Software Engineering Group Project

Final Report

|  |  |
| --- | --- |
| Author: | Group 12 |
| Config Ref: | SE\_G12\_FR\_00 |
| Date: | 2013-02-14 |
| Version: | 0.2 |
| Status: | Draft |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2008

CONTENTS

CONTENTS 2

1. Introduction 3

1.1 Purpose of this Document 3

1.2 Scope 3

1.3 Objectives 3

2. Management Summary 3

2.1 Program 3

2.2 Documents 3

2.3 Difficulties and their Resolutions 4

2.4 Team Performance 5

2.5 Historical Account 5

2.6 Final State of Project 6

2.7 Performance of each team member 6

2.8 Critical Evaluation of the team and the Project 6

2.9 Test Report 7

REFERENCES 18

DOCUMENT HISTORY 19

# Introduction

This is the final report for the group project.

## Purpose of this Document

The purpose of this document is to clearly state how much we as a group have accomplished on the project. It is also to enable the markers to evaluate our group on how well we have completed the project.

## Scope

This document aims to provide the markers they concluded summary of how our group project has been. Reflecting on how we worked as a team and how successful we are is shown in the final program and this report.

## Objectives

The objective of this document is to provide the markers with a summary of how the group project has been.

# Management Summary

## Program

## Documents

### Project Plan

The project plan document was successful on the hand in by making certain decisions and creating foundations for moving on in our design. By explaining the intended users and technologies, we were able to decide for ourselves what the best technologies were to use for our group project. We began with initial sketches of the user interface which then turned into the HTML generated design. This allowed us to have a starting point for when we began the implementation of Java and the server to server communication. Additionally, by generating a Gantt chart, we were able to set milestones and goals which allowed us to have good time management and be successful with our deadlines. By having successful Risk Analysis, we were able to predict possible risks to our group, thus allowing us to be prepared for the worst. Due to the Risk Analysis we were able to solve any problems that did arise, with little repercussions.

Alternatively, the Use Case diagram that was generated was a little confusing and poorly laid out. Thanks to feedback and our QA manager we were able to resolve the issue and create a more readable and understandable Use Case diagram.

### Test Specification

By generating many tests, which were created with the functional requirements in mind, we were able to look in depth at what was needed and what tests would be needed for these requirements. Whilst looking at the functional requirements we allocated different requirements to different members of our group, thus allowing for us to come together in a meeting and compile with the addition of possible alternatives and additions. With the 11 different requirements we were able as a group to create 69 tests for the program which looked at both client and server side testing with the addition of our JUnit tests. Eventually we had a solid document for the testing however when reading through as a group, we noticed that many of the functional requirements were very similar in terms of the tests we had generated. We decided to merge these into multiple requirement tests, meaning not to repeat the same test for different requirements, since the tests satisfied both.

### Design Specification

There is a lot of detail in our Design Specification document, and includes a lot of key information for when we began coding and began to bring everything more together. For example, the Decomposition Descriptions allowed us to look at the different client and server applets, and what each applet will be doing to interact cross server and with each other. Additionally the information of the significant classes, not only give an insight into how the program has been made, but also, what information will need to be included into each class, thus the generation of the Class Skeletons we created to go along side with these pieces of information. The Component diagrams we created allow for front-end use and back-end use and how the different servlets will interact with each other. After we had wrote the Design Document, there was a change in the specification, which removed the gender of monsters and made them asexual. Although this was to make the requirements simpler, it did mean having to go back and remove the inheritance diagram in relation to the gender breeding and fighting. Additionally we had our detailed designs, which allows for more information to be placed into a document, in which when referred back to by ourselves, we can see how to implement certain aspects of the design. To compliment these the algorithms, although slightly basic, are a clear, understandable way to implement the code into our program, to allow the battles to take place, the ability for someone to win and of course the breeding algorithm.

However after some feedback, we realized that there was some work to do on our Design Specification document. For example, the UML designs, need to be rewritten to include more detail in the grouped data, additionally the component diagrams weren’t quite laid out correctly, however have been amended so that the servlets are interacting more, rather than just having the singular servlet linking to everything. As stated before, with the requirement change, the female/male details are to be removed, since these are asexual monsters now, which leads to the database having to be edited for the breeding and monster holding data, as well as different breeding and fighting algorithms, with perhaps a bit more detail.

## Difficulties and their Resolutions

### Gantt Chat

The main issue that arose with the Gantt chart was the over complicated design package that we initially chose to use. The package was called Gantt project which we began to generate our Gantt chart in, due to being complicated and non-user friendly, we decided to scrap the package for a more user friendly piece of software. This allowed us to have the ability to quickly glance at the chart and see time frames and our milestones whereas Gantt project did not. Additionally the package we decided to use was more suited to our project needs.

### GitHub

We decided to use GitHub to upload and share the work we had done with other group members rather than using email or SVN. However a few issues arose with the use of this software, one of these was the lack of knowledge that some members of our group had with GitHub, this lead to deciding that in one of our group meetings we would have a tutorial session to teach members how to use GitHub. This was successful and allowed the sharing of work to flow, however there were a handful who never quite grasped the concept of GitHub. Additionally the rest of the group had the issue of version control conflicts and merging errors, however this was resolved once correct system file management was implemented.

### Inter Group Standards

Our group project has to communicate with other groups projects, taking that into account the standards meeting was arranged in order to develop a set of standards that will make the communication possible. Two people from each group had to attend this meeting and having so many people in one room resulted in having multiple opinions that were preventing standards to be developed in a fast and correct way. Server to server standards were not relevant because of the client change in the specification, additionally some features of the specification were not relevant to the program due to the standards.

