

The Software Development Life Cycle – Group Project

End-of-Project Report

Author(s)	John Bolton, Adrian Gawryszewski, Sam Clements, Stoyko Kodzhabashev, Nathan Hand, Daniel Cornwell, Daniel Stuecy, Miles Warburton, James Barnett
Configuration Reference	
Last Updated	17/02/13
Version Number	1
Document Status	Release

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Copyright © Aberystwyth University 2012-13

Table of Contents

1	Management Summary.....	4
2	A historical account of the project.....	5
3	Final state of the project.....	7
3.1	Working Features.....	7
3.2	Unimplemented features.....	7
4	Performance of each team member.....	8
4.1	Daniel Cornwell (Deputy Project Manager).....	8
4.2	Daniel Steucy.....	8
4.3	Adrian Gawryszewski.....	8
4.4	Stoyko Kodzhabeshev.....	9
4.5	Sam Clements (Deputy QA Manager).....	9
4.6	Miles Warburton.....	10
4.7	John Bolton (QA Manager).....	10
4.8	James Barnett.....	11
4.9	Nathan Hand (Project Manager).....	11
5	Evaluation of the Team and the entire project.....	12
5.1	How did the team perform as a whole, and how could that have been improved?.....	12
5.2	How could the project have been improved?.....	12
5.3	What were the most important lessons learned about software projects and about working in teams?.....	12
6	Appendices.....	13
6.1	Project test report.....	13
6.2	Project maintenance manual.....	21
6.2.1	Program description.....	21
6.2.2	Program structure.....	21
6.2.3	Algorithms.....	21
6.2.4	Main Data Areas.....	21
6.2.5	Files.....	22
6.2.6	Interfaces.....	22
6.2.7	Suggestions For Improvements.....	22
6.2.8	Things to watch when making changes.....	22
6.2.9	Physical limitations of the program.....	22
6.2.10	Rebuilding and testing.....	22
6.3	Personal reflective reports.....	23
6.3.1	Stoyko Kodzhabashev.....	23
6.3.2	Nathan Hand.....	23
6.3.3	Daniel Cornwell.....	24
6.3.4	Adrian Gawryszewski.....	24
6.3.5	Miles Warburton.....	24
6.3.6	John Bolton.....	24
6.3.7	Sam Clements.....	25
6.3.8	Daniel Stuessy.....	25
6.3.9	James Barnett.....	26
6.4	Project Plan.....	27
6.4.1	Diagrams.....	27
6.4.1.1	Account Management Use Case Diagram.....	27
6.4.1.2	Offers Use Case Diagram.....	28
6.4.2	Gantt Chart.....	28
6.4.3	Event Timeline.....	29
6.4.4	Risk Assessment.....	31
6.5	Design.....	33
6.5.1	INTRODUCTION.....	33
6.5.1.1	Purpose of this document.....	33
6.5.1.2	Scope.....	33
6.5.1.3	Objectives.....	33
6.5.2	DOCUMENT CONTENT.....	33
6.5.2.1	Decomposition Description.....	33
6.5.2.2	Monster Mash Significant Classes.....	33

6.5.2.3Monster Mash Package Descriptions:	34
6.5.2.4Monster Mash Application Requirements Mapping	35
6.5.2.5Dependency Description	36
6.5.2.6Interface Description	37
6.6User interaction design	42
6.6.1INTRODUCTION	42
6.6.1.1Purpose of this document	42
6.6.1.2Scope	42
6.6.1.3Objectives	43
6.6.2DOCUMENT CONTENT	43
6.6.2.1Account management	43
6.6.2.2Register/Unregister	43
6.6.2.3Login/Logoff	45
6.6.2.4Offers	47
6.6.2.5Friend Requests	47
6.6.2.6Fight request	47
6.6.2.7Breed	49
6.6.2.8Trade	49
6.6.2.9Notifications	51
6.6.3Accept an offer	51
6.7Other Views	52
6.7.1Home Screen	52
6.7.2View Friends Monsters	52
6.7.3Profile View	53
6.8Test specification	53
6.8.1INTRODUCTION	53
6.8.1.1Purpose of this document	53
6.8.1.2Scope	53
6.8.1.3Objectives	53
6.8.2DOCUMENT CONTENT	53
7REFERENCES	61
8DOCUMENT CHANGE HISTORY	61

1 Management Summary

Overall the project did achieve quite a lot. The functionality of the back end of the application worked well and the algorithms and methods used to implement them were put together well.

The only main problem being that the functional parts of the program on the back end were not linked with the website so that the application could be run from a browser and therefore be played by the end user.

However the registration and login parts of the website worked fine with the required validation to ensure that no spam accounts could be set up and the required field types were used.

The project documents are in a good state and both the project plan and design specification have been given a grade indication of C. This of course can be better and therefore amendments to these documents have been made based on the feedback given to bring them up to a better standard ready for final submission.

During the project there were a few problems that the group ran into. The first being that certain members had work outside of university and trying to allocate and structure project work around this fairly was difficult.

Also some of the coding some members found difficult and therefore had to be redistributed to others so that it could be amended by the end of coding week ready for submission.

There were problems communicating effectively with other groups at the standards meetings as most couldn't come to an agreement on how things should work which therefore led to a delay in getting things ready for the implementation phase of the project.

The problem with the standards was overcome by working closely with a couple of groups only to come to a solution on what standards to use so that the games could work together on different servers. The project never actually got to this stage but this difficulty was still overcome with this in mind at the time.

Overall the group worked well but could have worked better. Certain members found certain tasks difficult and needed help at times. Other members had work and commitments outside of the department and therefore the group had to make the most of the time available to get as much of the work done as possible.

2 A historical account of the project

Main Event	Date	What took place and actions of project members
Arrange project team positions	08/10/12	There was a group meeting and we decided as a group who was suitable for each position ready for the project ahead.
Introduction and details of the proposed system	08/10/12	The entire team was introduced to the project and were told to research the task ahead.
First part of project documentation to be delivered	15/10/12	Certain members in the group were given the task of putting together the use case design diagram for the project. Others were given the job of putting together the user interface design documentation, Gantt chart, project plan and risk analysis.
First deliverable	29/10/12	The above documentation was delivered by this date.
Diagrams	29/10/12	A class diagram and component diagram were made to outline the structure of the projects code.
Test identification	29/10/12	The entire team had the job of identifying tests for the system so that once the application had been built certain tests could be run for quality assurance purposes.
Standards meeting	29/10/12	Certain members of the group met with the other groups to discuss standards and come to a decision on what standards were to be used for the project.
Test Specification	29/10/12	To create the test specification for the final system so that it could be delivered.
Allocating work for coding week	29/10/12	The project manager split up the workload so it was fairly distributed to members of the project for coding week including documentation.
Skeleton code	05/11/12	Those given the task of putting together the skeleton code did so ready for implementation.
Standards meeting	12/11/12	Another meeting to discuss standards again due to confusion from the first meetings over certain data types.
Second deliverable	12/11/12	The test specification was delivered by this date.
Architectural and interface description	12/11/12	Certain task members were given the task of putting together the architectural description and interface description of the application.
Start of coding	12/11/12	The code was starting to be implemented at this point so that it could get up to a required standard before integration and testing week.
References	26/11/12	References and appendices were put together by the group.
Standards meeting	26/11/12	Another meeting with the other groups took place to

		make sure that all were up to speed on what standards were being used.
Third deliverable	03/12/12	The design specification was delivered for the proposed system.
Prototype demo submission	10/12/12	Prototyping software was researched and used to create a prototype of the application and submitted.
Integration and testing week	28/01/12	<p>During the week the group members had tasks allocated to implement in terms of code and to assist other members in completing work. The basic server was set up, tomcat installed and configured as required. Database communication was also set up.</p> <p>The account management and details for the accounts were implemented. The fighting code was implemented using the required methods.</p> <p>The JSP pages were created and put on the server to be seen and validation tests were used on forms to make sure on the required data types could be used.</p> <p>Testing was done during this week as well on as many issues as could be found up to the point where the code was at.</p>
Fourth deliverable	03/02/12	The fourth deliverable was to deliver the software to the client.
Final deliverable	18/02/12	All documentation submitted as required on the given date.

3 Final state of the project

3.1 *Working Features*

- Database Storage - Java classes that write and read information into/out from the database work. We are able to establish connections successfully to our SQL database. We have to code that can create new rows, get rows out and turn them into classes and code to edit single cells of the database.
- Website - We have jsp pages that can display all necessary web pages needed for our game.
- Client Classes - We have client classes that are able to pull information for a certain directory. These can be used to get other users and their monsters.
- Monster breeding and fighting - We have a monster fighting algorithm that can take 2 monsters and determine a victor. We also have a constructor that takes 2 monsters and create a new one based on their attributes.
- Player register and login - We can register a player and login with it using our database connectors and web pages.

3.2 *Unimplemented features*

- Server to server communications - Aside from pulling information from the directory we were unable to get our program to have complete server to server communication. We have servlets that can handle requests from other servers however we were unable to test them.
- Monster Mash gameplay - As a result of the above, we were unable to get our application to do things such as adding friends, battling, trading, breeding etc. We can only register and login. Most of the code for it is contained in the servlets classes, but as said above these were not tested.

4 Performance of each team member

4.1 Daniel Cornwell (Deputy Project Manager)

Daniel's duties were involved around coding, initially this was regarding the database connection code and interaction over Java and then later his work was primarily over the API/ JSON connections between our server and other servers. Each of these tasks required a substantial amount of research as these tasks were never something that either of us had done previously within Java.

Daniel never expressed interest with his position of Deputy Project Manager, however this position was only filled in the case that the primary Project Manager was away/unable to attend and run meetings/fulfil his duties. So while Daniel didn't undertake any project management tasks it was only because they weren't needed throughout the project.

Daniel's performance for the project was great and generally helpful to myself and to other project members. He was consistently a great worker for the group project and every task he was given was completed, and even if something wasn't working as it should in the code, it was a very close implementation of what we required.

Daniel turned up to almost every meeting, missing perhaps 1 meeting for which he had informed me beforehand. Overall he was a great member of the team with a superb performance.

Agreed with group member? (yes or no) if no why?

Agreed with Daniel

4.2 Daniel Steucy

Daniel's duties were quite varied, as he was required to be pulled away from a certain task if that task required more work or if it appeared to be too challenging. His primary tasks involved documentation as well as writing the JUnit tests that our project can be tested against. During implementation and testing week Daniel was working with Daniel Cornwell with the API as well as various JSP-related tasks.

Daniel's production of work left a lot to be desired, he failed to complete certain work that was asked of him and would often forget that he was given a task to complete, or to update a certain document. This deficit in productivity was likely due to simple forgetfulness, and because he attended the meeting he didn't think to read the minutes to remind himself of any tasks.

Towards the end of the project, during work week, I felt he really tried harder to help with work and generally get aspects of the project working. So while I was pleased with his effort and work during implementation and testing week, that level of work and dedication could have really been helpful to the entire project rather than just in the final hurdle regarding the code/ implementation of code.

Agreed with group member? (yes or no) if no why?

Agreed with Daniel

4.3 Adrian Gawryszewski

Adrian's duties involved certain aspects of documentation as well as getting the basic idea of a HTML and CSS set-up for the front end of the website. He also took part in JSP conversion of his HTML files into JSP files.

Adrian's production of work was relatively consistent and the work he produced was generally fine, however we had

real problems with the work he created being integrated with the project. This problem completely stems from the fact that Adrian found it impossible to get Github working with his version of the Mac operating system which unfortunately made all of his work hard to implement or get updated between all members of the group project. This was a problem during the main development stages of the group project but it became a complete mess during coding week, where Adrian was working on code but it was either outdated from small updates made by other team members or generally hard for us to get a hold of that work to continue development.

Regardless of these problems Adrian was a very helpful and enthusiastic worker, although I feel that due to his prior engagements that he couldn't change (his job) he was unable to attend several meetings. Although this is not his fault, it did create problems between him and the group - problems that could have potentially been avoided with better minute-taking and uploading.

Agreed with group member? (yes or no) if no why?

Agreed with Adrian

4.4 Stoyko Kodzhabeshev

Stoyko's duties involved coding of various aspects of the project, focusing on certain algorithms such as fighting as well as writing JUnit tests for the project and his own code/algorithms. He was also responsible for large amounts of the initial Java code, setting up the classes and making the “skeleton” code for the entire project. During implementation and testing week Stoyko was very active in the updating of various tests as well as monster constructors in the set-up for changes in the SQL database and API updates.

Stoyko's performance for the project was helpful and at a good level. His coding ability was definitely one of the strongest in the group, however I don't think he fully understood the project which made all work that he did limited in quantity, or I didn't feel that he could take some initiative and would ask me for work constantly never surpassing/improving his implementation of the work that he was given. So while his performance was great when he had work to do, he was always asking me for tasks, doing them to the letter and never taking any initiative and building upon his work.

