**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](#_Toc347411286)

[1.1. Introduction 7](#_Toc347411287)

[1.1.1. Purpose of this Document 7](#_Toc347411288)

[1.1.2. Scope 7](#_Toc347411289)

[1.1.3. Objectives 7](#_Toc347411290)

[1.2. Overview of Proposed System 7](#_Toc347411291)

[1.2.1. Technology Being Used 8](#_Toc347411292)

[1.2.2. Use-cases 8](#_Toc347411293)

[1.2.2.1. Use-case for Users 8](#_Toc347411294)

[1.2.2.2. Use-case for Server 8](#_Toc347411295)

[1.2.3. User Interface Design 9](#_Toc347411296)

[1.2.3.1. User Interface Layout Designs 10](#_Toc347411297)

[1.2.3.1.1. Menu Bar 10](#_Toc347411298)

[1.2.3.1.2. Cash Pile 10](#_Toc347411299)

[1.2.3.1.3. Banner 10](#_Toc347411300)

[1.2.3.1.4. Create Account 11](#_Toc347411301)

[1.2.3.1.5. Login 12](#_Toc347411302)

[1.2.3.1.6. Homepage 13](#_Toc347411303)

[1.2.3.1.7. Friend’s Page 14](#_Toc347411304)

[1.2.3.1.8. Battle Requests 15](#_Toc347411305)

[1.2.3.1.9. Friend Requests 16](#_Toc347411306)

[1.2.3.1.10. Breed Options 17](#_Toc347411307)

[1.2.3.1.11. Selling Options 18](#_Toc347411308)

[1.2.3.1.12. Battle Report 19](#_Toc347411309)

[1.2.4. Risk Assessment Table 20](#_Toc347411310)

[1.2.5. Gantt Chart 22](#_Toc347411311)

[1.3. References 23](#_Toc347411312)

[1.4. Document Change History 23](#_Toc347411313)

[2. Test Specification 24](#_Toc347411314)

[2.1. Introduction 25](#_Toc347411315)

[2.1.1. Purpose of this Document 25](#_Toc347411316)

[2.1.2. Scope 25](#_Toc347411317)

[2.1.3. Objectives 25](#_Toc347411318)

[2.2. General Approach to Testing 25](#_Toc347411319)

[2.3. Test Table 26](#_Toc347411320)

[2.4. References 26](#_Toc347411321)

[2.5. Document Change History 26](#_Toc347411322)

[3. Design Specification 27](#_Toc347411323)

[3.1. Introduction 28](#_Toc347411324)

[3.1.1. Purpose of this Document 28](#_Toc347411325)

[3.1.2. Scope 28](#_Toc347411326)

[3.1.3. Objectives 28](#_Toc347411327)

[3.2. Decomposition Description 28](#_Toc347411328)

[3.2.1. Application in System 28](#_Toc347411329)

[3.2.2. Significant Classes 29](#_Toc347411330)

[3.2.2.1. LoginController 29](#_Toc347411331)

[3.2.2.2. ViewMonsterList 29](#_Toc347411332)

[3.2.2.3. ViewFriends 29](#_Toc347411333)

[3.2.2.4. FriendRequest 29](#_Toc347411334)

[3.2.2.5. MonsterAction 29](#_Toc347411335)

[3.2.3. Requirement-Class Mapping 29](#_Toc347411336)

[3.3. Dependency Description 30](#_Toc347411337)

[3.3.1. Component Diagram 30](#_Toc347411338)

[3.4. Interface Description 30](#_Toc347411339)

[3.4.1. LoginController 30](#_Toc347411340)

[3.4.2. ViewMonsterList 31](#_Toc347411341)

[3.4.3. ViewFriends 32](#_Toc347411342)

[3.4.4. FriendRequest 32](#_Toc347411343)

[3.4.5. MonsterAction 33](#_Toc347411344)

[3.5. Detailed Design 34](#_Toc347411345)

[3.5.1. Sequence Diagram Description 34](#_Toc347411346)

[3.5.1.1. Battle Report 34](#_Toc347411347)

[3.5.1.2. Friend Request 34](#_Toc347411348)

[3.5.1.3. Breeding Offer 35](#_Toc347411349)