### Technical Issues

Since we started to develop the program we encounter a number of technical difficulties, for example due to the requirement change meant that some of the code that we wrote was unusable as it catered to an incorrect specification. This then lead to database malfunctions which meant that the database we created had to be re-written. Whilst writing the program the main issue we had was the concept of group programming as there were a lot of conflicts in the code and some communication errors between group members. However this was overcome thanks to the decision to have individual group members working on different parts of the program and frequently pushing to our GitHub repository, thus allowing for constantly updated versions.

## Team Performance

As a team we felt that we all worked together well by overcoming the problems and difficulties that arose and successfully coming together to create our program and support documents. We had a few issues with attendance at meetings, however the people who did miss the meetings for an array of reasons, had the minutes to read and work to catch up on, which everyone did well.

## Historical Account

### Main Project Events

|  |  |  |  |
| --- | --- | --- | --- |
| # | Start Date | Event | End Date |
| TL.1 | 18/10/2012 | Interaction Design | 29/10/2012 |
| TL.1.1 | 18/10/2012 | Introduction | 29/10/2012 |
| TL.1.2 | 18/10/2012 | Overview | 29/10/2012 |
| TL.1.3 | 18/10/2012 | Use Case Diagram | 29/10/2012 |
| TL.1.4 | 18/10/2012 | User Interface Design | 29/10/2012 |
| TL.1.5 | 18/10/2012 | Gantt Chart | 29/10/2012 |
| TL.1.6 | 18/10/2012 | Risk Analysis | 29/10/2012 |
| TL.2 | 29/10/2012 | Test Specification | 13/11/2012 |
| TL.3 | 29/10/2012 | Design Specification | 03/12/2012 |
| TL.4 | 05/10/2012 | First Prototype | 10/12/2012 |
| TL.5 | 10/12/2012 | Further Integration | 21/01/2013 |
| TL.6 | 28/01/2013 | Integration and testing week | 01/02/2013 |
| TL.7 | 01/02/2013 | Delivery of software | 01/02/2013 |
| TL.8 | 01/02/2013 | Acceptance Testing | 01/02/2013 |
| TL.9 | 04/02/2013 | Document Hand In | 18/02/2013 |

### Producing a plan

From the start of this project, we all knew it was going to be difficult and require a lot of planning and allocation of jobs. In the first piece of documentation we generated a Project Plan to help us get the project off to a start. We met 2 to 3 times a week, and then every day during integration and testing week to allow us to constantly come together and see how each other were getting on. As well as meeting each week a few times, the coders of the groups came together a little bit more with the Project Manager to allow them to set some standards within the group, and the coding style. Along with this we had constant communication throughout the project using emails containing the minutes of the meetings if for whatever reason, someone missed the meeting. This then led to one of the group members generating a forum, in which we had the ability to communicate ideas that we had, or possible code that had been generated. Although this was made with good intentions, the use of the forum was extremely minimal, partly due to the decision to use GitHub, everything was stored on there, thus, reducing the need for the forum, it was a good idea, and a possible idea for the future in terms of other projects, however for this one, we felt it didn’t meet the requirements we were going to use it for. As well as getting started on creating meeting times, allocating jobs, we had to begin actually generating the plan for the project, which meant looking at Use Case diagrams, sketches by hand that we did, which then lead onto a HTML generated design.

### Time Management

Managing time is crucial for every project that takes time to complete. Our project was given a number of deadlines that made the time management easier. We as a group were set regular milestones, which didn’t just include what was in the Gantt chart, but for example work we had to take home to finish off. Due to having multiple meetings a week, this allowed us to start on a task at the beginning of the week, and having it done by another meeting that same week, meaning that when we were all busy with additional modules, we weren’t playing catch up, rather than having a final read through along with the QA manager making sure everything was formatted correctly for the hand in. additionally thanks for having an incredibly detailed task list, we were able to strategize the oncoming issues we had to overcome, and tasks we had to complete, thus meaning that, for example, when we began coding, we knew the order of things that had to be completed, which removed the concept of downtime within the group, most importantly, this allowed every member to be doing something all the time.