While I agree that this could also be the fault of the Project Manager not being able to give him more detailed instructions, group members are expected to try and take some initiative and not wait around for the Project Manager's input at every turn.

Agreed with group member? (yes or no) if no why?

Agreed with Stoyko

4.5 Sam Clements (Deputy QA Manager)

Sam's duties were mostly to do with certain design aspects regarding the entire project folder layout, GitHub “Guru”. This involved generally fixing the problems that we had with the GitHub when we had conflicting code uploads (which was a common problem) as well as other design additions during the project. Sam was also responsible for other sections of coding such as breeding, and he also seemed to inadvertently take an active role in general code “maintenance”, making sure things were tabbed properly and that they met the coding standards. While this was done well, this was the task of the QA Manager, therefore I feel that his time and skills could have been more focused on codes of greater complexity. Sam seemed to have a good idea of how this project would come together and how certain aspects should work based on him doing them in personal projects previously.

Overall Sam's performance was decent and steady, however as he seemed to be one of the best coders of our group, in my opinion he could have done more with the tasks that he was given, or he could have been more active in the harder to code sections of the project. Sam would seem to take it upon his self to do work for which he was perhaps “over

qualified”, the example being the cleaning up of code during coding week. As I've said, while this was under his deputy QA Manager title, I feel his skills could have been put to better use.

Sam attending almost all of the meetings, missing only a few, and he was generally a great input to the team, but I do feel he could have done more complex tasks as he was competent enough to do so more easily than others in the group.

Agreed with group member? (yes or no) if no why?

Agreed with Sam

4.6 Miles Warburton

Miles's duties involved documentation from the beginning as well as various research tasks. His work was not really focused on much coding, if any, but documentation and help in the design/database side of the project in its early stages. Later on in the project, work revolved around some simple JavaScript implementations as well as other small tasks regarding JSP pages.

His performance for the entire project seemed slow, or as if it was lacking interest or effort, as things would get done but nothing was done as quick as I could have expected from certain other members of the group. I feel that Miles was very quiet in all the meetings and generally didn't pay much attention or seem particularly interested during the meetings/projects.

With him being very quiet and unsure of himself or the work he might have had to do, I feel it's possible that Miles didn't understand all of the work that he was given, however he never voiced this problem.

Agreed with group member? (yes or no) if no why?

4.7 John Bolton (QA Manager)

John's duties involved being the QA Manager (taking minutes at the meetings, checking over others' work, etc.), so he had a full time job being a QA Manager and was not expected to carry out any other tasks by myself. He did however volunteer to do certain things before his work load properly started, this included helping in some research tasks as well as some initial JUnit tests.

John's performance during the entire project started out really well; every week he would turn up to meetings and take the minutes and then upload them. Generally John was a very good QA Manager at the start of the project. However as the project went on, John's note-taking and attendance to some meetings worsened - even if he attended the meetings and took the minutes there were instances when these minutes didn't appear on GitHub or via e-mail.

John's job as a QA Manager seemed to dissolve as the project continued. He became less of a QA Manager and more a regular person looking to help in aspects of coding. While I allowed John to continue this, a failure of my own responsibility was that I didn't really notice until too late that his work as a QA Manager was somewhat deficient. John had a requirement to check over people's work as it was done and to try to point out flaws in the documents and coding as well as generally suggest possible improvements.

None of the documents that were uploaded to a deliverable seemed to have been checked by John, however he cannot be blamed for this lack of work entirely as work was late to be uploaded to GitHub on regular occasions by various group members. John's overall performance therefore didn't resemble much of a QA Manager, but more of someone looking for something to do for the group while his own work was neglected during most of the project as a whole.

Agreed with group member? (yes or no) if no why?

Agreed with John

4.8 James Barnett

James's duties revolved around research, documentation and some very minor amounts of JavaScript/HTML help during implementation and testing week. His primary role has always however been in documentation as it's one of his stronger areas compared to his programming.

James's performance for the group project as a whole was very good. He was able to get done to a suitable standard work and anything else that was asked of him. He was constantly in contact with myself generally trying to make sure he was doing the correct work and asking me small questions or for my opinion on certain things. James has been rather helpful during the project for a number of documentation or research-related tasks. Unfortunately due to his lack of knowledge regarding some of the coding this did make some of the documentation hard for him to do.

Overall James had a solid performance throughout the project, being at the meetings rather often as well as giving his input as often as possible. On the other hand, I feel that he would often sell himself short in his ability and his lack of confidence would make me reluctant to give him harder tasks or to give him too much at once, which did make working around his (perceived) ability a task in itself.

Agreed with group member? (yes or no) if no why?

Agreed with James

4.9 Nathan Hand (Project Manager)

My tasks were focussed on being the Project Manager. While this took up a large amount of time I also handed myself tasks such as working on documentation when a group member was sick or unable to complete their work as well as setting up the entire remote Tomcat/SQL server, as I had access to those kinds of resources. I also played a large part in the JSP login/registration scripts for the project.

My performance as a Project Manager was good but I believe I could have done much better at many aspects of the project management position. I always knew what people were doing and why they were doing it, however this might not have been clear to everyone else at any one time. Thus if I was to become sick for any reason, activities such as the project meetings and general work allocation would have been very hard for my Deputy Project Manager to pick up on.

My overall performance was a very active one and I would often make sure that people had something to do and that they knew exactly it meant to do such things. However over the Christmas period during which I had set a number of tasks for all group members, almost nothing was done during this time. I am aware that I could have been more proactive over the Christmas period with regards to making sure everyone was actually doing work. Plans regarding any kind of meeting didn't happen due to 0 activity on GitHub, and I felt at the time that they knew what to do and as such nothing had changed since before Christmas. This was wrong and I should have been more active and on top of the project to make sure that we were much more prepared before implementation and testing week.

Agreed with group member? (yes or no) if no why?

Agreed with Nathan

5 Evaluation of the Team and the entire project

5.1 How did the team perform as a whole, and how could that have been improved?

The team as a whole performed well together, there were no major problems with the team members working together as a whole. There were no major personality clashes with the team and we worked together well with no major problems between team members.

The team however as a whole all seemed to be under the same speed of work, which generally lead to last minute working very hard rather than staggering work over an entire week or holiday period. Because the entire team never seemed to be working on the project pro-actively this has lead to various problems/ rushed jobs.

This problem could have been improved if the project leader was perhaps more active in the idea of handing out yellow or red cards or being quicker to point out the teams obvious flaws when dealing with deliverables.

5.2 How could the project have been improved?

The project as a whole could have been improved in a number of different areas, for example lots of the team felt that all of the QA documents that related to the program where written badly and hard to understand. The fact that we had to produce a working program based on these documents didn't make the process very easy.

The API for the project was a complete mess, the project required a certain degree of working together between groups to come up with a decent API. None of this requirement was met as things were being decided too late to get a proper implementation working by the end of the project.

Overall the project could have been improved with the better production of QA documents to make sure that the team had a greater understanding of what was required for the program. If changes where made to the QA docs these were not posted for the team to see but were merely expected to be noticed. An example of this was the changing of monster breeding from having a gender (male or female) to being asexual where this variable was no longer accounted for.

5.3 What were the most important lessons learned about software projects and about working in teams?

This project has taught the team many things about working in teams and working on software projects. Some of the most important lessons learnt included:

- The importance of keeping decent note-taking
- Knowing the strengths and weaknesses of team members abilities
- Decent version control, including knowledge on how to use them effectively
- Time management

6 Appendices

6.1 Project test report

Register/Unregister

Test Ref	Req. being tested	Text Content	Input	Output	Pass criteria	Pass/fail
T-SN-01	FR6	Check if “register” button works.	Press register button on a main page.	A new page with fields to fill in e.g. username and password occurs.	Redirection to a new page.	Pass
T-SN-02	FR6	Check if system will register an account without filling in any data on the register page.	Enter no data on the registration page and then press “continue” button.	Error message warns of missing data.	Error message displayed on a page. Account not added to a server.	Pass
T-SN-03	FR6	Check if system will register an account with some data missing e.g. password.	Fill in all fields beside password field and press “continue” button.	Error message warns of missing data.	Error message displayed on a page. Account not added to a server.	Pass

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
T-SN-04	FR6	Check if system will register an account when confirm password field differs to password.	Fill in all data., but misspell “confirm password” field, so that it differs to password.	Error message warns of incorrect data.	Error message displayed on a page. Account not added to a server.	Pass
T-SN-05	FR6	Check if system will register an account when login already in use.	Fill in login that is already in use on a registration page and click “continue” button.	Error message warns of existing account.	Error message displayed. Account not added to a server.	Pass
T-SN-06	FR1	Check if system will register an account when	Fill in all the data properly and press “continue”	A new window occurs, telling that account has been added	New window with positive message occurs.	Pass

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
		all the data is filled in correctly.	button.	successfully.	Account added to a server.	
T-SN-07	FR6	Check if “unregister” works.	Go to MyProfile and press “unregister” button.	A new page occurs with some fields to fill in e.g. username, password.	Redirection to a new page.	Fail – we wrote a servlet that should deal with unregister, but it doesn't work properly.

Login/Log off

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	
T-SN-11	FR1	Check if system lets user login without username and password	Fill in nothing and press “login” button.	Error message warns of incorrect login or password.	Error message on a page. Login fails.	Pass
T-SN-12	FR1	Check if system lets user log in, if one field is empty.	Fill in only one field. Press “login” button.	Error message warns of incorrect password or login.	Error message on a page. Login fails.	Pass
T-SN-13	FR1	Check if system lets user login when password is incorrect.	Fill in both fields, but misspell password. Press “login” button.	Error message warns of incorrect login or password.	Error message on a page. Login fails.	Pass
T-SN-14	FR6	Check if system lets user log off when “log off” button is pressed.	When logged in press “log off” button in the top-left corner.	The home page occurs.	Home page occurs. User logged off.	Pass

Offers:**Assumptions:**

To perform some of the tests below we have to assume, that we have created at least two accounts. It will help us to inspect interactions between accounts and check, if the fight engine works properly. To make it easier we will create accounts test1 and test2.

Add/remove friend

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
T-SN-15	FR9	Check if system sends an “add friend” request. This requires to check if target friend's details are correct.	Go to test1 and in the “add friend” field type in test2(friend you want to add) and press “add friend” button.	Test1: Message occurs saying that message was sent. Test 2: Add friend request occurs in offers window.	Request sent from test1. Request received on test 2.	Fail – we fail to write an appropriate code and implement it.
T-SN-16	FR9	Check if system adds a friend when request declined.	Test2: when request occurs in notification window press “decline”.	Test1: Message occurs saying that your request was declined.	Test1: Message occurs. Both: No friend added to a friends list.	Fail – as above
T-SN-17	FR9	Check if system adds a friend when request accepted.	Test2: when request occurs in notification window press “accept”.	Both: Message occurs saying that a friend was added to a friends list.	Both: Message occurs. Friend added to a list.	Fail – as above
T-SN-18	FR9	Check if system adds a friend when “add button” is pressed, but request isn't accepted or rejected.	Test1: send a friend request. Test2: take no action.	Test1: Message occurs saying that message was sent. Test 2: Add friend request occurs in offers window.	Test1: Message occurs. Test2: Request received. Both: No friend added to a friends list.	Fail – as above
T-SN-19	FR6	Check if system deletes a friend from your friend list, when “remove a friend” button pressed.	Test1: Go to your friends list and remove test2.	Both: Message occurs saying that a friend was removed from a list.	Both: Message occurs. Friend removed from a list.	Fail – as above

Fighting.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
T-SN-20	FR6	Check if “fight” button works.	Test1: Go to your friends list and choose test2, pick one of monsters and press “fight”. Test2: Take no action.	Test1: Message saying, that request has been sent occurs. Test2: Request occurs in notification window.	Test1: Message occurs. Test2: Request occurs. Both: Any other action isn't taken.	Fail – we fail to write an appropriate code and implement it.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
T-SN-21	FR10	Check if system lets to fight without acceptance.	Test1: Pick one of friend's monsters and click "fight". Test2: Take no action.	Test1: Message saying, that request has been sent occurs. Test2: Request occurs in notification window.	Message displayed. Request sent. No other action is taken.	Fail – as above
T-SN-22	FR10	Check if system lets to fight when request rejected.	Test1: Pick one of friend's monsters, press "fight" Test2: Reject an offer.	Test1: Message saying, that your request has been rejected occurs.	Fight hasn't been taken.	Fail – as above
T-SN-23	FR10	Check if system lets users fight when request accepted.	Test1: Pick one of friend's monsters, press "fight" Test2: Accept an offer.	Both: Message with a result is displayed saying who has won and lost and what is a reward.	Message occurs. Fight has been taken.	Fail – as above
T-SN-24	FR10	Check if winner receives a reward.	After a fight: Test1: If won inspect your balance. Else if test2 won: inspect your balance.	After a fight: Both: Message with a result of fight occurs.	Winner's balance changed. Loser's balance didn't change.	Fail – as above
T-SN-25	FR10	Check if after a fight monster's health is decreased.	Test1: Proceed to fight. After that, inspect monster.	New window with monster's statistics occurs.	Monster's health is decreased.	Fail – as above

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
T-SN-26	FR10	Check if after monster is injured an injury status occurs in its profile.	Both: Inspect your monster after every fight and check, if it was injured afterwards.	In the result window a message saying that your monster was injured during a fight.	When monster injured – injury notice appears in its statistics.	Fail – we fail to write an appropriate code and implement it.
T-SN-27	FR10	Check if after monster is dead system removes it from a monsters list.	Proceed to fight till you monster dies.	Message saying your monster is dead.	Message occurs. Monster removed from monsters list.	Fail – as above
T-SN-28	FR10	Check if system	Proceed to fight till your	Message occurs that	Message occurs.	Fail – as above

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
		generates new monster, after your last monster died.	last monster is dead.	your last monster is dead and new one was generated.	Monster removed from a list. New monster generated and added to a list.	