[3.5.1.4. Buy/Sell Request 35](#_Toc347411350)

[3.5.2. Pseudo-Code for Significant Algorithms 36](#_Toc347411351)

[3.5.2.1. Breeding 36](#_Toc347411352)

[3.5.2.2. Ageing 37](#_Toc347411353)

[3.5.2.3. Fighting 37](#_Toc347411354)

[3.6. References 37](#_Toc347411355)

[3.7. Document Change History 38](#_Toc347411356)

[4. End-Of-Project Report 39](#_Toc347411357)

[4.1. Management Summary 40](#_Toc347411358)

[4.2. Historical Account of Project 40](#_Toc347411359)

[4.3. Performance of Team 40](#_Toc347411360)

[4.4. Critical Evaluation of Team and Project 40](#_Toc347411361)

[5. Appendices 41](#_Toc347411362)

[5.1. Appendix A – Requirements 42](#_Toc347411363)

[5.1.1. Functional Requirements 42](#_Toc347411364)

[5.1.1.1. FR1 Server-based Authentication 42](#_Toc347411365)

[5.1.1.2. FR2 Server Friends List 42](#_Toc347411366)

[5.1.1.3. FR3 Server Monster List 42](#_Toc347411367)

[5.1.1.4. FR4 Server Monster Fights 42](#_Toc347411368)

[5.1.1.5. FR5 Server-server Communication 42](#_Toc347411369)

[5.1.1.6. FR6 Client Options 42](#_Toc347411370)

[5.1.1.7. FR7 Start-up of Software in Browser 42](#_Toc347411371)

[5.1.1.8. FR8 Game Display in Browser 43](#_Toc347411372)

[5.1.1.9. FR9 Friend Matching 43](#_Toc347411373)

[5.1.1.10. FR10 Fight Notifications 43](#_Toc347411374)

[5.1.1.11. FR11 Friends Rich List 43](#_Toc347411375)

[5.1.2. External Interface Requirements 43](#_Toc347411376)

[5.1.2.1. EIR1 Appearance of Interface 43](#_Toc347411377)

[5.1.3. Performance Requirements 43](#_Toc347411378)

[5.1.3.1. PR1 Response of Program to User Input 43](#_Toc347411379)

[5.1.3.2. PR2 Target Computer for System 43](#_Toc347411380)

[5.2. Appendix B – Test Report 44](#_Toc347411381)

[5.2.1. Appendix B1 – Test Table 44](#_Toc347411382)

[5.2.2. Appendix B2 – Failed Tests 52](#_Toc347411383)

[5.3. Appendix C – Sequence Diagrams 53](#_Toc347411384)

[5.3.1. Appendix C1 – Battle Report Seq. Diagram 53](#_Toc347411385)

[5.3.2. Appendix C2 – Friend Request Seq. Diagram 53](#_Toc347411386)

[5.3.3. Appendix C3 – Breeding Offer Seq. Diagram 54](#_Toc347411387)

[5.3.4. Appendix C4 – Buy/Sell Seq. Diagram 55](#_Toc347411388)

[5.4. Appendix D – Maintenance Manual 56](#_Toc347411389)

[5.4.1. Program Description 56](#_Toc347411390)

[5.4.2. Program Structure 56](#_Toc347411391)

[5.4.3. Algorithms 56](#_Toc347411392)

[5.4.4. Main Data Areas 56](#_Toc347411393)

[5.4.5. Files 56](#_Toc347411394)

[5.4.6. Interfaces 56](#_Toc347411395)

[5.4.7. Suggestions for Improvements 56](#_Toc347411396)

[5.4.8. Things to Watch When Making Changes 56](#_Toc347411397)

[5.4.9. Physical Limitations of Program 56](#_Toc347411398)

[5.4.10. Rebuilding and Testing 56](#_Toc347411399)

[5.5. Personal Reflective Reports 56](#_Toc347411400)

[5.5.1. Chris Savill – chs17 56](#_Toc347411401)

[5.5.2. Richard Gray – rig6 56](#_Toc347411402)

[5.5.3. Edward Davies – edd14 56](#_Toc347411403)