## Final State of Project

/\* not done, Pavel was doing this\*/

## Performance of each team member

/\* not done, Llion was doing this\*/

## Critical Evaluation of the team and the Project

/\* not done, Llion and Yarrow were doing this\*/

# Test Report

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Test Ref* | Requirement being tested | Test Content | Input | Output | Pass Criteria | Pass Y/N |
| *SE\_TT\_001* | FR1 | Check that a user can register. | Valid data for the registration form, email, user, and password, such as; “[sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)” as email,  “secret” as passwords, and “sis13” as username. | User should be redirected to the main page, and the new user should be created in the database. | New user is created in DB, and user is redirected to main page. The information in the database is the same as that entered in the form |  |
| *SE\_TT\_002* | FR1 | If the password given in the confirmation is different do not create a new user. | “[sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)” as email,  “foo” as password, and “bar” as confirmation, and “sis13” as username. | The user should be shown an error saying that the two passwords are not the same. | The user is shown an error and no new user is created in the DB. |  |
| *SE\_TT\_003* | FR1 | See if email is registered already | “[sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)” as email,  “secret” as passwords, and “sis13” as username. | The user should be shown an error saying that email is in use, and link to recover a password. | Error is shown, and no new user is created in the DB. |  |
| *SE\_TT\_004* | FR1 | See if user can login with valid login information | “[sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)” as email,  “secret” as password | User should get a session and be redirected to mainpage. | The user is given a valid session and is redirected to the mainpage. |  |
| *SE\_TT\_005* | FR1 | Invalid login information should be rejected. | “[sis13@aber.ac.uk](mailto:sis13@aber.ac.uk)” as user and “wrongpassw” as password. | The user should see an error saying that his login information is incorrect. | The user is shown an error and no valid session is created for the user |  |
| *SE\_TT\_006* | FR1 | Check that a user can register. | Valid data for the registration form, email, user, and password | User should be redirected to the main page, and the new user should be created in the DB. | New user is created in DB, and user is redirected to main page. |  |
| *SE\_TT\_007* | FR2 | User should see list of their friends upon login. | Log in details.  lwv@aber.ac.uk as the username.  1234pass as the password. Also accounts for the friends are entered into the DB. | List of users friends:  John  Paul. | Shows List of friends. |  |
| *SE\_TT\_008* | FR2 | User should be able to send a friend request by entering an email address. | Enter email: [yap@aber.ac.uk](mailto:yap@aber.ac.uk), where there is a an account registered for this email. | Tell the User that a friend request was sent successfully. | Message saying request sent. |  |
| *SE\_TT\_009* | FR2 | Check that user had entered a vald email address. | Enter a invalid email:  lwv@@dg.cffriu. | Error message warns the user they must enter a valid email. | Warning of invalid email. |  |
| *SE\_TT\_010* | FR2 | Check the user has entered a users email that exists on a Monster Mash Server. | Enter a email not presently used by a user:  llion@me.com. | Error message saying that no such user exists. | Warning that no such user exists. |  |
| *SE\_TT\_011* | FR3 | Check that new window will appear when user hovers mouse over monsters in the monster list. | Hover mouse over single monster from the monster list. | Small window with all the monster’s attributes appears. | All the monster attributes are visible to the user in new window. |  |
| *SE\_TT\_012* | FR3 | When user creates new account, it receives a random monster and small amount of money. | User creates new account. | New account with basic monster and small amount of money is created. | New user has basic monster and small pot of virtual money attached to their account. |  |
| *SE\_TT\_013* | FR3 | Check that there is some probability that monster can be injured or ill. | Leave server running with some random account for x time. | After x time it is possible that monster is ill or injured. This will be reflected in the monsters attributes. | Monster being ill is added to the attributes and shown to a user. |  |
| *SE\_TT\_014* | FR3 | Check that monster is getting older while server is running. | Leave server running with some random account for x time. | After x time monster is older. | Monster is older and the age attributes is increased and shown in the attributes box. |  |
| *SE\_TT\_015* | FR3 | Check that very old or ill monster can die. | Leave server running for a long time with random account. | After x time monster will die, because it was very old or ill. | Monster dies and is removed from the list, an update message is sent to the user. |  |
| *SE\_TT\_016* | FR3 | Check that there is some probability that monster gets better after illness. | Leave server running with some random account for x time. | After x time monster gets better. | Monster gets better, illness is no-longer displayed as part of its attributes. |  |
| *SE\_TT\_017* | FR4 | To test that a user has the ability to “monster mash” (battle other users with their monsters. | Clicking on a friend and then choosing to battle them. | Some data about the battle should be given out. | That correct data about the battle is sent to both players |  |
| *SE\_TT\_018* | FR4 | To test whether the system accurately assigns the monetary value to monsters and that this is distributed correctly after battle. | Getting two monsters of known value to fight. | The distribution of money to the winning player. | The correct amount of money is added to the winners total. |  |
| *SE\_TT\_019* | FR4 | To test whether a user can send battle requests. | Clicking on another player and choosing to send them a request. | The request received by the other player. | The second player receives the notification and correct data about being challenged. |  |
| *SE\_TT\_020* | FR4 | To test that our ‘battle algorithm’ is working and produces the expected results. | Conducting multiple battles with known attributes, so that the likely outcome is known. | The battle logs. | That although there is an element of chance involved the algorithm is carried out. |  |
| *SE\_TT\_021* | FR4 | To test whether an element of chance is included in the algorithm. | Conducting many battles with monsters of known attributes. | The battle logs. | That in one of the battles a monster with lower attributes that should have lost does in fact win. |  |
| *SE\_TT\_022* | FR5 | User enters an email not registered on any servers an error should be shown. | An email that is not registered such as “[friend@server.com](mailto:friend@server.com)” as email. | A message saying the user could not be found. | A message saying that the user is not found and no request is stored. |  |
| *SE\_TT\_023* | FR5 | Whether a user can buy a monster listed on another server. | User buys a monster stored on another server. | Monster has been bought message. | Message that monster has been bought and monster is now stored under the user locally, and users founds are changed. |  |
| *SE\_TT\_024* | FR5 | User cannot buy monsters without the right amount of money. | User buys a monster stored on another server | Error message saying the user has insufficient funds is shown | The buy request is not sent and user is shown an error message. |  |
| *SE\_TT\_025* | FR5 | User can sell monster to friends on other servers. | Friends on a different server buys a monster from our test user | A notification that the monster is sold, to whom and for what. | A notification is shown to the user and the monster is transferred to the other server and removing the local entry. |  |
| *SE\_TT\_026* | FR5 | User’s monster is not sold when the request has insufficient funds. | A friend on a different server buys a monster from our test user while having insufficient funds. | Response to the server sending the request with an exception. | The users monster is not sold and exception is sent to the server sending the request. |  |
| *SE\_TT\_027* | FR5 | Users from other servers can request user data from us by passing a valid email. | Remote server requests the data for a user with a valid email. | Response containing the user data is sent. | Response containing the user data is sent. |  |
| *SE\_TT\_028* | FR5 | A user can breed their monster with users on other servers. | User views the mating list of friends and accepts a mating offer with a user on another server. | The local user should receive some funds and a message saying the “breeding” was successful. | Users receives funds, is shown a notification and the results are stored. |  |
| *SE\_TT\_029* | FR5 | User can list their monsters for breeding so users on other servers can breed with them | User lists a monster for breeding, and a user on another server accepts the offer. | Local user should get a notification with the attributes of the child | User receives notification and the child is stored in users monster list. |  |
| *SE\_TT\_030* | FR5 | User can send fight request to friends on other servers. | User selects a friend and sends a fight request. | The user gets a notification that the request is sent. | Request is sent to remote server, and the user gets a notification. |  |
| *SE\_TT\_031* | FR5 | Remote request is accepted and the monsters fight. | A request has been sent to a remote server. | The output should be a message saying if the user has won/lost with new attributes for the user’s monster. | User should be shown notification, and the result should be stored correctly in the DB. |  |