Breed:

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/fail
T-SN-29	FR3	Check if “breed” button works.	Test1: Press “breed” button. Pick your monster and insert a price. Test2: Take no action.	Test1: Message occurs saying that your breed offer has been sent. Test2: Offer occurs in notification window.	Test1: Message occurs. Test2: Offer occurs in notification window.	Fail – we fail to write an appropriate code and implement it.
T-SN-30	FR3	Check if system will allow to breed without an offer being accepted by any other player.	Test1: Press “breed” button. Pick your monster and insert a price. Test2: Take no action.	Test1: Message occurs saying that your breed offer has been sent. Test2: Offer occurs in notification window.	Test1: Message occurs. Test2: Offer occurs in notification window. Both: Monster isn't generated.	Fail – as above

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/fail
T-SN-31	FR3	Check if system will allow to breed, when offer accepted.	Test1: Proceed to breed. Test2: Accept an offer.	Test1: New monster occurs in a list. Test2: Message saying that breeding was successful.	Test1: New monster, balance decreased. Test2: Message occurs, balance increased.	Fail – as above
T-SN-32	FR3	Check if you can accept the same offer more than once.	Test1: After offer has already been accepted. Take no action. Test2: Try to accept same offer again.		After first acceptance offer disappears from a notification window. No other action is	Fail – as above

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/fail
					taken.	
T-SN-33	FR3	Check if new monster has a value and statistics.	Test1: Proceed to breed. After accepted inspect new monster. Test2: Accept an offer.	A window with monster's statistics appears.	New monster has value and stats.	Fail – as above

Trade:

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
T-SN-39	FR8	Check if “sell monster” button works.	Test1: Go to your monster list, pick a monster and press “sell monster” button. Test2: Take no action.	Test1: Message saying that offer has been sent to all friends occurs. Test2: Offer occurs in a notification window.	Test1: Message occurs. Test2: Offer occurs in notification window.	Fail – we fail to write an appropriate code and implement it.
T-SN-40	FR8	Check if system allows to sell monster without any other player's acceptance.	Test1: Proceed to sell a monster. Test2: Take no action.	Test1: Message saying that your request has been sent. Test2: Offer occurs in notification window.	Test1: Message occurs. Test2: Offer occurs in notification window. Both: Nothing else happens.	Fail – as above
T-SN-41	FR8	Check if system allows to sell a monster when offer accepted.	Test1: Proceed to sell monster. Test2: Accept an offer.	Test1: Message saying that your offer has been accepted occurs.	Test1: Message occurs, monster removed from a list balance increased. Test2: Monster appears on a monster list, balance decreased.	Fail – as above
T-SN-42	FR8	Check if offer disappears when it has already been accepted.	Test1: Proceed to sell. Test2: Accept an offer. Inspect a notification window in order to accept	Test1: No offer in notification window. Test2: No offer in notification window.	After offer accepted once it disappears from a notification window.	Fail – as above

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/Fail
			same offer again.			

For all offers.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	
T-SN-43	FR8	Check if offer has deactivated after monster died.	Test1: Fight until monster dies. Test2: Fight test1's monster until it is dead.	Both: Offer referring to the dead monster are not in the notifications window.	Both: After monster is dead all referring offers are deactivated.	Fail – we fail to write an appropriate code and implement it.

Views: (user interaction design document = UIDD)

Test ref	Req. being tested	Text content	Input	Output	Pass criteria	Pass/fail
T-SN-44	FR7	Check the login page loads when the homepage is requested via URL.	Test1: Enter homepage URL in browser.	Test1: The login view as shown in the UIDD loads in the browser.	The login page loads correctly with all the login and sign up features	Pass
T-SN-45	FR8	Check the monster list appears after login.	Test1: Login to game.	Test1: The user is presented with the home screen view as shown in the UIDD.	The home screen displays correctly with all of its features as shown in the UIDD.	Fail – we fail to write an appropriate code and implement it.
T-SN-46	FR8	Check if a friend's monster list appears by request.	Test1: Click on a friend in the friends list at the left.	Test1: A list of all the monsters the friend you clicked on owns, as seen in the UIDD.	The friend's monster view as shown in the UIDD is displayed correctly with all of its features.	Fail – as above
T-SN-47	FR8	Check if a user's profile displays by request.	Test1: Click on user in friend list.	Test1: The user's profile information should appear.	The profile view as shown in the UIDD is displayed correctly with all of its features.	Partly pass – not all of player's data are being displayed

T-SN-48	FR8	Check if the notifications page displays correctly and by request.	Test1: Click on the notifications page button/link.	Test1: The user's notifications about offers, battles, and other relevant in-game information the user has to know, are displayed.	The notifications view as shown in the UIDD is displayed correctly with all of its features.	Fail – as above
T-SN-49	FR8	Check if the 'accept offer' view displays correctly and by request.	Test1: Click on an offer in the notifications page.	Test1: The necessary features of the 'Accept Offer' view from the UIDD are displayed, allowing the user to accept or decline an offer.	The 'Accept Offer' view as shown in the UIDD is displayed correctly with all of its features.	Fail – as above
T-SN-50	FR8	Check if the 'choose monster' view displays correctly and by request.	Test1: Click on a battle request in the notifications page.	Test1: The 'Choose a Monster' view is displayed, allowing the user to choose which monster to fight the opponent's.	The 'Choose a Monster' view as shown in the UIDD is displayed correctly with all of its features.	Fail – as above

6.2 Project maintenance manual

6.2.1 Program description

This program is a game that is designed to teach about evolution. Each user has monsters that they are able to breed, trade and fight with. These monsters have several statistics that vary in strength over their lifetime. Monsters will eventually die of old age. The user will only be able to interact with other users that they have added to their friends list. Fights will result in the death of the losing monster and the winning user will receive a currency reward. Users are able to purchase monsters from friends or offer their own up for sale. Two monsters may be bred to create an offspring that will have statistics similar to their parents. A user may offer a monster for their friends to breed with for a cash price. The objective of the game is to collect as much money as possible.

6.2.2 Program structure

The program, while not fully completed, has several significant components:

- The data structures -- `aber.dcs.cs211.group07.data` -- which contains a large number of classes detailing the various objects that the program uses or had plans to use (for instance, the Request and Offer classes ended up unused).
- The database classes, and the database itself. This includes a few classes designed to make connecting and using the tables easier (MonsterTableConnector, PlayerTableConnector). These were not in the original design, but were created during the project as a quicker way to handle the SQL connections than using individual SQL command across the codebase -- basically, a very rudimentary ORM (Object-Relational Mapper). The database itself was a MySQL db, with the tables setup manually using PHPMyAdmin. However, the original SQL commands to setup the database can be found in `src/monstermash/WEB-INF/classes/aber/dcs/cs211/group07/db` under the names `monster_mash.sql` and `monster_mashupdated.sql`.
- The JSP/HTML front end. This includes the various pages that were done as part of the front end. Multiple versions or styles exist, mostly due to miscommunication or lack of communication in the group. These are found under `src/monstermash`, as well as in the `servlets` package. Due to difficulty understanding and running tomcat -- especially under mixed development environments, a lot of this couldn't be tested locally by the authors and so had to be published to the live server to test this.
- The servlets back end. This includes most of the servlets, which were intended to handle the inter-server API. This was one of the least finished pieces of the program, due to having unclear and changing requirements through the project -- it was only during the final weeks that an API standard was finalized. It was also difficult to test without having a clear API or other servers to test against, which resulted in a lot of the code remaining untested.

6.2.3 Algorithms

Refer to the design document 6.5

6.2.4 Main Data Areas

The data structures are defined in the Design Specification, under the heading "Monster Mash Significant Classes".

6.2.5 Files

Our program used no fixed or created files, as all data was stored in an SQL database.

A dump of the SQL database from the instance of our application can be found in `sql/handyman_monster.sql`.

6.2.6 Interfaces

Refer to the design document 6.5

6.2.7 Suggestions For Improvements

Most of the coding was done to the design specification, but as for actually implementing it we only managed to get a few things working. This was mainly down to not sticking to the agreed time line that was set. Communication was also another big issue that attributed to the project not reaching fruition, as there were problems with some group members not having proper access to Github, which when it came to merging peoples work, the system had huge problems. There was also other complications during the development period, which is that we only had access to one production server. It had been mentioned that it would be ideal if everyone could set-up their own development environment for the project just in case anything unfortunate was to happen, which ultimately problems did arise -the tomcat server missing various libraries, not having full access to the MySQL database and the university network going down.

Basically if we were consistent with what we discussed during group meetings, our project could have been in a much better condition.

6.2.8 Things to watch when making changes

Make sure to read the documentation regarding which API you're using/going to use as this is very important for implementing server to server communication.

If you're going edit the database, make sure that it is reflected in the source code, as this program relies heavily on a MySQL back end to store its data.

6.2.9 Physical limitations of the program

Our system hasn't been stress tested, so it is uncertain how it will perform during times of high traffic.

6.2.10 Rebuilding and testing

When rebuilding the project it is important to run the JUnit tests first. They can be located in the source code of the project can be run easily through any IDE if you have the needed library. If any of the tests fail you will be given a message saying which test failed and information on what this test was testing. They are checking all the constructors of the project and whether the algorithms in them are working with the current version of the project. If all the JUnit test pass all changed or updated files need to be uploaded to the server which is running the project. If there are any changes to the database or the way the information is stored the actual database may need some changes. After setting up the project you need to run all the tests described in the 'Test specification' document. Each of them have input and output described as well as a pass criteria which describes whether each of the tests have passed or not. If new tests are needed depending on the changes made this should be another JUnit test or a test described in the 'Test specification' document. If it's a JUnit test related to a new class a new JUnit class should be created and it should be testing all the functionality of the new class. If it's a test for the 'Test specification' document the test need to be included and described in the appropriate section of the test table.

6.3 *Personal reflective reports*

6.3.1 Stoyko Kodzhabashev

At the beginning of the project there were many discussions about what had to be done and how we were going to approach the group project. At the meetings we were splitting things that have to be researched in order for the group project to be done. We also got our roles in the group project. I was given a role of a coder and my task was to write the skeleton code of the project and later on to add the implementation of the Monster and Fighting class. During the coding week I took upon writing the JUnit test also because the person responsible for them didn't submit any in time. During the coding week there were a lot of problems with little things for example the fact that we kept changing the variables a monster is going to have and the database to store them till the very last day. Because of these changes there was always someone who had to update or re-write some of the code we already had. We couldn't connect to the database at all and it made some of the work impossible to do as long the person responsible for the database connection fixed it.

Some of the group members really struggled with setting up version control (GitHub) or working with it and even after getting help and making it work they managed to break it. This was really frustrating for the whole group as we could not see what that person is doing or having him send his code through the email and us having to find what are the differences in the classes.

I believe that there is always room for improvement like everyone attending to the meetings or at least having an excuse. Most of the members didn't even bothered to turn to the meetings and even skipped the one that was right after the exam we all had. We also agreed to work during the winter but almost no one even added their coordinates (Skype for example) so we can catch up to the work if someone actually did some.

There are a lot of things I learned about during this project, the most important of which was team work, the importance of communication and how important the group you will work with is and to do your task without delaying them because this sometimes slows or even interrupts the work of some of the members of the group.

As a total I believe that the group could have performed much better if everyone had put more effort into the project and if we had set up our own variables not trying to match the variables of other groups in order to be able to do server-to-server communication when we weren't even able to communicate with our server.

