot to do. In all fairness, he did deliver the code on time and it was very well written. Ivan was in charge of the servlets but I think we underestimated how much work that would involve and may have over-worked him slightly although again, however, he produced the code on time.

The only real problem was that by the Thursday of testing week we had hit a wall with the servlets and we were unsure about why they were not integrating with the Java code but, in the last hour of the day we had a breakthrough which gave the coders (Jacob, Ivan and Rich) a boost to get fixing other problems.

The documentation has been worked on by everyone especially by Ollie, Kit, Chris, Sam and I. I think it has been written to a very good standard.

The thing I have learnt most by doing this project is the importance of clarity and understanding between people and how it is vital to work as a team to accomplish tasks successfully. If we were to do this again I think I would try and help a lot more with the HMTL and/or maybe the servlet development.

### 5.5.4. Sam Morrison – sjm16

Each member of this group project participated with 100% effort. Although the end product of the game wasn’t as we had expected, nobody could fault the effort that was put in by each member.

Chris led the team with great example – positivity, commitment and desire on achieving a good mark for the project. During the beginning of the project, we planned meetings which had no outstanding absentees. Chris made sure the group knew exactly what was required from them and that each member understood the tasks given which was a key aspect to how he ran this group. During coding week, Chris had been suffering from an illness and required to take the day off; which Ed stepped in as deputy project leader and ran the team with similar standards.

The coders (Rich, Jacob and Ivan) played a huge part of this project, mainly focusing on what they felt comfortable with respectively, but also carrying out additional research to ensure their work was substantial. They decided among themselves that Rich would deal with HTML, CSS and JavaScript with Jacob working on the backend of the program leaving Ivan solely working on the servlets. We realised, in hindsight, that Ivan’s task was a huge task for one person to accomplish on their own.

Kit was the main tester and supported the coders when she felt she could offer assistance. She also worked on sequence diagrams as well as the testing table with help from Ollie.

Being a Business and IT student, my role mainly consisted of documentation and making sure the final delivery of the project was kept to the QA standards, with help from Ollie, Ed, Kit and Chris. Ed and I designed the User Interfaces (with annotations) of how the game would look, which Rich was able to follow. Ollie helped with making sure each member’s work was of high standard and amending when needed.

This project was a successful experience, regardless of the outcome as the philosophy from each member was strong – nobody had any problems with each other, commitments was respected, and the team worked as a unit rather than as individuals.

### 5.5.5. Jacob Smith – jas32

Despite final delivery not being met, overall the project went well and it was unfortunate that every unimplemented feature (and the associated QA), with the exception of the server-server code, was just ‘one more hour’ from delivery.

As technical lead I should have paid more attention to the progress and management of the Servlet development. This was due to the fact that when I came to attempt the Server-Server code (half-way through implementation and testing week) with the servlet framework we were using, I discovered just how difficult it was to work with and the issues Ivan and Rich had been working hard to fix.

I did not take any of the Web Development or Database modules in the first year as I am a dual-honours student and as such was not well equipped to manage the servlet development, leaving most of it to Ivan. I recognise, in hindsight, that I should have been more ‘hands on’ with the technology at a much earlier stage as this would have given us time to solve the technical problems we had with the framework.

Despite team morale hitting rock bottom Thursday afternoon (just before the Servlet framework began to work) there was no infighting or blame despite the immense stress that must have been felt.

Pair programming efforts by myself and Ivan got the servlet framework working just before closing Thursday - we were able to quickly integrate and smoke-test many of the features that were included in the build finalised the following morning. File IO and serialisation came that morning, just before I had to leave for lectures - leaving the team to smoke-test and lock in the build without me.

It is with great remorse that I am left no other conclusion but the failure of the project was, ultimately, my own fault. Had I performed better as technical lead, and led servlet development more strongly - instead of focusing on where my strengths lay in the data sub-system - the problems that plagued integration and testing week could have been cleared well in advance and the project may have been a total success.

### 5.5.6. Ivan Cholakov – ivc

In my opinion our group worked very well together and for the most part, the tasks were distributed evenly and sensibly. The project was managed by Chris Savill who did a very good job of motivating the group to work together and to strive for success. Every meeting had clearly defined goals that needed to be reached. Most of the work was well enough divided between members and the group leader did a good job dividing the workload so that none of the group members had too much work to do on their own. In my opinion an improvement to that approach would be to organize everyone who can code to work together.

In the end the tasks were divided as follows: Chris – group manager, Jacob – database and server application, myself doing the front end and functionality implementation, Richard with the prototypes of the webpages which involved HTML, CSS and JavaScript, Ollie was our QA manager, Kit was our main tester, she was responsible for the JUnit tests, Ed and Sam did an amazing job keeping the documentation as professional as it could be.

Although I had to deal with researching and implementing technologies that were new to us (Java EE Servlets and JSP), I was also involved in some of the documentation. For that purpose I had to team up with Richard and discuss how to write the overview of the system implementation. By the end of coding week we had most of the functionality in the backend (server application) and some of the functionality implemented in the frontend. We, as a team, supported each other and everyone was willing to help anyone as much as they could. I think that everyone worked really hard for this project even if it is unfinished.