| *SE\_TT\_032* | FR5 | User remote fight request is canceled. | A fight request is sent to another user, and then the pending request is canceled. | No battle occurs, and notification is sent to the user. | A notification detailing the challenge and the subsequent withdrawal is sent to both users |  |
| *SE\_TT\_033* | FR5 | User can accept remote fight requests | User clicks the accept button of a remote fight request. | The output should be a message saying if the user has won/lost with new attributes for the user’s monster. | User should be shown notification, and the result should be stored correctly in the DB. |  |
| *SE\_TT\_034* | FR5 | User can decline remote fight requests. | User clicks the decline button. | Notification should be removed. | No battle is held and the notifications of the users updated. |  |
| *SE\_TT\_035* | FR6 | Successfully register with correct credentials | Username, e-mail address, password | No error, the user is registered and logged in to the Mainpage. | Data passed correctly between server and database |  |
| *SE\_TT\_036* | FR6 | Reject registration with already registered username | Existing username, e-mail address, password | Error about username already taken | Server rejects adding record to a database after comparing username with existing records |  |
| *SE\_TT\_037* | FR6 | Reject registration with already registered e-mail address | Username, existing e-mail address, password | Error about e-mail address already registered | Server rejects adding record to a database after comparing e-mail with existing records |  |
| *SE\_TT\_038* | FR6 | Reject registration with password not meeting criteria (e.g. too short) | Username, e-mail address, weak password | Error about too weak password | Server rejects adding record to a database after unsuccessful validation of password |  |
| *SE\_TT\_039* | FR6 | Login attempt with correct username and password | Existing username, correct password | No error | Data passed correctly between server and database |  |
| *SE\_TT\_040* | FR6 | Login attempt with incorrect username | Not existing username, any password | Error about incorrect username or password | Data passed correctly between server and database |  |
| *SE\_TT\_041* | FR6 | Login attempt with incorrect password | Existing username, incorrect password | Error about incorrect username and password | Data passed correctly between server and database |  |
| *SE\_TT\_042* | FR6 | Check that a friend request can be sent to existing user | Send a friend request to existing user | User gets the request | Server passes the request |  |
| *SE\_TT\_043* | FR6 | Check that a friend request cannot be sent to not existing user | Send a friend request to not existing user | Error message sent to the user notifying them that the user does not exist. | Server returns an error after not finding the user in the database |  |
| *SE\_TT\_044* | FR6 | Remove friend from friends list | Clicking on a friend and select to remove that friend. | Friend list updated without selected friend. | Server removes the connection |  |
| *SE\_TT\_045* | FR6 | Remove user who has removed you from their friends list is also removed from your friends list. | By having two test users that are friends were one removes the other from their friends list. | The second user's friend list is also updated with the first user removed from their friends list as well. | Server returns an error after not finding the user in the friends list |  |
| *SE\_TT\_046* | FR6 | Accept someone’s request. | Accept request. | User added to friends. | Server adds users to each others friends lists. |  |
| *SE\_TT\_047* | FR6 | Deny someone’s friend request request. | Deny request. | The user is not added to the friend list and the request is removed. | Server cancels the request and returns an appropriate message to user who sent the request |  |
| *SE\_TT\_048* | FR6 | Check that a monster can be offered for sale | Offer monster for sale | Other users see the monster on the market and are able to buy it | Another user is able to see and purchase the monsters the test user has put up for sale. |  |
| *SE\_TT\_049* | FR6 | Check that a monster can be bought | Buy a monster | Monster is transferred to the user who bought it | The test user is able to purchase a monster put up for sale by another test user. |  |
| *SE\_TT\_050* | FR6 | Check that a monster can be offered for breeding | Offer monster for breeding | Other users see the monster on the breeding page and are able to purchase breeding | Another test user is able to see and accept the offer for breeding |  |
| *SE\_TT\_051* | FR6 | Check that a monster can be purchased for breeding | Purchase monster for breeding | Clicking and selecting monsters that are being offered for breeding on the mating screen | The user is able to accept the offers of other users for breeding |  |
| *SE\_TT\_052* | FR7 | Check that user with correct credentials can be logged in | Username and password of a registered user | User is logged in and redirected to the main game page | Data passed correctly between client and server |  |
| *SE\_TT\_053* | FR7 | Check that user with incorrect credentials cannot be logged in | Incorrect username and / or password | User is redirected to a page displaying “Incorrect username / password” message | Data passed correctly between client and server |  |
| *SE\_TT\_054* | FR7 | Check that a logged in user can log out | Click the “logout” button | User redirected back to login screen | Data passed correctly between client and server |  |
| *SE\_TT\_055* | FR7 | Check that user can register | Fill in the registration form and press “register button” | User is now registered and can log in | Data passed correctly between client and server |  |
| *SE\_TT\_056* | FR8 | Whether the main screen is visible after a user has logged. | Their log in details. | The main screen is displayed. | The main screen is correct displayed with the update panel and side tabs. |  |
| *SE\_TT\_057* | FR8 | Whether the update panel is correctly displayed | Looking at the main page and the update panel | The update panel is displayed | The current updates for that is user is displayed in the center of the screen. |  |
| *SE\_TT\_058* | FR8 | Whether the friends list is displayed | Looking at the main screen | The friends list is displayed to a user | The list of friends is correctly shown at the side of the screen to the user |  |
| *SE\_TT\_059* | FR8 | Whether the monster list is displayed | Looking at the main screen | The monster list is displayed to the user | The list of the users monsters is displayed at the side of the screen. |  |
| *SE\_TT\_060* | FR8 | Whether challenge requests are shown to a user | Looking at the update panel, after a second test user issues a challenge to the user. | The updates on the update panel | The challenge is displayed in the update panel correctly. |  |
| *SE\_TT\_061* | FR8 | Whether a user can interact with the displayed elements of the Mainpage. | Clicking on various aspects of the page and interacting with them such as the monsters or friends. | The update for chosen action occurs such as a battle or friend request | The action chosen by the user occurs and the update panel is updated with this information. |  |
| *SE\_TT\_062* | FR9 | When a user’s friend request is accepted the friend should be added to the user’s friend list. | Friend request will be sent and needs to be accepted by the friend. | Friend should appear on the users friend list. | Friend appears on users friend list. |  |
| *SE\_TT\_063* | FR9 | User should be able to decline a friend request and the request should be removed. | The user clicks decline on the friend request. | Friend request should disappear and the friend should not appear on the list. | Request disappears and friend is added. |  |
| *SE\_TT\_064* | FR9 | When a users friend request is declined the rejected request should disappear. | Friend request will be sent and needs to be declined by the friend. | Friend shouldn’t appear on the users friend list and the request should disappear. | Request disappears and no friend is added. |  |
| *SE\_TT\_065* | FR10 | Whether a friend’s monster that was killed, has been removed from their Monster list. | Win a battle against a friend and kill their monster. | Monster should now be removed from the list following the battle. | The monster list for that user is updated in the DB and no longer appears there or is shown to the user. |  |
| *SE\_TT\_066* | FR10 | That a user’s monster was killed and has been removed from their monster list. | Lose a battle, and have the monster killed. | The user’s monster should now be removed from the user’s monster list. | The monster is no longer displayed |  |
| *SE\_TT\_067* | FR10 | Winning a battle will cause the user to gain prize money, and have it added to their account | Win a battle, with prize money. | The user’s money should have increased by the prize money amount. Adding this to their previous total. | The user has increased wealth and the display is updated |  |
| *SE\_TT\_068* | FR10 | Update the user’s monster after a battle. | Win a battle and receive an injury | Monster should be damaged in some way, following the battle with another monster. | The monsters attributes should be changed to show that it has an injury |  |
| *SE\_TT\_069* | FR11 | User should be able to see a list of their friends (Including themselves) with the wealth of each, ordered by wealth. | User should click on the leaderboard. | List of friends and their wealth should appear in order of wealth. | List of friends in order of wealth. |  |
| *SE\_TT\_070* | FR7 | Load home page | Type in URL of server to browser | Application should start and display a screen with login and registration options | Applications starts  and options displayed. |  |
| *SE\_TT\_071* | FR1, Fr6 | Register valid ne iser | Register user id  myid@aber.ac.  uk and  password  pword | Registration  accepted  message or  next screen. | Registration accepted |  |
|  |  |  |  |  |  |  |