6.3.2 Nathan Hand

The project as a whole started off really well, members of the team seemed to be involved and really were excited to start work. However as the project leader I think that I noticed more then others a continued decline in effort from certain team members. This would be shown in the form that every meeting that we had after the first couple always had at least one team member missing, typically over 1/2 the team would not be in attendance. This meant that team meetings seemed to be far less productive as we might have been confused about certain workers input for the project during the week and needed to ask them about it.

The project as a whole has taught me many things, this was the first time I had been a project manager for any kind of major project such as this. My time as a project manager has taught me that it's a difficult task to keep on top of everything and everyone. Quite often feeling large amounts of pressure when things are due in soon and not as much is done as I would like to have seen by that point. However I feel that being a project manager has improved my interpersonal skills as well as time management and generally has given me a much greater understanding of what I'm likely to see in a real world environment.

Overall I look back at this project as a great learning experience, although it was frustrating and challenging at times this project has taught me many things.

6.3.3 Daniel Cornwell

During this project I was tasked to do things I was not familiar with, and hence learnt new skills. I have learnt a lot about databases, (since I did not do the database module during first year), such as how to write SQL commands and design code in Java to write into databases. I have also learnt about how to get information using web services, such as servlets and clients.

I have also developed my team working skills. I was not familiar in working in such a large group as well as working in such a professional manner. I feel this project has taught me a lot about what is required when working on designing software.

6.3.4 Adrian Gawryszewski

As a part of my work was to work on two documents and HTML/CSS template. Documentation part referred to User Interaction Design and Testing Specification. In first of these two I was responsible for a description of any action player can take during the game. In relation to this part I need then to write appropriate tests to check these actions are working in our program.

After that I started working on a HTML/CSS template. This includes some basic graphics, created and modified using Photoshop. During coding week I was working on register/unregister, login and monster generator (didn't work in the end). I wrote some simple servlets based on Daniel Cornwell's code accessing and dealing with databases.

This exercise showed me how hard it is to work in a group. I realized that this is something totally different than building an application from a scratch on my own. It takes more time and requires good communication skills. Another thing I learnt is how important in this whole process was the actual design and specifying what we want to build. Before that I thought documentation is a waste of time or that writing testing plan before having something working is rather not the best idea. I thought the most important part was to sit down and start coding to produce a solution (game).

This project gave me lots of new experiences and even without feedback from the lecturer I think I know what mistakes I've made and what to do to prevent them in a future.

In the end I would just like to add that I realize we failed as a group, but in the same time we got a decent lesson and we won't make same mistakes in a future working on real projects.

6.3.5 Miles Warburton

I felt that I had struggled quite a lot with this group project as it took quite a large endeavour to work with people of varying programming competency. I improved a bit on my java coding, but not by much as I had only worked on the database side of things.

I didn't gain a positive experience from this group project and I found it quite difficult to manage the work over a huge team. I do however; appreciate that I did get a chance to experience what it would be like to work on much bigger software engineering projects.

6.3.6 John Bolton

I found this project to be quite challenging. I had the role of quality assurance manager, so I was supposed to be checking through all work that was done by everyone in the group looking for any errors and making sure that all of the work was of good quality, as well as taking the minutes in meetings. I found it difficult to perform my job properly as most people were either not using github or uploading their work only when they were finished shortly before the deadline, not allowing me enough time to check the documents before the submission deadlines.

During implementation and coding week, I found that I was getting sucked into helping individual people debug their code, and I lost sight of the project as a whole. A lot of people were either not using github properly or not using it at all, which made it hard for me to check their work. I ended up spending most of the week trying to help some people getting their code working when they were having problems with it rather than looking at all of the project and checking that people were completing allocated tasks and submitting functional code. I feel that the project suffered as a result of this.

6.3.7 Sam Clements

The project started off with enthusiasm from most members, but most of the early work went into the data classes and structure, leaving the group feeling it had accomplished much more than it had. Once we reached coding week, a lot of time was spent learning the software we were working with – mainly tomcat and SQL.

We were also significantly delayed by the standards being incomplete or frequently changing – this meant that we had to spend a lot of time rewriting code to fit with newer versions of the standard, or writing code that would later turn out to be useless since no standard had been decided yet. During coding week, it changed very frequently.

We also commonly had problems with multiple people working on the same parts at once, causing problems with duplicated work or with git merge conflicts. This was made worse by the git software most of the team were using – Github for Windows – often doing the wrong type of merge and causing problems that team members did not know how to fix.

6.3.8 Daniel Stuessy

Working on this group project has been an interesting experience. It was of a large scale, and nothing like my previous ones. For this reason, it provided an opportunity for growth in the areas of competence and interpersonal skills.

Under the category of competence, time management possibly plays the biggest role. In the past, this has been of great weakness, slowly improving over the years. However, even more improvement is required, as has been noted in my group leader's critical feedback report. Specific points made were that I was highly forgetful, and as a result, falling behind on assigned tasks. This made me a weak and unreliable end of the group. I was, however, unaware of the level of forgetfulness. I did know, I was forgetting a few things, but not to the extent stated in the report. To improve this, using a calendar for personal note taking could help. This would require putting in the effort, making a habit of checking the calendar at regular times.

Under the category of interpersonal skills, I could say some improvement has been made. The first group project I had in this course went through several issues with interpersonal skills. People just did not get along, or communicate in an effective way with each other, causing delays and inefficiency. In this year's group project however, there were minor, maybe even no issues as to my best recollection. This was of great benefit to situations where arguments were easily caused. Of course, this area also spares some room for improvement, which I can obtain through the practice of social skills.

Generally, this has been an interesting learning experience, as the size and type of the project were quite different to previous ones. This gave me the chance to critically observe my performance and reflect on possible improvement. In regards to time management, the bane of my academic career, there is still much to learn. Interpersonal skills, on the other hand, seemed to improve during this group project, however still need refinement. Hopefully, with better time management skills I will be able to perform better in future group project.

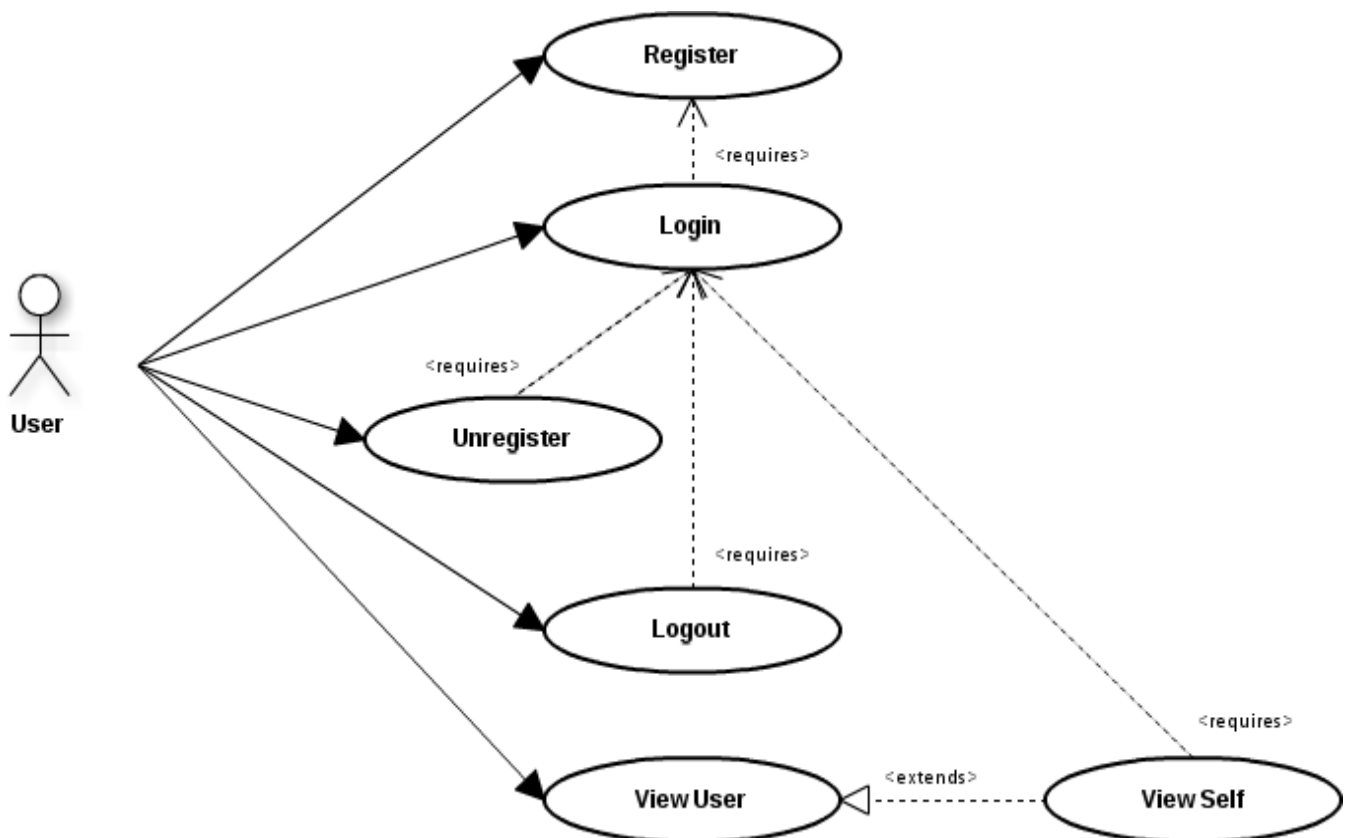
6.3.9 James Barnett

James has submitted his report via blackboard individually.

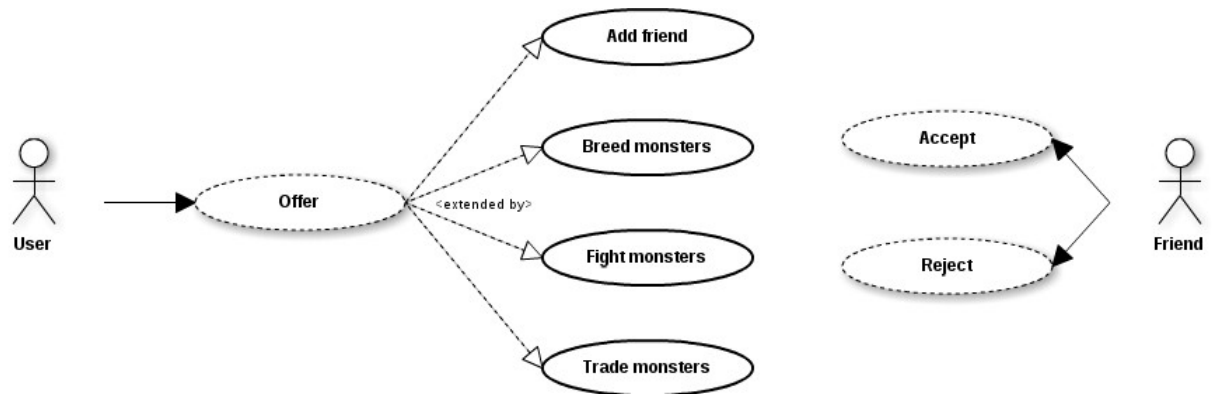
6.4 Project Plan

6.4.1 Diagrams

6.4.1.1 Account Management Use Case Diagram



6.4.1.2 Offers Use Case Diagram



6.4.2 Gantt Chart

See docs/final-docs/Gantt chart

6.4.3 Event Timeline

Main Event	Date	What took place and actions of project members
Arrange project team positions	08/10/12	Have a group meeting to decide positions in the team.
Introduction and details of the proposed system	08/10/12	Introduce the team members to the task ahead and give details of the system and discuss plans ahead.
First part of project documentation to be delivered	15/10/12	To allocate tasks and plan ahead so that work is collaborated on time for the first deliverable.
First deliverable	29/10/12	To deliver the work by this date.
Diagrams	29/10/12	To create a class and component diagram to give an understanding of the system structure.
Test identification	29/10/12	To identify tests that can be carried out on the finished system.
Standards meeting	29/10/12	To attend a standards meeting to discuss with the other group standards to be used.
Test Specification	29/10/12	To create the test specification for the final system so that it could be delivered.
Allocating work for coding week	29/10/12	To split up the workload so it was fairly distributed to members of the project for coding week including documentation.
Skeleton code	05/11/12	To prepare the skeleton code ready for the implementation of main methods.
Standards meeting	12/11/12	To attend a standards meeting to discuss with the other group standards to be used.
Second deliverable	12/11/12	To deliver all required documentation by this point.
Architectural and interface description	12/11/12	To put together the architectural description and interface description.
Start of coding	12/11/12	To start coding.
References	26/11/12	To put together references and appendices for the project.
Standards meeting	26/11/12	To attend a standards meeting to discuss with the other group standards to be used.
Third deliverable	03/12/12	To deliver all the required documentation up to this point which included the test specification.
Prototype demo submission	10/12/12	To research prototyping software to be used.
Integration and testing week	28/01/12	This week will be used to implement all the main methods in terms of coding and bring together the application in terms of the web interface and the back end code.