[5.5.4. Sam Morrison – sjm16 56](#_Toc347411404)

[5.5.5. Jacob Smith – jas32 56](#_Toc347411405)

[5.5.6. Ivan Cholakov – ivc 56](#_Toc347411406)

[5.5.7. Katherine Rose Farmer – krf 56](#_Toc347411407)

[5.5.8. Ollie Roe – olr1 56](#_Toc347411408)

# 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.

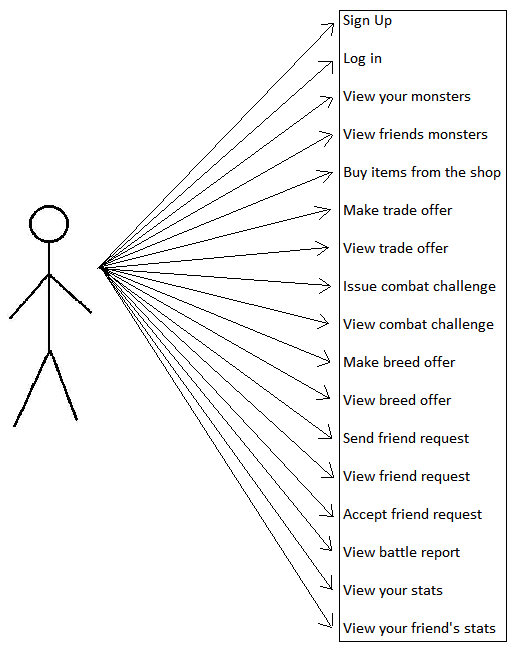
Glassfish is an open – source application server by Sun Microsystems. It provides full support for Java EE, JavaBeans, JPA, JSF etc. Glassfish has many more administration and monitoring tools than alternative software, such as Tomcat. We will be using Java Servlets to deal with requests from the client 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.

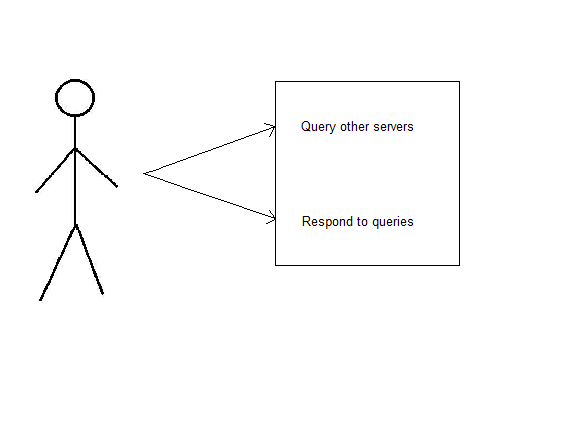
### 1.2.1. Technology Being Used

### 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.

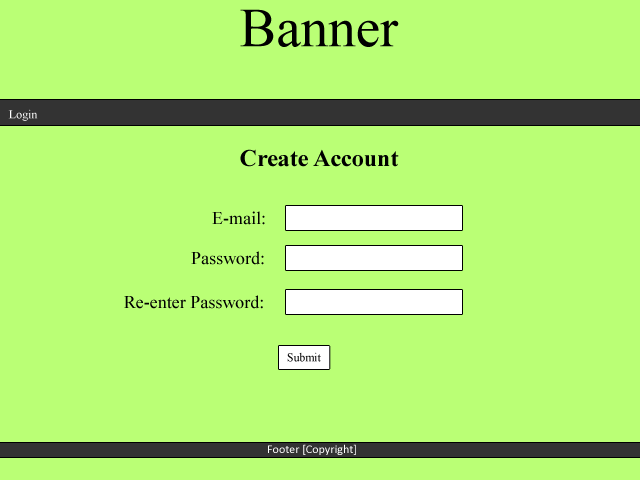
##### 1.2.3.1.2. Cash Pile

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'.