### Failed test explanations

# MAINTENANCE MANUAL

## Program Description

Monster Mash is a web based game, that runs on standard browsers, in which each client hosts a selection of monsters, which contain certain genetic attributes to enhance battling and breeding with other clients on different servers. Battling can be used to gain money; alternatively the players can breed and sell monsters for additional funds. Additionally, players can send and receive friend requests to interact in the game with zed players. The objective of the game is to be top of the leaderboard, and to enhance knowledge of genetics whilst enjoying the game itself at the same time.

## Program Structure

## /\*stan\*/

## Data Areas

/\*Pavel\*/

## Algorithms and Methods

/\* I think filip was meant to be doing some of this\*/

Source Packages

Default package

CreateAccountPage.java

private boolean isValidEmailAddress(String email) {

boolean result = true;

try {

InternetAddress emailAddr = new InternetAddress(email);

emailAddr.validate();

} catch (AddressException e) {

result = false;

}

return result;

}

This method verifies that the email entered into the register page is a true email, and it also tests the email address to check whether or not it will validate. This will return the result false, if it fails, and will continue if it is true.

public String MD5(String md5) {

try {

java.security.MessageDigest md = java.security.MessageDigest.getInstance("MD5");

byte[] array = md.digest(md5.getBytes());

StringBuffer sb = new StringBuffer();

for (int i = 0; i < array.length; ++i) {

sb.append(Integer.toHexString((array[i] & 0xFF) | 0x100).substring(1,3));

}

return sb.toString();

} catch (java.security.NoSuchAlgorithmException e) {

}

return null;

}

I have no idea what this does.

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String email = request.getParameter("email");

String username = request.getParameter("username");

String monsterName = request.getParameter("monster");

String password = request.getParameter("password");

String cpassword = request.getParameter("cpassword");

String errorMessage = null;

PersistenceManager pm = new PersistenceManager();

// Simple validation

if(email.length() < 1 || !isValidEmailAddress(email)){

errorMessage = "Please enter correct email address.";

}else if(monsterName.length() < 5 || monsterName.length() > 32){

errorMessage = "Please enter correct monster name.";

}else if(password.length() < 5 || password.length() > 255){

errorMessage = "Please enter correct password.";

}else if(!password.equals(cpassword)){

errorMessage = "Passwords are not the same.";

}else if(pm.accountExists(email)){

errorMessage = "There is already account with this email address.";

}else{

password = MD5(password);

Player tmp = new Player(email, username, password, MONEY\_AMOUNT, monsterName);

// Store player in DB

pm.storePlayer(tmp);

// Redirect to login page

request.setAttribute("message", "Account created successfully. You can sign in now.");

request.getRequestDispatcher("/WEB-INF/login\_page.jsp").forward(request, response);

return;

}

// Display error message

request.setAttribute("errorMessage", errorMessage);

request.getRequestDispatcher("/WEB-INF/create\_account\_page.jsp").forward(request, response);

}

To me, this is an algorithm loop, but I can explain in the methods area instead.