Fourth deliverable	03/02/12	To deliver the application to the end user.
Final deliverable	18/02/12	To deliver all of the work including the documentation.

6.4.4 Risk Assessment

This will look at various risks that can occur during the projects life. Considerations will be made and based on the risks, that can or may arise, actions will be put in place to either stop them occurring or to help recovery in such a situation.

Due to the nature of this project, there are not as many risks as there would be if this were a real life project. For example, there are no funding considerations to take into account, there are no staffing with top talent issues as the groups were chosen by the department and there are not as many software issues to take into consideration given that the university provides all the adequate software needed to successfully complete the project as required.

Risk	Action To Take
One of the major risks to the project is group members may become ill, fall behind or have a number of other personal issues to deal with that can have a direct impact on their productivity in terms of the project. If a project member falls behind there is a chance it could have an impact on their whole project productivity and therefore cause major problems in getting the project completed on time.	To combat this there are a number of things that can be done. Members who are ill shall be required to alert the project manager as soon as possible if they know it is likely that they are going to be off for some time so the work they had been allocated can be given to another member in the group and the whole project is not held back. If a member can't attend a meeting or meet a deadline they should also inform the project manager so that the project manager knows why. Time at the end of the project can also be set so that those that need to catch up on work can do so.
There is a risk that the project manager or quality assurance manager can become ill or cannot attend meetings for personal reasons.	To help stop this having severe consequences on the project work deputy roles have been given out to other members in the group. These then from attending meetings themselves should be able to pick up where those in the more senior positions left off if they are to become ill or not able to attend.
Group member turnover is another key risk. If a key programmer who has done most of the code just decides to drop out of university and there is no one else with understanding of the code that has been created it can either entirely derail the project or set the project back by quite a bit.	It's therefore essential that at least a couple of group members are working on the code so that if personnel leave for whatever reason there is someone there to carry on the implementation and update the other members of what is going on in terms of the implementation and integration of the application.
There is a risk that a project member may have issues with their own personal hardware and lose important documentation or code that they have themselves worked on.	To overcome this it is suggested that as soon as a piece of work is completed it is not only backed up on another device or online but submitted to the git repository so that the other members can sync with it and they will then themselves have a copy of the work done. More group members having a copy of completed work minimises the risk of losing it.

Communication breakdown between groups is possibly one of the biggest threats to this project in terms of risk. If standards are not set by groups and a decision is made with a strict understanding from all involved then there is a major problem whereby the applications created are unlikely to be able to communicate with one another. This would then mean that the project fails to deliver on one of the requirements set out at the start of the project in server communication.	To deal with this effectively it is important a number of standards meetings take place so each group is kept in the loop in terms of what standards are being used and why.
Another threat is there could be a loss of internet connection which holds back work production on the project. For example; if the on-campus network has connection problems during coding week this may cause synchronisation errors which may lead to lack of productivity. It may also stop work being able to be accessed if it is all stored online and there are no offline copies stored to be readily worked on.	To overcome this the group should keep an offline copy of all work on a regular basis by syncing with the Git repository so that if there is a connection failure they can carry on doing work on an offline copy and then sync at a time when connectivity is restored.
Lack of understanding from certain group members can also be a problem for any group project. If there is a lack of understanding and a member has been allocated work they may not do the required work and instead do something else believing they are doing the right thing.	To overcome this it is important that if group members have any doubts or misunderstandings about a piece of allocated work that they speak to the project manager to come to an understanding on the issue.
The scope is another issue. The project requirements and features listed may be beyond that of what the development team can deliver in the time available.	Assign the appropriate tasks to the group members with the skills to carry out those tasks efficiently and set work in accordance to project members strengths.
Another risk is developing the wrong user interface. As the front of the application it is important that the user interface is as required with the functionality set in the project requirements. Failure to do so could mean that the customer is unhappy with the end product.	To overcome this it is important that the goals are understood and that the user interface does exactly what it is meant to do without confusing the user. A good understanding of its purpose means it is likely to be implemented correctly and give the end user a more enjoyable experience using the software. If this were a real world project then surveys could also be handed out to the public to receive feedback on the user interface design.
Plagiarism is a massive risk to the group project as one person committing such an offence could have consequences on the project as a whole and cause problems. This also applies to a real world project whereby committing plagiarism could not only have productivity implications but financial and legal implications as well.	To overcome this it is essential that all code is checked and verified and that each user submits work clearly identifying the author or creator of the work so if plagiarism is an issue it is easy to identify the person responsible. Git also helps with this as submits are recorded so its easy to backtrack and see who submitted the work.

6.5 Design

6.5.1 INTRODUCTION

6.5.1.1 Purpose of this document

This purpose of this document is to describe the format of the design specifications for the application we are creating for our software engineering group project. This document describes the layout and structure of the design.

6.5.1.2 Scope

This document includes the following sections Decomposition description, dependency description, interface description, detailed design. This document should be read by all group project members.

6.5.1.3 Objectives

The objective of this document is to help aid the production of a design specification that is a complete and accurate translation of the client's requirements into a description of the design elements necessary for the implementation phase.

6.5.2 DOCUMENT CONTENT

6.5.2.1 Decomposition Description

Monster Mash Application

The Application to be created will create, manage and maintain an ongoing game called “Monster Mash”. The application will allow for multiple players to have “Monster farms” with various “Monsters” which allow for player to player interaction in the form of fighting, breeding & trading.

The application will run in a web based environment using JSON to send off data and receive data from the application server. The application will also communicate with a select number of other servers running a similar application, the servers should be able to operate with each other as if it was interacting with its self.

6.5.2.2 Monster Mash Significant Classes

The API Class

The API class is going to be our “hub” for communication between the client to server and server-server interactions. This will include setting up our requests, offers etc. Everything that happens between 2 players will need to go through this class sending and receiving data with the use of JSON.

The Player Class

The player class deals with getting our players friends, monsters, requests, offers etc. All things that our player is going to do inside of the application will be handled by the player class. Other things that occur with the players variables will also take place such as crediting them with money for a transaction such as a fight or debit the player because they accepted a breed offer.

The Monster Class

The monster class deals with our monster attributes, monsters have certain attributes such as strength, evasion, toughness etc. All of these attributes are determined by their age, so rather than storing this in our database we're going

to generate our monsters attributes based on their age each time they have an interaction. Interactions will include things such as fighting, breeding etc. These interactions require another player and as such we want the most up to date data when we finally undertake our interaction.

The Request Abstract Class

The request Abstract class will be extended by FriendRequest and FightRequest the class will contain methods for Accepting and rejecting requests and then the extended classes will build upon those for specific functionality.

The Offer Abstract Class

The Offer Abstract Class is to be extended by BreedOffer and TradeOffer the class will contain methods for making the offer, sending the correct player and monster references etc. The extended classes will build upon the methods for specific functionality.

6.5.2.3 Monster Mash Package Descriptions:

aber.dcs.cs211.group07.data

The Data package includes classes such as Player, Monster, MonsterReference, PlayerReference and the interface reference.

This package contains the classes for the data that we're going to be working with, such as monsters having attributes which are useful and need to be known in fight request/ offers. The methods for getting our players monsters, friends, requests and offers etc.

aber.dcs.cs211.group07.data.offers

The Data.Offers package includes classes such as BreedOffer, TradeOffer & the abstract class Offer.

This package will deal with the sending of offers and the various offers that we have available to us, this can be easily expanded from breed and trading offers.

aber.dcs.cs211.group07.data.requests

The Data.Requests package includes classes such as FriendRequest, FightRequest & the abstract class Request.

This package will deal with the receiving/ sending of requests from other players these requests can be easily extended from friends and fight requests.

aber.dcs.cs211.group07.db

The DB package includes classes such as DatabaseConnector, MonsterTableConnector, PlayerTableConnector.

This package deals with the database interaction for the application, we have to create players, monsters and store their data in the database all database interaction will be handled within this package.

aber.dcs.cs211.group07.tests

The Tests package includes classes for testing out data such as DataTests.

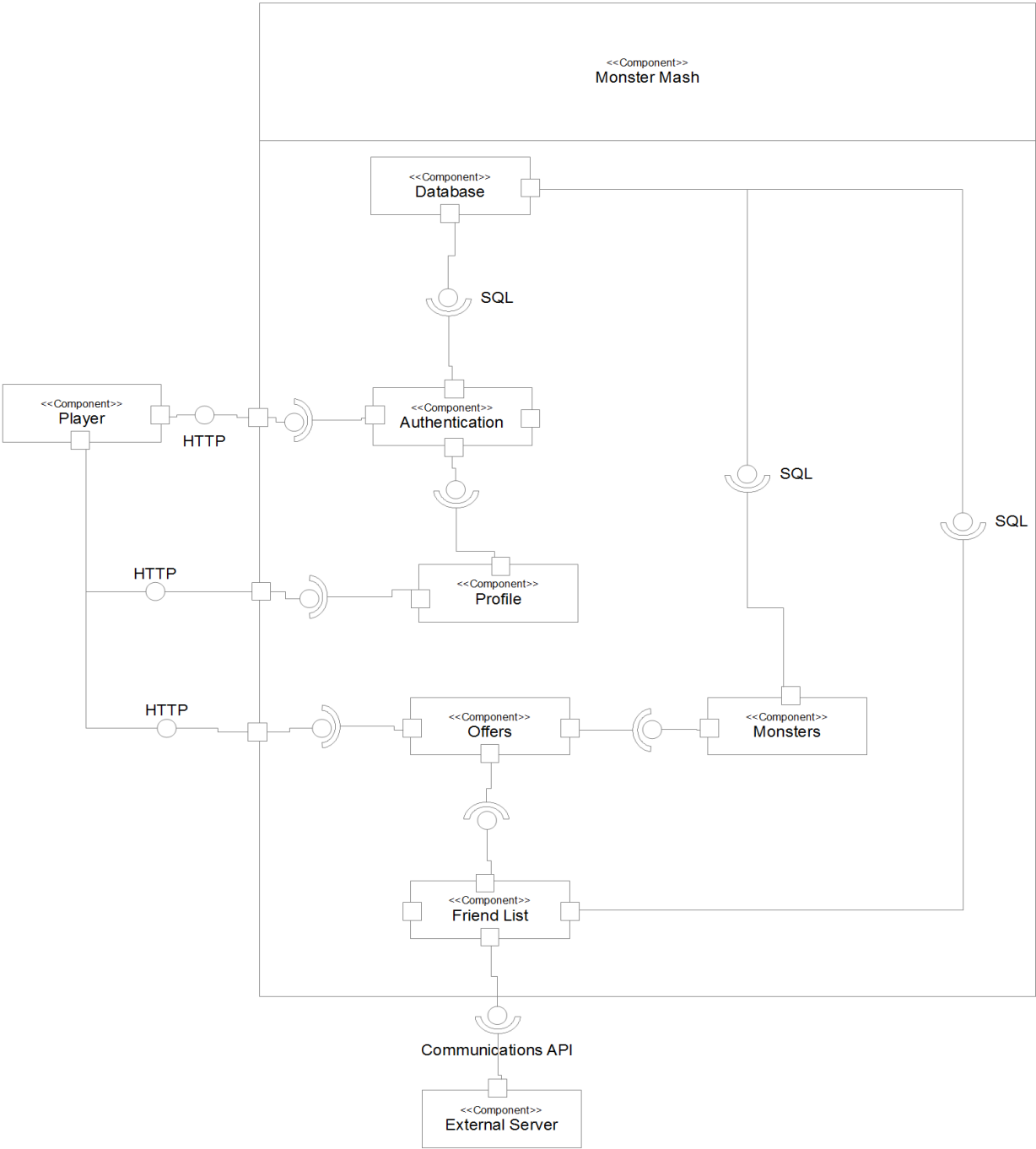
This package deals with the JUnit testing for our application, this will implement as many tests as practical from the test specification document to test our application, this will be especially useful throughout the project for monitoring progress of the application in a working state.

6.5.2.4 *Monster Mash Application Requirements Mapping*

The following table indicates the requirements that we meet and which classes meet those requirements.

Requirement	Class providing requirement
FR1	APIClass, DatabaseConnectorClass
FR2	PlayerClass, APIClass, DatabaseConnectorClass
FR3	PlayerClass, APIClass, DatabaseConnectorClass
FR4	PlayerClass, MonsterClass
FR5	APIClass, PlayerReferenceClass, MonsterReferenceClass, BreedOfferClass, FightRequestClass, FriendRequestClass.
FR6	APIClass, DatabaseConnectorClass, PlayerClass
FR7	PlayerClass, DatabaseConnectorClass, APIClass
FR8	PlayerClass, APIClass, DatabaseConnectorClass
FR9	RequestInterface, FriendRequestClass
FR10	RequestInterface, FightRequestClass
FR11	DatabaseConnectorClass, PlayerClass, APIClass

6.5.2.5 *Dependency Description*



6.5.2.6 Interface Description

The Player Class, Public, no extensions.