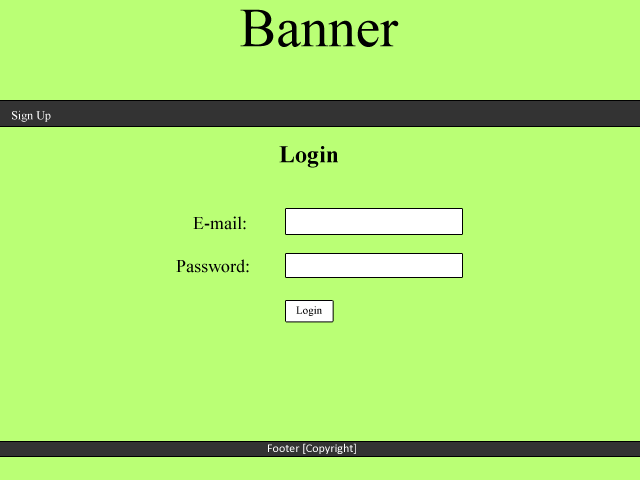
##### 1.2.3.1.4. Create Account



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

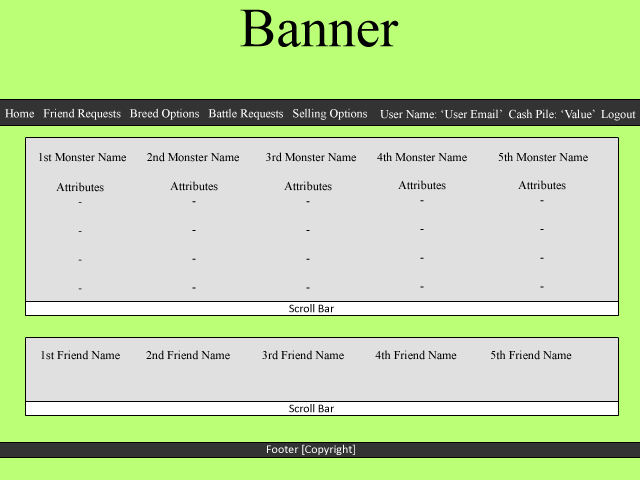


Non-registered users will click this to sign up. This will navigate the user to the “Create Account” page.

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

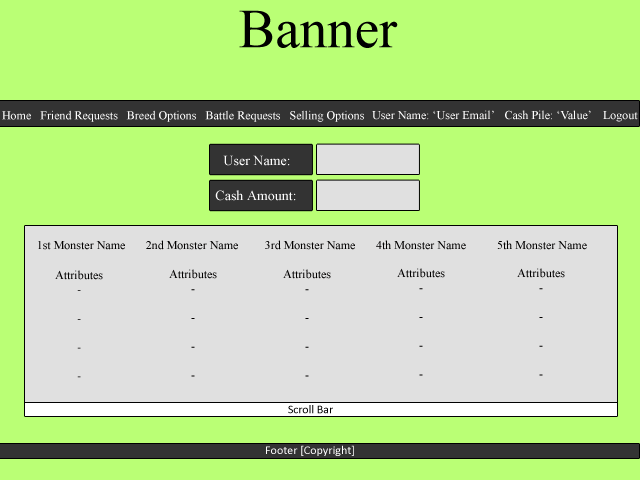


The user’s monsters will appear here and will have the monster’s name and attributes.

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.