One major problem we encountered was that Jacob's approach to building the server side of the project needed some modifications due to the fact that we used Glassfish as our server container. We found this out pretty late in development so we could not fix the problem entirely by the end of the project. The problem was that Glassfish didn't allow reading and writing to txt files in the way that we tackled the project. Also, I think I did not distribute my time properly to have enough to build the frontend on my own. I never did any web modules so I was behind from the moment I had this job assigned to me, but still I think we needed more coders to participate in the coding of the project.

In my opinion our group did well as a team and everyone contributed to the extent of their abilities. The group manager did his job almost perfectly, Jacob wrote an amazing server-side application, Rich made the design of our program look as professional as possible. All in all, I learned a lot from this project. For example I learned a lot about web development, team work on a major project and Java EE. I understand now the importance of good requirements and design specification documents and that it is easier to change the functionality of the software in the design phase rather than in the implementation phase. If I had to do the project again with the same resources I would have assigned all the people able to produce adequate code to work on the code rather than documentation. In my opinion the project was a success because we all got to experience working in a team and I am happy that the team worked together until the very end without clashes or bickering.

### 5.5.7. Katherine Rose Farmer – krf

From the beginning to the end, the amazing teamwork within out group never faltered. It was an amazing group to have the pleasure of working with and, in the end; no one person should take the blame for the project not reaching its full potential.

Chris was an impressive team leader, always ready to motivate and drive the team forward. All the meetings were led well by him or Ed if Chris was ill and unable to attend. They both had clear agendas of what that meeting was needed for and how it was going to be structured. The three main coders (Jacob, Ivan and Rich) worked extremely hard to get the program working. Jacob had the backend of the program completed in advance and made a move to work on the server to server interaction. Rich did the entirety of the HTML, CSS and JavaScript needed to make the website look and feel as good as possible. The final designs were amazing and the website looked professional. Ivan worked alone on the servlets, taking the time to learn how to implement them properly. This was however where we could have performed better. The servlets were a tremendous task for one person, which was something we have only now realized in hindsight. There is no doubt that if another member of the team had been assigned to servlets and assisted Ivan, the program would have been fully functional. I worked as main tester for the group and attempted to assist the coders as best as possible to insure that the final system testing would be as complete as possible. However I have no experience in servlets and was unable to help where it was most needed.

Ollie, Sam, Ed, myself and Chris have worked incredibly hard on the documentation, making it neat, professional and up to the standards of the QA documents. We have worked just as hard as the coders, even if our efforts may seem small in comparison to the task that was the program, and should be commended just as highly as the coders.

The most important thing I have learned from this is that no matter the hardships we face, it is better not to fall apart and bicker. I am extremely proud that at the end of this, we were still a solid team and did not blame one person for the limited functionality of the program. We shall learn from these mistakes, having been out first major project, and we shall grow because of them.

### 5.5.8. Ollie Roe – olr1

I have thoroughly enjoyed working with my group during the development of our project and feel that we worked very well as a team. Though we did not complete the task to its full extent, we stayed co-operative and well-spirited even in the darkest of hours.

With strong leadership from Chris and his deputy Ed, we succeeded in completing all of the deliverables asked for prior to implementation and testing week before each of their respective deadlines and with high standards.

Chris was strong, motivated and organised throughout the development of our project. He led the team very well through thick and thin and was always available to be asked questions about anything regarding the project. He didn’t make snap judgements; Chris discussed every decision with at least one other member of the group (preferably one relevant to the subject) and pulled conclusions from those discussions. He was a hard taskmaster, enforcing regular meetings with every member of the group to make sure we were all always working on the same page. In my opinion Chris did a very good job leading this project especially considering that he lost a lot of time over Christmas and in Implementation and Testing week due to illness.

Ed was a brilliant deputy to Chris, supporting him throughout the project’s development and filling in for him when he was not able to attend meetings etc. Ed made sure that everyone worked at a reasonable pace and kept a close eye on their working hours ensuring no one was over (or under) worked. He also helped out a lot with the documentation and produced various diagrams and our Gantt charts.

Our coders (Jacob, Rich and Ivan) chose to work on the parts of the project which they felt suited them best.

Jacob took on the back-end for the program which, in hindsight, was probably too big a workload for only one group member to handle even though he was able to complete fully functioning and well laid out code before his allotted deadline. Even after he had finished his part of the project, he was always on hand to help the other coders, explain anything regarding his code and help the testers create appropriate tests for his part of the code.

Rich wrote the HTML/CSS needed for the pages of our website as well as some JavaScript which was used for validation. He kept to the designs we had made in the design specification well, only making any unplanned changes after speaking to necessary group members. The look and feel of our game was solely down to him and they have turned out excellently.

Ivan took on the mammoth task of writing the servlets needed to make our back-end and front-end work together to make the game work how we wanted it. In hindsight, this may have been too big a task to assign to one person as, even though Ivan put a lot of work into completing the servlets on time and in a working state, he was not able to complete everything before the deadline. In my opinion it is through no fault of his own that he was unable to complete his work and maybe he would have been able to complete his work if we (the managers, Chris and I) had assigned him a smaller task.