This class contains numerous variables as well as many methods and contains all the important data in regards to the players of the application.

Variables

+ id: int	The id variable is of type int. This will be an auto number that continuously counts up to avoid giving two or more players the same id.
+ serverID: int	The serverID variable will be of type int. This will be for the same reasons as above. This will with the correct validation stop servers of the same ID connecting with one another.
+ email: String	The email variable is of type String. This is because it needs to contain letters, numbers and symbols. To make sure that the correct email format is used validation will be implemented.
+ password: String	The password variable is of type String for the same reasons as above. Validation will also be run on this to make sure the password is secure.
+ money: int	The money variable will be of type int as it will be a numerical value.

Methods

+ getFriends(): List <Player>	This method will get friends by showing a list of other players. This will allow the user to view a list of other players in their friends list.
+ getOffers(): List <Offer>	This method will allow users to view a list of the current offers.
+ getRequest(): List <Requests>	This method will allow users to view a list of the current requests.
+ getMonsters(): List <Monster>	This method will allow the user to get monsters from a list of monsters and display them.
+ debit (amount:int)	This method will allow an amount to be debited from the account of the player. As the money variable is an int this variable has to be of the same type.
+ credit (amount: int)	This method will allow an amount to be credited to the account of the player. As the money variable is an int this variable has to be of the same type.

The Monster Class, Public, No extensions.

This class contains numerous variables as well as many methods and contains all the important data in regards to the monsters of the application.

Variables

+ owner: Player	The owner variable allows the owner of a certain monster to be set. The owner will be of type Player.
+ name: String	The name variable allows the name of the monster to be set.
+ birth: final Date	The birth variable allows the date of the monsters birth to be set.
+ gender: final bool	The gender variable will be of type boolean. This will mean that the monster can be either male or female only.
+ health_lost: double = 0.0 AGE_RATE: Time	Health lost occurs from fighting as our monster can loose some or all of his health in a fight.

Methods

+ getHealth(): double	This method will allow the health to be returned for the monster.
+ getStrength(): double	This method will allow the strength to be returned for the monster.
+ getToughness(): double	This method will allow the monsters toughness to be returned
+ getEvasion(): double	This method will allow the evasion skill of the monster to be returned.
+ getAge(): Time	This method will allow the age of the monster to be returned based on time in relation to the birth date.

The Offer Class, Public, Abstract.

This class contains numerous variables as well as many methods and contains all the important data in regards to the offers in the application.

Variables

+ player: PlayerReference	This variable is the reference back to our player, so that when we send an offer to our friends we can use this reference to interact with the correct player's stuff.
+ monster: MonsterReference	This variable is the reference back to our monster, so that when we send an offer to our friends we can use this reference to interact with the correct monster from our player.
+ cost: int	This variable gets the cost based on the offer made.

Methods

+ getFromPlayer(): Player	This method will allow a player to get a monster from another player in the application.
+ accept(p:Player)	This method will allow a player to accept an offer from another player in the application.

The PlayerReference Class, Public, implements reference.

This class implements reference because reference is an interface for all references which we would require in an interaction. If we would later need to implement more references back to more information such as a “Team/ Clan” in a later development of the application for extra competitive game play.

This class contains the get for the player as we need to know which player we're dealing with in interactions.

Methods

+ get(): Player	This method will allow a player to be gotten from elsewhere in the application by the server. (i.e. this class is called in both the Offer and Request classes numerous times.
-----------------	---

The MonsterReference Class, Public, Implements reference.

This class implements reference because reference is an interface for all references which we would require in an interaction. If we would later need to implement more references back to more information such as a “Team/ Clan” in a later development of the application for extra competitive game play.

This class contains the get for the monster as we need to know which monster we're dealing with in interactions between players and monsters such as fighting.

Methods

+ get(): Monster	This method will allow a monster to be gotten from elsewhere in the application by the server. (i.e. this class is called in both the Offer and Request classes.
------------------	---

The Request Class, Public, Abstract Class.

This class contains numerous variables as well as many methods and contains all the important data in regards to the requests that can be made in the application.

Variables

+ fromPlayer: PlayerReference	This variable holds the information of the “fromPlayer” which will hold the reference for which the request came from.
+ toPlayer: PlayerReference	This variable holds the information of the “toPlayer” which is the player receiving the request from the “fromPlayer”.

Methods

+ accept()	This method will allow a player to accept a request.
+ reject()	This method will allow a player to reject a request.

The BreedOffer Class, Public, Extends Offer.

This class extends offer because it's a type of offer, we could later add multiple offers and as such have an abstract class called offer which includes the outline of every offer.

This class is for the specific offer of a Breeding offer, all breed offers are made to all friends in the friends list and

contain the Player and the Monster being offered for breeding.

Methods

+ accept(p:Player, m:Monster, c:Cost)	This method will allow a player to accept an offer from another player in the application, this includes a player and a monster as they're both arguments. Cost is also going to be taken from the accepting player.
---------------------------------------	--

The TradeOffer Class, Public, Extends Offer.

This class extends offer because it's a type of offer, we could later add multiple offers and as such have an abstract class called offer which includes the outline of every offer.

This class is for the specific offer of a Trade offer, all trade offers are made to all friends in the friends list and contain the Player and the Monster being offered for trade with the monster being essentially sold for money.

Methods

+ accept(p:Player, m:Monster, c:Cost)	This method will allow a player to accept an offer from another player in the application, this includes a player and a monster as they're both arguments. Cost is also going to be taken from the accepting player.
---------------------------------------	--

The FightRequest Class, Public, Extends Request.

This class extends Request because a fight is a type of request between two players that must be accepted the same as a friend request.

This class contains numerous variables as well as many methods and contains all the important data in regards to the FightRequests that can be made in the application.

Variables

+ fromPlayer: PlayerReference	This variable holds the information of the “fromPlayer” which will hold the reference for which the request came from.
+ toPlayer: PlayerReference	This variable holds the information of the “toPlayer” which is the player receiving the request from the “fromPlayer”.

Methods

+ accept()	This method will allow a player to accept a fight request and then the fight will take place on the server of the player accepting the fight.
+ reject()	This method will allow a player to reject a request.

The FriendRequest Class, Public, Extends Request.

This class extends Request because a fight is a type of request between two players that must be accepted the same as a fight request.

This class contains numerous variables as well as many methods and contains all the important data in regards to the FriendRequests that can be made in the application.

Variables

+ fromPlayer: PlayerReference	This variable holds the information of the “fromPlayer” which will hold the reference for which the request came from.
+ toPlayer: PlayerReference	This variable holds the information of the “toPlayer” which is the player receiving the request from the “fromPlayer”.

Methods

+ accept()	This method will allow a player to accept a fight request and then the fight will take place on the server of the player accepting the fight.
+ reject()	This method will allow a player to reject a request.

The MonsterTableConnector Class, Public, No Extensions.

This class deals with the connecting and editing of database information in the Monster table, this table contains the information of the monsters and who their owners are.

Variables

+Statement: Statement	Statement used to get result sets from the database.
+Results: ResultSet	Results from our database connection and query so that we can process information.
+MonsterTable: String	Table query variable, this can be changed for more specific functionality/ query results.
+Host: String	Variable for the host database address
+UserName: String	The database username variable.
+Password: String	The database password variable.

Methods

+createMonster(Monster mon)	The creation of a monster into the database taking a monster as the argument which will include all the information that a monster needs to be stored in the database
+deleteMonster(Monster mon)	The deletion of a monster from the database, taking a monster as the argument so that we can remove the monster after death or owner account de-activation.
+getMonster(int monID)	Getting the monster out of our database so that we can pass it to the monster class and generate our attributes before we send it off to fight etc.
+editHealthLost(Monster mon, double amount)	When we've taken damage from a fight we need to update our health lost to reflect that fact.
+getSex(Monster mon)	When putting up our monster for trading, breeding and generally purposes where gender is important we've got to

	find out the sex of our monster before hand.
--	--

The PlayerTableConnector Class, Public, No Extensions.

This class deals with the connecting and editing of database information in the Monster table, this table contains the information of the monsters and who their owners are.

Variables

+Statement: Statement	Statement used to get result sets from the database.
+Results: ResultSet	Results from our database connection and query so that we can process information.
+MonsterTable: String	Table query variable, this can be changed for more specific functionality/ query results.
+Host: String	Variable for the host database address
+UserName: String	The database username variable.
+Password: String	The database password variable.

Methods

+createPlayer(Player newPlayer)	The creation of a player into the database taking a player as the argument which will include all the information that a player needs to be stored in the database
+deletePlayer(Player newPlayer)	The deletion of a player from the database, taking a player as the argument so that we can remove the player after deletion of the account.
+getPlayer(int playerID)	Getting the player out of our database so that we can pass it to the player class and get our monsters etc for display back to the user.
+editMoney(Player player, int amount)	When we use the credit or debit methods in player we've got to then update the SQL with the appropriate data.
+login(String Username, String Password)	Server side login, we're logging out player into the application.

6.6 User interaction design

6.6.1 INTRODUCTION

6.6.1.1 Purpose of this document

This document describes the user interaction design for the .

6.6.1.2 Scope

This document covers the possible interactions a user may have with the end product.

6.6.1.3 Objectives

The objectives of this document are to:

- Describe all possible user interactions with the end product, without any reference to technical aspects of the product.-----

6.6.2 DOCUMENT CONTENT

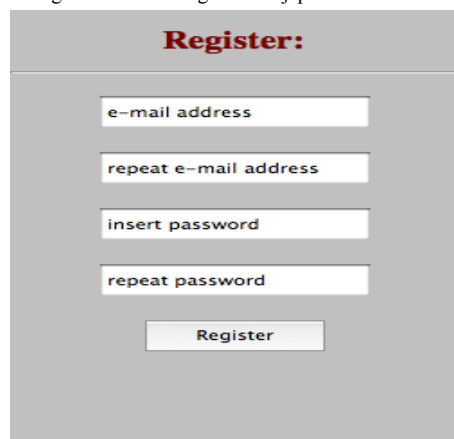
6.6.2.1 Account management

When a user starts playing the 'Monster Mash' game, they will first have to register and then login in order to start playing. The homepage for the game will display options for the user to provide an email and password to login, or a button to register.

6.6.2.2 Register/Unregister

The first step you need to make in order to play the game is to create an account. When you click on the "register" button, you will be taken to another page where you will be able to fill in your information, including an email and password. When this has been filled in, the user will be able to click a button at the bottom of the page to continue. A page will then be displayed telling the user if the action was successful or not.

Pic 1. Registrtrtion box – registration.jsp

A screenshot of a web registration form. The form has a grey background with a white border. At the top, the word "Register:" is written in a bold, red, serif font. Below this, there are four white input fields with black text labels: "e-mail address", "repeat e-mail address", "insert password", and "repeat password". At the bottom of the form, there is a white button with the word "Register" in black text.

Users can remove their account with the "unregister" button placed on MyProfile site. Again, a page will then be displayed telling the user if the action was successful or not.

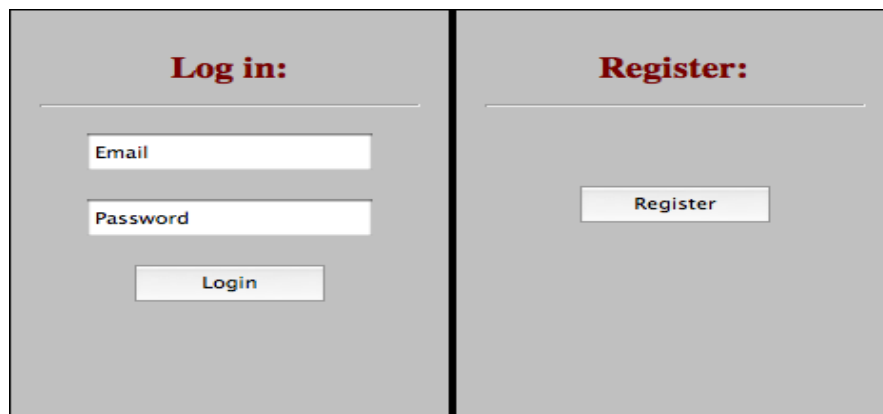
Pic 2. Piece of MyProfile.jsp containing unregister button.



6.6.2.3 Login/Logoff

A login form will be displayed on the homepage. The user will need to enter the correct account details, and will then be taken to their account page. If the account details are incorrect, they will be returned to the home page and shown a message explaining this. Underneath the form will be an option to register.