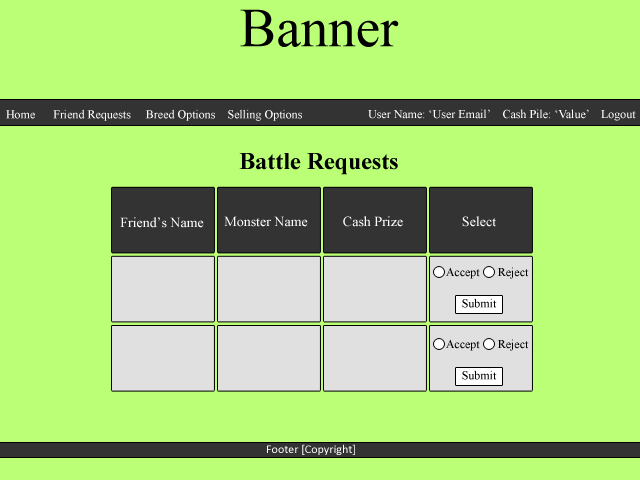
##### 1.2.3.1.7. Friend’s Page



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



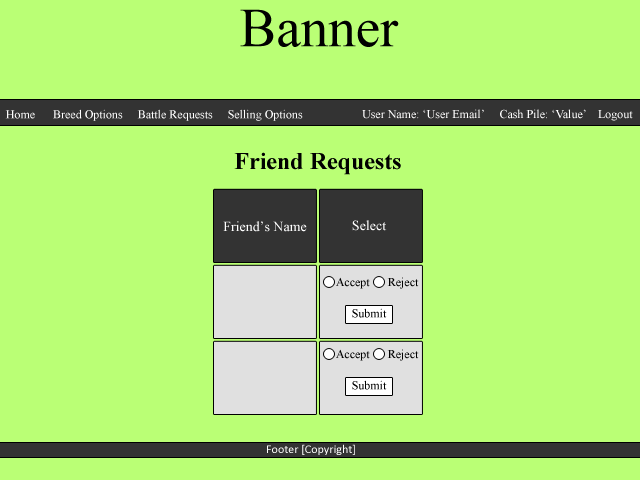
The user will have the option to choose to accept or reject the request.

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.

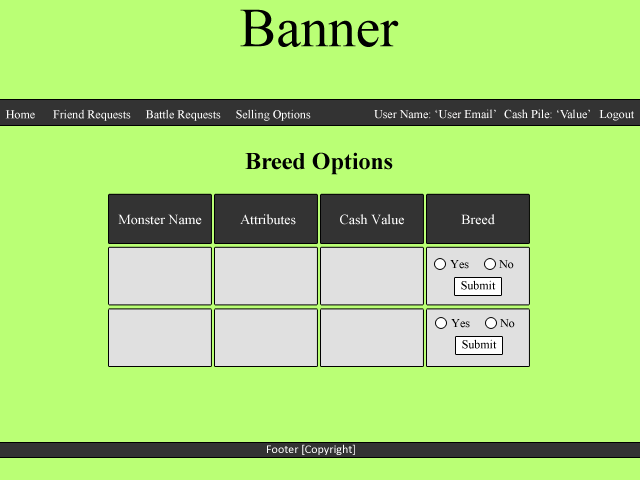
##### 1.2.3.1.9. Friend Requests



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



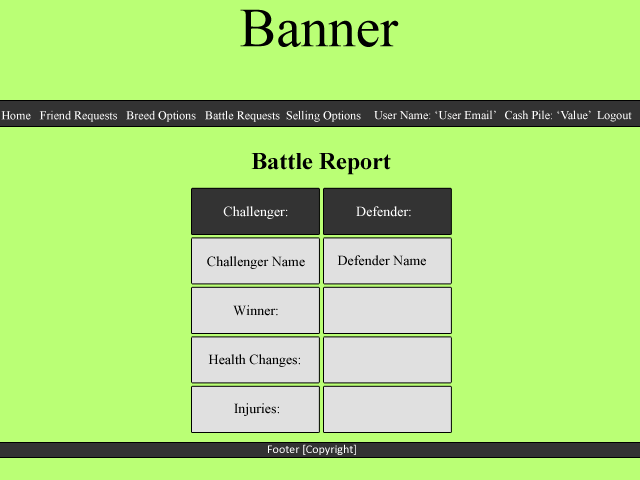
The user can add a cash value to the monster which will represent the sale price.

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

### 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

## 4.2. Historical Account of Project

## 4.3. Performance of Team

## 4.4. Critical Evaluation of Team and 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.

### 5.4.2. Program Structure

### 5.4.3. Algorithms

### 5.4.4. Main Data Areas

### 5.4.5. Files

### 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.

### 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.

### 5.4.9. Physical Limitations of Program

### 5.4.10. Rebuilding and Testing

## 5.5. Personal Reflective Reports

### 5.5.1. Chris Savill – chs17

### 5.5.2. Richard Gray – rig6

### 5.5.3. Edward Davies – edd14

### 5.5.4. Sam Morrison – sjm16

### 5.5.5. Jacob Smith – jas32

### 5.5.6. Ivan Cholakov – ivc

### 5.5.7. Katherine Rose Farmer – krf

### 5.5.8. Ollie Roe – olr1