HighscoresPage.java

private void getDataFromDB(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpSession session = request.getSession(false);

if(session == null || session.getAttribute("user") == null){

// Redirects when user is not logged in

response.sendRedirect("");

}else{

Player current = (Player)session.getAttribute("user");

PersistenceManager pm = new PersistenceManager();

// Updates player informations

current = pm.getPlayer(current.getUserID());

session.setAttribute("user", current);

// Saves all notifications to attribute

request.setAttribute("notificationList", current.getNotifications());

// Saves all friends and friend requests to attribute

request.setAttribute("friendList", current.getFriends());

// Saves all friend requests to attribute

request.setAttribute("requestList", pm.getFriendRequestList(current.getUserID()));

// Saves all monsters to attribute

request.setAttribute("monsterList", pm.getMonsterList(current.getUserID()));

request.getRequestDispatcher("/WEB-INF/highscores\_page.jsp").forward(request, response);

}

}

This method gets the data from the databse through a HTTP servlet request, it will redirect when user isn’t logged in, however will save the users data to the database when logged in. It gets the players information, saves all attributes and monsters, and finally saves the monster list to the user ID in the highscores.jsp file.

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpSession session = request.getSession(false);

if(session == null || session.getAttribute("user") == null){

// Redirects when user is not logged in

response.sendRedirect("");

}else{

PersistenceManager pm = new PersistenceManager();

Player current = (Player)session.getAttribute("user");

ArrayList<String> highscores = pm.getHighscores(current.getUserID());

request.setAttribute("highscores", highscores);

this.getDataFromDB(request, response);

}

}

Similar to before, however this actually requests the highscores and responds the database data into a table format to display the information.

LoginPage.java

public String MD5(String md5) {

try {

java.security.MessageDigest md = java.security.MessageDigest.getInstance("MD5");

byte[] array = md.digest(md5.getBytes());

StringBuffer sb = new StringBuffer();

for (int i = 0; i < array.length; ++i) {

sb.append(Integer.toHexString((array[i] & 0xFF) | 0x100).substring(1,3));

}

return sb.toString();

} catch (java.security.NoSuchAlgorithmException e) {

}

return null;

}

This method I don’t actually know what this does.

MainPage.java

private void sendFriendRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpSession session = request.getSession(false);

// Checks if user is logged in

if(session != null && session.getAttribute("user") != null){

// Gets email from POST

String email = request.getParameter("email");

PersistenceManager pm = new PersistenceManager();

Player sender = (Player)session.getAttribute("user");

// Checks if user with this email address exists

String[] receiver = pm.getPlayerIdAndServer(email);

if(receiver[0].equals("0")){

request.setAttribute("alertMessage", "Cannot find user with this email address.");

}else if(pm.isFriendRequestSent(sender.getUserID(), receiver[0])){

request.setAttribute("alertMessage", "Cannot send friend request to this player.");

}else if(sender.getUserID().equals(receiver[0])){

request.setAttribute("alertMessage", "Cannot send friend request to yourself.");

}else{

String message = "Friend request to <b>"+email+"</b> sent successfully.";

sender.addNotification(new Notification(message, "You have sent friend request to <b>"+email+"</b>.", sender));

pm.storeNotifications(sender);

int receiverServerID = Integer.parseInt(receiver[1]);

pm.sendFriendRequest(sender.getUserID(), receiver[0], receiverServerID);

if(receiver[1].equals("12")){

//Receiver is on our server

Player receiverObject = pm.getPlayer(receiver[0]);

receiverObject.addNotification(new Notification("Received friend request from <b>"+sender.getUsername()+"</b>", "You have received friend request from <b>"+sender.getUsername()+"</b>.", receiverObject));

pm.storeNotifications(receiverObject);

// Save updated player's object in session

session.setAttribute("user", sender);

request.setAttribute("alertMessage", message);

}else{

// TODO: Receiver is on different server

}

}

}

}

The point of this method is to check the HTTP servlet to run the PersistenceManager and receive the PlayerID and Sever (thus with the email ID). Additionally it requests the Attributes of the friend, and returns a non valid email error message if false, along with if you send the request to yourself, another error message will appear. If the friend request is successful it will add a new notifaction message saying it has been success with the email in the javascript generate message. If it is successful, the player will receive the friend object and display in the friends list. Alternatively if you receive a request, it will show another notifaction stating that someone has added you, and similiarly if accept the request, then it will display the friend object in the friends list. TO DO DIFFERENT SERVER FRIENDS.

private void respondToFriendRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpSession session = request.getSession(false);

// Check if user is logged in

if(session != null && session.getAttribute("user") != null){

Player logged = (Player)session.getAttribute("user");

PersistenceManager pm = new PersistenceManager();

// Check if acceptFriendRequest occured:

if(request.getParameter("acceptFriendRequest") != null){

String friendshipID = request.getParameter("acceptFriendRequest");

pm.acceptFriendRequest(friendshipID, logged.getUserID());

}

// Check if cancelFriendRequest occured:

if(request.getParameter("cancelFriendRequest") != null){

String friendshipID = request.getParameter("cancelFriendRequest");

pm.cancelFriendRequest(friendshipID, logged.getUserID());

}

}

}

As an addition to the previous method, this method checks if the response of the friendship request was successful, or it was cancelled. This gives the option to cancel a request for friendship, or to check if the friendship was successful.