Pic. 3 Login box – index.jsp

A screenshot of a web form divided into two panels by a vertical black line. The left panel is titled "Log in:" in a bold, dark red serif font. It contains two input fields: "Email" and "Password", both with light gray backgrounds and black borders. Below these fields is a "Login" button with a light gray background and black border. The right panel is titled "Register:" in a bold, dark red serif font. It contains a single "Register" button with a light gray background and black border.

Once the user has logged into their account, they will be able to see a logoff button. Pressing it will immediately take them out of the game and back to the home page of the game.

Pic 4. Menu bar tabs contains Log Out button.

[MyProfile](#)

[Notification](#)

[Monsters](#)

[Highscores](#)

[Log out](#)

6.6.2.4 Offers

There are several types of user interaction that can be described in terms of “offers”, where one user offers a request to another user, who then has the chance to either accept or reject the offer. There are four interactions that work this way:

6.6.2.5 Friend Requests

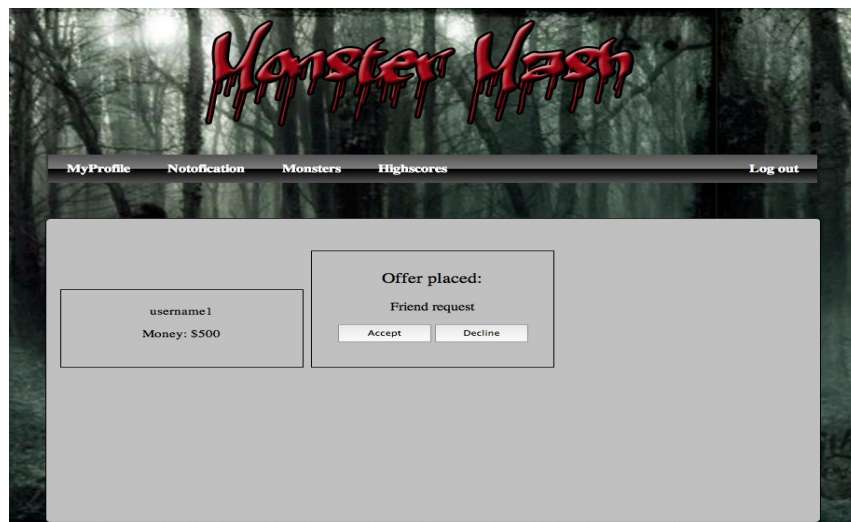
To send friend request to another player we need to insert friends name (email address) in a text box, placed below friends list and press “Add” button. After that a friend request will occur in a notifications panel.

Pic 4. Friends table containing a textfield.

FRIENDS:	\$
user1@email.com	\$50
user2@email.com	\$150
user3@email.com	\$250
<input type="text" value="user@email.com"/>	<input type="button" value="Add"/>

When click on friend request in notification panel you will be redirected to new window, where you will be able to accept or decline friends request.

Pic. 5 Friend request



If offer is accepted a new friend occurs in your friend table. From now on you can send offers to this player.

6.6.2.6 Fight request

To sent a fight request to another player you have to press “fight” button placed under your friend name in friends table.

Pic. 6 Fight button under player record in a table.

The screenshot shows a web interface titled 'FRIENDS:'. Below the title is a large empty rectangular area. To the right of this area, the text '\$ 900' is displayed. Below the large area is a white rectangular input field. At the bottom left of the interface is a button labeled 'Add'. At the bottom right is a button labeled 'Fight'.

You will be redirected to another page, where you have to choose one monster from your and friend's farm and specify a prize.

Pic. 7 Fight request page.

The screenshot shows a web interface titled 'Offer placed:'. Below the title is the text 'Fight request'. Below this is a 'Prize:' label followed by a text input field containing the value '100'. Below the input field is a button labeled 'Sent fight request'. To the left of the central form is a box containing the text 'me@email.com', 'Money: \$500', and two checkboxes: '✓Monster 1' and '☐Monster 2'. To the right of the central form is a box containing the text 'your_friend@email.com', 'Money: \$500', and two checkboxes: '☐Monster 1' and '✓Monster 2'.

Your opponent has to accept your offer, otherwise any action won't be taken. If accepted monsters proceed to fight. After the fight a winner receives money reward. A report from a fight occurs on a notifications page.

As a result of any fight your monster can be injured or die. Injury may reduce the monsters attributes, and death will remove it from the owners list of owned monsters.

6.6.2.7 Breed

To send a breed request to your friends you need to go to your monster farm. Under every Monster record is placed a “Offer to breed” button.

The screenshot shows a user interface for a monster farm. At the top left, it says "MONEY: \$\$\$". Below this is a grid of six monster records, labeled Monster 1 through Monster 6. Each record has two buttons: "Stats" and "Offer to breed". To the right of the grid is a "FRIENDS:" list with a "\$" column. The list contains six entries: Player 1 (\$ 900), Player 2 (\$ 800), Player 3 (\$ 2503), Player 4 (\$ 321), Player 5 (\$ 621), and Player 6 (\$ 123). At the bottom of the friends list is an "Add" button.

When you click it, you will be redirected to another page, where monster with stats are displayed and you will be asked to enter a price. Then you have to press a “Send breed offer” and it will be sent to every one from your friends list.

The screenshot shows a page titled "Offer placed:". On the left, there is a box containing the user's information: "me@email.com", "Money: \$500", "Monster 1", and "Stats:". On the right, there is a form for the "Breed request". It has a "Prize:" label followed by a text input field. Below the input field is a button labeled "Sent breed request".

If anyone will accept your offer, you will get a reward (price) and other person gets a new monster. If offer is rejected nothing happens.

6.6.2.8 Trade

This game allows the user to trade our monsters to other players. We might decide we want to sell one of our creatures in order to earn some extra money etc. To send a sell request we need to go to our monster farm and press the ”sell” button under the chosen monster.

Monster 1 <input type="button" value="Stats"/> <input type="button" value="Offer to breed"/> <input type="button" value="Sell"/>	Monster 2 <input type="button" value="Stats"/> <input type="button" value="Offer to breed"/> <input type="button" value="Sell"/>	Monster 3 <input type="button" value="Stats"/> <input type="button" value="Offer to breed"/> <input type="button" value="Sell"/>	FRIENDS: \$ <hr/> Player 1 \$ 900 Player 2 \$ 800 Player 3 \$ 2503 Player 4 \$ 321 Player 5 \$ 621 Player 6 \$ 123 <hr/> <input type="text"/> <input type="button" value="Add"/>
Monster 4 <input type="button" value="Stats"/> <input type="button" value="Offer to breed"/> <input type="button" value="Sell"/>	Monster 5 <input type="button" value="Stats"/> <input type="button" value="Offer to breed"/> <input type="button" value="Sell"/>	Monster 6 <input type="button" value="Stats"/> <input type="button" value="Offer to breed"/> <input type="button" value="Sell"/>	

System redirects us to another page, where we decide how much we expect to earn on this transaction and press “sent sell request”. This will sent sell request to all players from our friends list.

me@email.com Money: \$500 Monster 1 Stats:	Offer placed: Sell request Price: <input type="text"/> <input type="button" value="Sent sell request"/>
---	---

If offer is accepted, we get money and the other side receives a monster. If offer is declined nothing happens. In both cases an appropriate message occurs on our notification page.

6.6.2.9 Notifications

Once people have made offers to you, they will appear on the notifications page. Clicking on a “View offer” button will take the user to the 'Accept offer'. page.

We can also check reports from our last fights (won and lost) by clicking on the “Show report” button.

6.6.3 Accept an offer

A user can select an offer, which will then show this screen, displaying details of the offer to be accepted. The details will be based on whether the offer is a friend request, or a monster trade/breed/fight. Either the friends or the monsters involved will be shown on the left and right, as a small card containing their stats. In the centre will be buttons to accept and reject the offer.

MyProfile	Notofication	Monsters	Highscores	Log out		
<table border="1"> <tr> <td> me@email.com Money: \$500 </td> <td> Offer placed: Friend request <input type="button" value="Accept"/> <input type="button" value="Reject"/> </td> </tr> </table>					me@email.com Money: \$500	Offer placed: Friend request <input type="button" value="Accept"/> <input type="button" value="Reject"/>
me@email.com Money: \$500	Offer placed: Friend request <input type="button" value="Accept"/> <input type="button" value="Reject"/>					

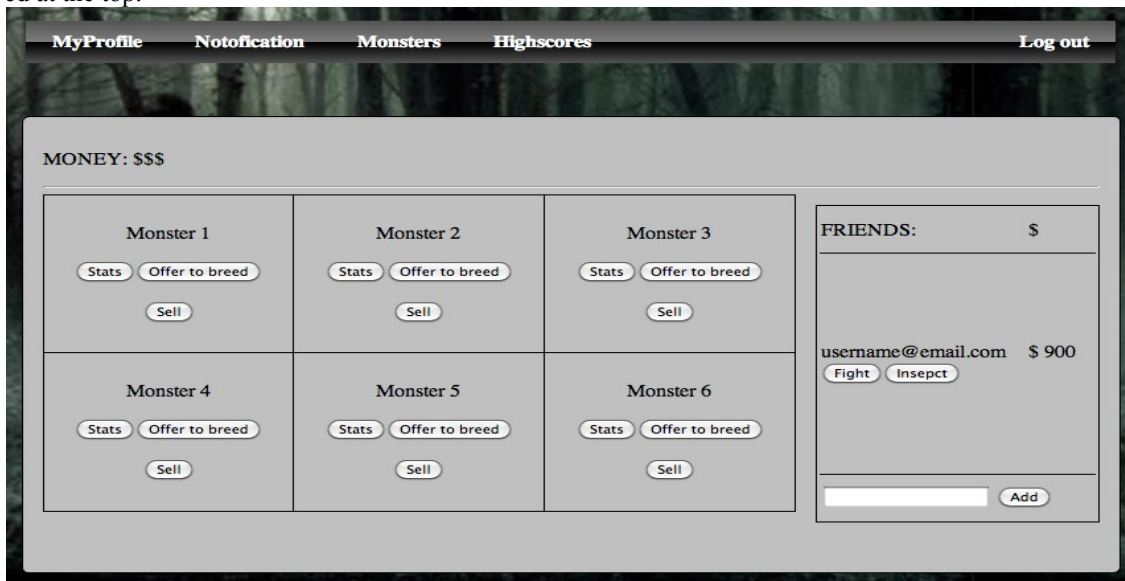
When the user continues, they will be taken back to the previous page and shown a message telling them that the offer has been accepted or rejected.

6.7 Other Views

6.7.1 Home Screen

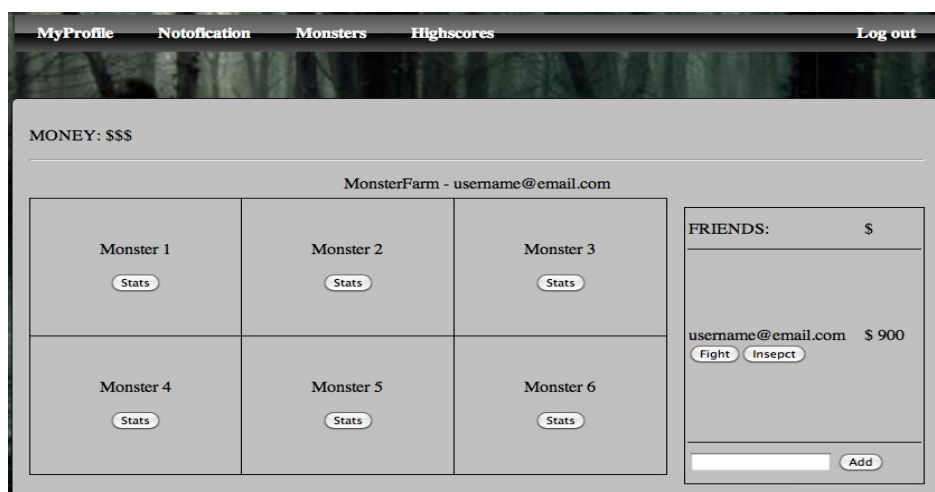
After you successfully login to a system you will be directed to a My Farm page.

From the main home screen you can see a panel of your monsters, a list of your friends on the far right and an option to add friends. You can also logout using the logout button at the top right corner. The money you currently have is displayed at the top.



6.7.2 View Friends Monsters

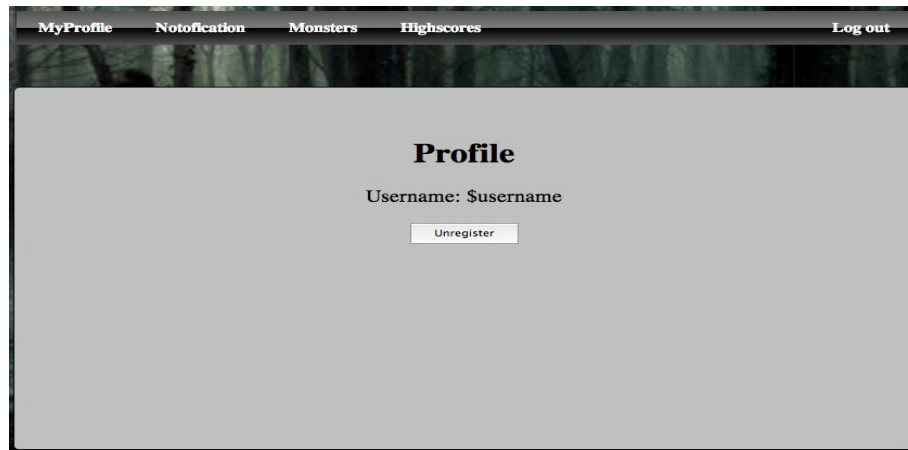
You can inspect you friend's monster farm and check what monster he/she has simply by clicking on the “Inspect” button under your friend's name in a friend's list table.



Below that you will see the main panel which will display the monsters that your friend has and the stats for each monster.

6.7.3 Profile View

On this page you can see your username and a “Unregister” button. You can use it to remove your account. You can also logout using the logout button at the top right corner.



6.8 Test specification

6.8.1 INTRODUCTION

6.8.1.1 Purpose of this document

The purpose of this document is to provide a customer with a list of test to exercise a web game engine.

6.8.1.2 Scope

This document specifies standard tests customers should perform to check the quality of the Monster Mash game. All tests contain a description of which type of action should be taken, the expected result and pass criteria.

6.8.1.3 Objectives

To describe the plan for testing and pass criteria.

6.8.2 DOCUMENT CONTENT

Register/Unregister

Test Ref	Req. being tested	Text Content	Input	Output	Pass criteria
T-SN-01	FR6	Check if “register” button works.	Press register button on a main page.	A new page with fields to fill in e.g. username and password occurs.	Redirection to a new page.

T-SN-02	FR6	Check if system will register an account without filling in any data on the register page.	Enter no data on the registration page and then press “continue” button.	Error message warns of missing data.	Error message displayed on a page. Account not added to a server.
T-SN-03	FR6	Check if system will register an account with some data missing e.g. password.	Fill in all fields beside password field and press “continue” button.	Error message warns of missing data.	Error message displayed on a page. Account not added to a server.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-04	FR6	Check if system will register an account when confirm password field differs to password.	Fill in all data., but misspell “confirm password” field, so that it differs to password.	Error message warns of incorrect data.	Error message displayed on a page. Account not added to a server.
T-SN-05	FR6	Check if system will register an account when login already in use.	Fill in login that is already in use on a registration page and click “continue” button.	Error message warns of existing account.	Error message displayed. Account not added to a server.
T-SN-06	FR1	Check if system will register an account when all the data is filled in correctly.	Fill in all the data properly and press “continue” button.	A new window occurs, telling that account has been added successfully.	New window with positive message occurs. Account added to a server.
T-SN-07	FR6	Check if “unregister” works.	Go to MyProfile and press “unregister” button.	A new page occurs with some fields to fill in e.g. username, password.	Redirection to a new page.

Login/Log off

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-11	FR1	Check if system lets user login without username and password	Fill in nothing and press "login" button.	Error message warns of incorrect login or password.	Error message on a page. Login fails.
T-SN-12	FR1	Check if system lets user log in, if one field is empty.	Fill in only one field. Press "login" button.	Error message warns of incorrect password or login.	Error message on a page. Login fails.
T-SN-13	FR1	Check if system lets user login when password is incorrect.	Fill in both fields, but misspell password. Press "login" button.	Error message warns of incorrect login or password.	Error message on a page. Login fails.
T-SN-14	FR6	Check if system lets user log off when "log off" button is pressed.	When logged in press "log off" button in the top-left corner.	The home page occurs.	Home page occurs. User logged off.

Offers:

Assumptions:

To perform some of the tests below we have to assume, that we have created at least two accounts. It will help us to inspect interactions between accounts and check, if the fight engine works properly. To make it easier we will create accounts test1 and test2.

Add/remove friend

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-15	FR9	Check if system sends an "add friend" request. This requires to check if target friend's details are correct.	Go to test1 and in the "add friend" field type in test2(friend you want to add) and press "add friend" button.	Test1: Message occurs saying that message was sent. Test 2: Add friend request occurs in offers window.	Request sent from test1. Request received on test 2.
T-SN-16	FR9	Check if system adds a friend when request declined.	Test2: when request occurs in notification window press "decline".	Test1: Message occurs saying that your request was declined.	Test1: Message occurs. Both: No friend added to a friends list.
T-SN-17	FR9	Check if system adds a	Test2: when request occurs	Both: Message occurs saying	Both: Message occurs. Friend added to a list.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
		friend when request accepted.	in notification window press "accept".	that a friend was added to a friends list.	
T-SN-18	FR9	Check if system adds a friend when "add button" is pressed, but request isn't accepted or rejected.	Test1: send a friend request. Test2: take no action.	Test1: Message occurs saying that message was sent. Test 2: Add friend request occurs in offers window.	Test1: Message occurs. Test2: Request received. Both: No friend added to a friends list.
T-SN-19	FR6	Check if system deletes a friend from your friend list, when "remove a friend" button pressed.	Test1: Go to your friends list and remove test2.	Both: Message occurs saying that a friend was removed from a list.	Both: Message occurs. Friend removed from a list.

Fighting.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-20	FR6	Check if "fight" button works.	Test1: Go to your friends list and choose test2, pick one of monsters and press "fight". Test2: Take no action.	Test1: Message saying, that request has been sent occurs. Test2: Request occurs in notification window.	Test1: Message occurs. Test2: Request occurs. Both: Any other action isn't taken.
T-SN-21	FR10	Check if system lets to fight without acceptance.	Test1: Pick one of friend's monsters and click "fight". Test2: Take no action.	Test1: Message saying, that request has been sent occurs. Test2: Request occurs in notification window.	Message displayed. Request sent. No other action is taken.
T-SN-22	FR10	Check if system lets to fight when request rejected.	Test1: Pick one of friend's monsters, press "fight" Test2: Reject an offer.	Test1: Message saying, that your request has been rejected occurs.	Fight hasn't been taken.
T-SN-23	FR10	Check if system lets	Test1: Pick one of friend's	Both: Message with a result is	Message occurs. Fight has been taken.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
		users fight when request accepted.	monsters, press "fight" Test2: Accept an offer.	displayed saying who has won and lost and what is a reward.	
T-SN-24	FR10	Check if winner receives a reward.	After a fight: Test1: If won inspect your balance. Else if test2 won: inspect your balance.	After a fight: Both: Message with a result of fight occurs.	Winner's balance changed. Loser's balance didn't change.
T-SN-25	FR10	Check if after a fight monster's health is decreased.	Test1: Proceed to fight. After that, inspect monster.	New window with monster's statistics occurs.	Monster's health is decreased.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-26	FR10	Check if after monster is injured an injury status occurs in its profile.	Both: Inspect your monster after every fight and check, if it was injured afterwards.	In the result window a message saying that your your monster was injured during a fight.	When monster injured – injury notice appears in its statistics.
T-SN-27	FR10	Check if after monster is dead system removes it from a monsters list.	Proceed to fight till you monster dies.	Message saying your monster is dead.	Message occurs. Monster removed from monsters list.
T-SN-28	FR10	Check if system generates new monster, after your last monster died.	Proceed to fight till your last monster is dead.	Message occurs that your last monster is dead and new one was generated.	Message occurs. Monster removed from a list. New monster generated and added to a list.

Breed:

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-29	FR3	Check if "breed" button works.	Test1: Press "breed" button. Pick your monster and insert a price. Test2: Take no action.	Test1: Message occurs saying that your breed offer has been sent. Test2: Offer occurs in notification	Test1: Message occurs. Test2: Offer occurs in notification window.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
				window.	
T-SN-30	FR3	Check if system will allow to breed without an offer being accepted by any other player.	Test1: Press “breed” button. Pick your monster and insert a price. Test2: Take no action.	Test1: Message occurs saying that your breed offer has been sent. Test2: Offer occurs in notification window.	Test1: Message occurs. Test2: Offer occurs in notification window. Both: Monster isn't generated.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-31	FR3	Check if system will allow to breed, when offer accepted.	Test1: Proceed to breed. Test2: Accept an offer.	Test1: New monster occurs in a list. Test2: Message saying that breeding was successful.	Test1: New monster, balance decreased. Test2: Message occurs, balance increased.
T-SN-32	FR3	Check if you can accept the same offer more than once.	Test1: After offer has already been accepted. Take no action. Test2: Try to accept same offer again.		After first acceptance offer disappears from a notification window. No other action is taken.
T-SN-33	FR3	Check if new monster has a value and statistics.	Test1: Proceed to breed. After accepted inspect new monster. Test2: Accept an offer.	A window with monster's statistics appears.	New monster has value and stats.

Trade:

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-39	FR8	Check if “sell monster” button works.	Test1: Go to your monster list, pick a monster and press “sell monster” button. Test2: Take no	Test1: Message saying that offer has been sent to all friends occurs. Test2: Offer occurs in a notification	Test1: Message occurs. Test2: Offer occurs in notification window.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
			action.	window.	
T-SN-40	FR8	Check if system allows to sell monster without any other player's acceptance.	Test1: Proceed to sell a monster. Test2: Take no action.	Test1: Message saying that your request has been sent. Test2: Offer occurs in notification window.	Test1: Message occurs. Test2: Offer occurs in notification window. Both: Nothing else happens.
T-SN-41	FR8	Check if system allows to sell a monster when offer accepted.	Test1: Proceed to sell monster. Test2: Accept an offer.	Test1: Message saying that your offer has been accepted occurs.	Test1: Message occurs, monster removed from a list balance increased. Test2: Monster appears on a monster list, balance decreased.
T-SN-42	FR8	Check if offer disappears when it has already been accepted.	Test1: Proceed to sell. Test2: Accept an offer. Inspect a notification window in order to accept same offer again.	Test1: No offer in notification window. Test2: No offer in notification window.	After offer accepted once it disappears from a notification window.

For all offers.

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-43	FR8	Check if offer has deactivated after monster died.	Test1: Fight until monster dies. Test2: Fight test1's monster until it is dead.	Both: Offer referring to the dead monster are not in the notifications window.	Both: After monster is dead all referring offers are deactivated.

Views: (user interaction design document = UIDD)

Test ref	Req. being tested	Text content	Input	Output	Pass criteria
T-SN-44	FR7	Check the login page loads when the homepage is requested via URL.	Test1: Enter homepage URL in browser.	Test1: The login view as shown in the UIDD loads in the browser.	The login page loads correctly with all the login and sign up features
T-SN-45	FR8	Check the	Test1: Login to	Test1: The user	The home screen displays

		monster list appears after login.	game.	is presented with the home screen view as shown in the UIDD.	correctly with all of its features as shown in the UIDD.
T-SN-46	FR8	Check if a friend's monster list appears by request.	Test1: Click on a friend in the friends list at the left.	Test1: A list of all the monsters the friend you clicked on owns, as seen in the UIDD.	The friend's monster view as shown in the UIDD is displayed correctly with all of its features.
T-SN-47	FR8	Check if a user's profile displays by request.	Test1: Click on user in friend list.	Test1: The user's profile information should appear.	The profile view as shown in the UIDD is displayed correctly with all of its features.
T-SN-48	FR8	Check if the notifications page displays correctly and by request.	Test1: Click on the notifications page button/link.	Test1: The user's notifications about offers, battles, and other relevant in-game information the user has to know, are displayed.	The notifications view as shown in the UIDD is displayed correctly with all of its features.
T-SN-49	FR8	Check if the 'accept offer' view displays correctly and by request.	Test1: Click on an offer in the notifications page.	Test1: The necessary features of the 'Accept Offer' view from the UIDD are displayed, allowing the user to accept or decline an offer.	The 'Accept Offer' view as shown in the UIDD is displayed correctly with all of its features.
T-SN-50	FR8	Check if the 'choose monster' view displays correctly and by request.	Test1: Click on a battle request in the notifications page.	Test1: The 'Choose a Monster' view is displayed, allowing the user to choose which monster to fight the opponent's.	The 'Choose a Monster' view as shown in the UIDD is displayed correctly with all of its features.

7 REFERENCES

This section is an automatic bibliography.

8 DOCUMENT CHANGE HISTORY

Version	CCF No.	Date	Sections changed	Changed by
1	N/A	18/02/13	Document created	John Bolton