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpSession session = request.getSession(false);

if (session == null || session.getAttribute("user") == null) {

// Redirects when user is not logged in

response.sendRedirect("");

} else {

PersistenceManager pm = new PersistenceManager();

Player current = (Player) session.getAttribute("user");

// Check if user wants to cancel offer

this.cancelOffer(request, response, pm, current);

// Check if user wants to buy monster

this.buyMonster(request, response, pm, current);

ArrayList<Monster> monsters = pm.getMonstersForSale(current.getUserID());

// Prepare strings:

ArrayList<String> monstersForSale = new ArrayList<String>();

for(Monster m: monsters){

monstersForSale.add("<li><a href=\"market?monster="+m.getId()+"&server="+m.getServerID()+"\"><b>Name:</b> "+m.getName()+" | <b>Owner:</b> "+pm.getPlayerUsername(m.getUserID(), m.getServerID())+" | <b>Price:</b> "+m.getSaleOffer()+"$ | <b>Stats:</b> def: "+(int)(m.getCurrentDefence()\*100)+" / hp: "+(int)(m.getCurrentHealth()\*100)+" / str: "+(int)(m.getCurrentStrength()\*100)+" </a></li>");

}

request.setAttribute("monstersForSale", monstersForSale);

this.getDataFromDB(request, response);

}

}

For the buying monsters method, like other methods it begins with the session being active, if true, it has the option to check the player has cnacelled their offer. Then we look at buying the monster, which goes into the monster array, which has been created from the monsters for sale method, which returns the monsters that are in fact for sale. By getting the server ID, owners name, player Username, ID and details of the monster, the monster is then bought, and requests the monster for sale if false.

private void cancelOffer(HttpServletRequest request, HttpServletResponse response, PersistenceManager pm, Player current) throws ServletException, IOException {

String monsterID = request.getParameter("cancelOffer");

if(monsterID != null){

if(pm.cancelMonsterOffer(current.getUserID(), monsterID)){

current.addNotification(new Notification("You have canceled your offer of <b>"+pm.getMonsterName(monsterID)+"</b>.", "<b>"+pm.getMonsterName(monsterID)+"</b> offer has been canceled by you. Now offer will not apper on the market.", current));

pm.storeNotifications(current);

}

}

}

The method above is the cancelOffer method, which when the cancelation is true, will set the notifaction to show that the offer has in fact been cancelled, and the offer will no longer appear on the market.

private void buyMonster(HttpServletRequest request, HttpServletResponse response, PersistenceManager pm, Player current) throws ServletException, IOException {

String monsterID = request.getParameter("monster");

String server = request.getParameter("server");

if(monsterID != null && server != null){

try{

String message = null;

int serverID = Integer.parseInt(server);

if(!pm.monsterExists(monsterID, serverID)){

message = "Monster doesn't exists";

}else if(!pm.canUserBuyMonster(current.getMoney(), monsterID, serverID)){

message = "You do not have enough money for buying this monster.";

}else{

pm.buyMonster(current.getUserID(), monsterID, serverID);

message = "You have bought new monster called "+pm.getMonsterName(monsterID)+".";

current.addNotification(new Notification("You have bought new monster called <b>"+pm.getMonsterName(monsterID)+"</b>.", "You have bought new monster called <b>"+pm.getMonsterName(monsterID)+"</b>. It will appear on your monster list now.", current));

pm.storeNotifications(current);

}

request.setAttribute("alertMessage", message);

}catch(Exception e){

}

}

}

The method buyMonster firstly checks to see if a monster actually exsists, and returning that fact in a message if false, similarly, if the user cannot afford the monster they are trying to buy then this will appear as a message as well. If all is well, and true, then the current monster selected from the array will be purchased and displayed as a notification on the homepage, which is requested from the servlet.

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpSession session = request.getSession(false);

if (session == null || session.getAttribute("user") == null) {

// Redirects when user is not logged in

response.sendRedirect("");

} else {

// Make new offer:

Player current = (Player) session.getAttribute("user");

String monsterID = request.getParameter("monsterID");

String offerAmount = request.getParameter("offerAmount");

String error = null;

PersistenceManager pm = new PersistenceManager();

if(monsterID == null || offerAmount == null){

error = "Please fill both fields.";

}else if(monsterID.length() < 1){

error = "Please select monster name.";

}else if(offerAmount.length() < 1){

error = "Please specify your offer amount.";

}else{

int amount = 0;

try{

amount = Integer.parseInt(offerAmount);

if(!pm.makeNewMarketOffer(current.getUserID(), monsterID, amount)){

error = "Incorrect monster name.";

}

}catch(Exception e){

error = "Incorrect amount.";

}

}

if(error != null){

request.setAttribute("alertMessage", error);

}else{

current.addNotification(new Notification("You offered <b>"+pm.getMonsterName(monsterID)+"</b> for sale for <b>"+offerAmount+"$</b>.", "<b>"+pm.getMonsterName(monsterID)+"</b> is now available for sale for <b>"+offerAmount+"$</b>. You cannot use this monster until you cancel your offer.", current));

pm.storeNotifications(current);

request.setAttribute("alertMessage", "You offered "+pm.getMonsterName(monsterID)+" for sale for <b>"+offerAmount+"$</b>.");

}

doGet(request, response);

}

}

The doPost method is a method that allows the player to offer his monster up for sale, requiring the monsters name and a price tag. This is done by setting the user, monster and offer attributes, with constraints of monster ID <1 and amount <1, if either of these two parameters are true, the program will throw an error. If the new market offer is successful, it will return a notification providing the previous does not equal null.

Database

PersistenceManager

private String randomString(int length){

Random random = new SecureRandom();

String letters = "abcdefghjkmnpqrstuvwxyzABCDEFGHJKMNPQRSTUVWXYZ23456789";

String pw = "";

for (int i=0; i<length; i++){

int index = (int)(random.nextDouble()\*letters.length());

pw += letters.substring(index, index+1);

}

return pw;

}

This method is the generator for the random monster and player names, it generates a random name, using the string letters and does a for loop to generate a selection of letters placed together. It then returns the result.

public boolean accountExists(String userID){

int count = 0;

try{

Statement stmt = connection.createStatement();

stmt = connection.createStatement();

ResultSet results = stmt.executeQuery("SELECT count(\"id\") FROM \"Player\" WHERE \"username\" = '"+userID+"'");

results.next();

count = results.getInt(1);

results.close();

stmt.close();

}catch (SQLException sqlExcept){

this.error = sqlExcept.getMessage();

}

// TODO: Check other servers! (SERVER<->SERVER)

if(count > 0){

return true;

}

return false;

}

booleanaccountExsists method is a true or false return that checks to the connection in the stmt, which then links into the database to find userID, which then counts the results and returns if the account exsists TO DO OTHER SERVERS CHECK

public ArrayList<Player> getFriendList(String playerID){

ArrayList<Player> friendList = new ArrayList<Player>();

try{

Statement stmt = connection.createStatement();

ResultSet result = stmt.executeQuery("SELECT \* FROM \"Friendship\" WHERE (\"sender\_id\" = '"+playerID+"' OR \"receiver\_id\" = '"+playerID+"') AND \"confirmed\" = 'Y'");

while(result.next()){

if(result.getString("sender\_id").equals(playerID+"")){

// Sender String id, String name, int serverID

friendList.add(new Player(result.getString("receiver\_id"), this.getPlayerUsername(result.getString("receiver\_id"), result.getInt("receiver\_server\_id")), result.getInt("receiver\_server\_id")));

}else{

// Receiver

friendList.add(new Player(result.getString("sender\_id"), this.getPlayerUsername(result.getString("sender\_id"), result.getInt("sender\_server\_id")), result.getInt("sender\_server\_id")));

}

}

result.close();

stmt.close();

}catch (SQLException sqlExcept){

System.err.println("Selecting friendships from DB error:\n"+sqlExcept.getMessage());

this.error = sqlExcept.getMessage();

}

return friendList;

}

This method works to return the friendslist, by creating a friendlist array in the platers ID and dragging it across from the databse in the friendship column and then returns the results through the user and server ID whilst adding the friends into the array. Both connections then close, with error messages is anything returns false, thus finally returning the friendlist array into object format on the GUI. This is the same for notifications, friendship requests and returning the monster list, just from different columns in the database.

public void sendFriendRequest(String senderID, String receiverID, int receiverServerID){

try{

Statement stmt = connection.createStatement();

String id = this.randomString(16);

stmt.execute("INSERT INTO \"Friendship\" (\"id\", \"sender\_id\", \"receiver\_id\", \"sender\_server\_id\", \"receiver\_server\_id\", \"confirmed\") VALUES ('"+id+"', '"+senderID+"', '"+receiverID+"', 12, "+receiverServerID+", 'N')");

}catch(SQLException sqlExcept){

System.err.println(sqlExcept.getMessage());

this.error = sqlExcept.getMessage();

}

}

Alternatively the sendFriendRequest connects to the database in the same way as before, however it inserts into the database using a SQL statement and confirms the values of the sender and receiver ID, whilst at the same time checking for system errors and SQL exceptions. This is the same concept for receiving a friendrequest and accepting/denying a request.

public ArrayList<String> getHighscores(String playerID){

ArrayList<Player> friends = this.getFriendList(playerID);

ArrayList<String> friendIDs = new ArrayList<String>();

ArrayList<String> toReturn = new ArrayList<String>();

for(Player p: friends){

if(p.getServerID() == 12){

friendIDs.add(p.getUserID());

}else{

// TODO: get player's money amount from different server

}

}

friendIDs.add(playerID);

// Preparing query

String query = "SELECT \* FROM \"Player\" WHERE ";

for(String s: friendIDs){

query += "\"id\" = '"+s+"' OR ";

}

query = query.substring(0, query.length()-4);

query += "ORDER BY \"money\" DESC";

try{

Statement stmt = connection.createStatement();

ResultSet result = stmt.executeQuery(query);

int i = 1;

while(result.next()){

toReturn.add("<tr><td>"+i+".</td><td><b>"+result.getString("username")+"</b></td><td>"+result.getInt("money")+"$</td></tr>");

i++;

}

result.close();

stmt.close();

}catch (SQLException sqlExcept){

System.err.println(sqlExcept.getMessage());

this.error = sqlExcept.getMessage();

}

return toReturn;

}

The getHighscores array method begins by generating 3 arrays and connecting to the server ID, thus recieveing friends ID and adding the player ID from the player column in the database. It is then ordered by amount of money descending. Finally the query is excuting which is looped until all results are returned into HTML table. The return gets the strings of the username and the int of money which after ordering gets closed along with the stmt and is returned to the final array to display in the HTML table form. This is similar to the monsterforbreeding method and monsterforsale method, which both connect into different columnsof the database.

Algorithms

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

String email = request.getParameter("email");

String password = request.getParameter("password");

if(email.length() < 1 || password.length() < 1){

request.setAttribute("errorMessage", "Please enter both your email and password.");

request.getRequestDispatcher("/WEB-INF/login\_page.jsp").forward(request, response);

}else{

PersistenceManager pm = new PersistenceManager();

password = this.MD5(password);

Player selected = pm.doLogin(email, password);

if(selected != null){

// If player exists save object to the session called "user"

HttpSession session = request.getSession(true);

session.setAttribute("user", selected);

response.sendRedirect("main");

}else{

// If null, there's no player with this email and password

request.setAttribute("errorMessage", "Password or email address is incorrect.");

request.getRequestDispatcher("/WEB-INF/login\_page.jsp").forward(request, response);

}

}

}

}

## Files

/\*stan\*/

## Interface Description

## Possible Improvements

/\*Llion\*/

## Possible Changes

## Limitations

/\*Sindre\*/

## Known Bugs

/\*Sindre\*/

## Rebuilding and Re-Testing Suggestions

/\*Stan\*/

REFERENCES

[1]

DOCUMENT HISTORY

| *Version* | *CCF No.* | *Date* | *Changes made to document* | *Changed by* |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |