Software Development Life cycle
End of project report

Author:	Tom Reed, Matt Whitmore, Dave Clark, Silhab Csoma, Mike Steel, Chris 'Tux' Lloyd, Aleksandra Badyda, Samuel Jackson, Chris Marriott
Config. Ref.:	SE.17.DS.01
Date:	2013-02-07
Version:	1
Status:	draft

Department of Computer Science,
Aberystwyth University,
Aberystwyth,
Ceredigion, SY23 3DB,
U.K.

CONTENTS

1 Design Plan

1.1 Purpose of this Section

This should provide everything designers, programmers and testers need to know to use the facilities provided by a module. It should include an outline for all parts of the system. The information provide should be enough to get a basic understanding of the system.

1.2 Scope

An outline for each class and its containing methods. Explanation of how the server works and significant algorithms. Outline of how the system is navigated and operated through sequence diagrams.

1.3 Objectives

The objective of this document is to give an overview of the system that is being produced.
The areas covered by this plan are:

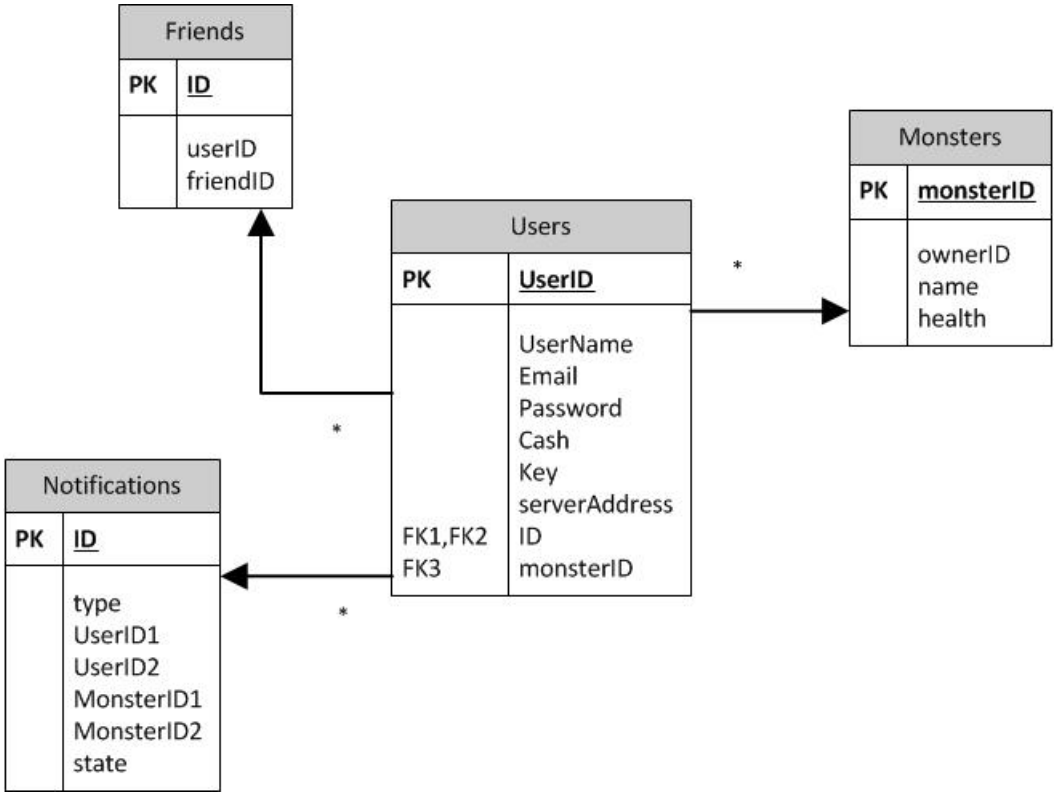
- Java Classes explanations
- Sequence diagrams
- Relational Database diagram
- JSON Table
- Algorithms

2 General Functionality

2.1 Description

An HTTP servlet is used to access the data contained within the backend of the server as if it was almost a web page. We will pass parameters between it and get a response. We will be working on the principle that one main servlet will perform different actions, passing through an actions variable and any data required to be processed. There are two methods of accessing this; using get and post requests. The get request will pass the parameters through the URL, the post request through a hidden layer, based on the users input. JavaScript will collate the necessary data, and attach the appropriate action command before sending to the server.

2.2 Relational Database diagram



2.3 Relational Database diagram explanation

This is a relational database diagram for our database, we have used 4 tables to store information about the User, the Users table to store information directly related e.g. user name etc. We then have the Monsters table in which will have information about the users monsters, the friends table to store the users friends and the notifications table to store information about any notification the user might have e.g. a user who wants to battle their monster with you.

3 Algorithms

The algorithm for the aging process ind strengths as follows:
=(10+2.7*A)*EXP((A*(-0.09)))
The A is the number of days that have passed. Upon birth it is at 100% health, after 7 days it rises to 150%. After 21 days(3 weeks) it's back to 100% and at 84 days(12 weeks) the monster shall be at 0%(Dead).

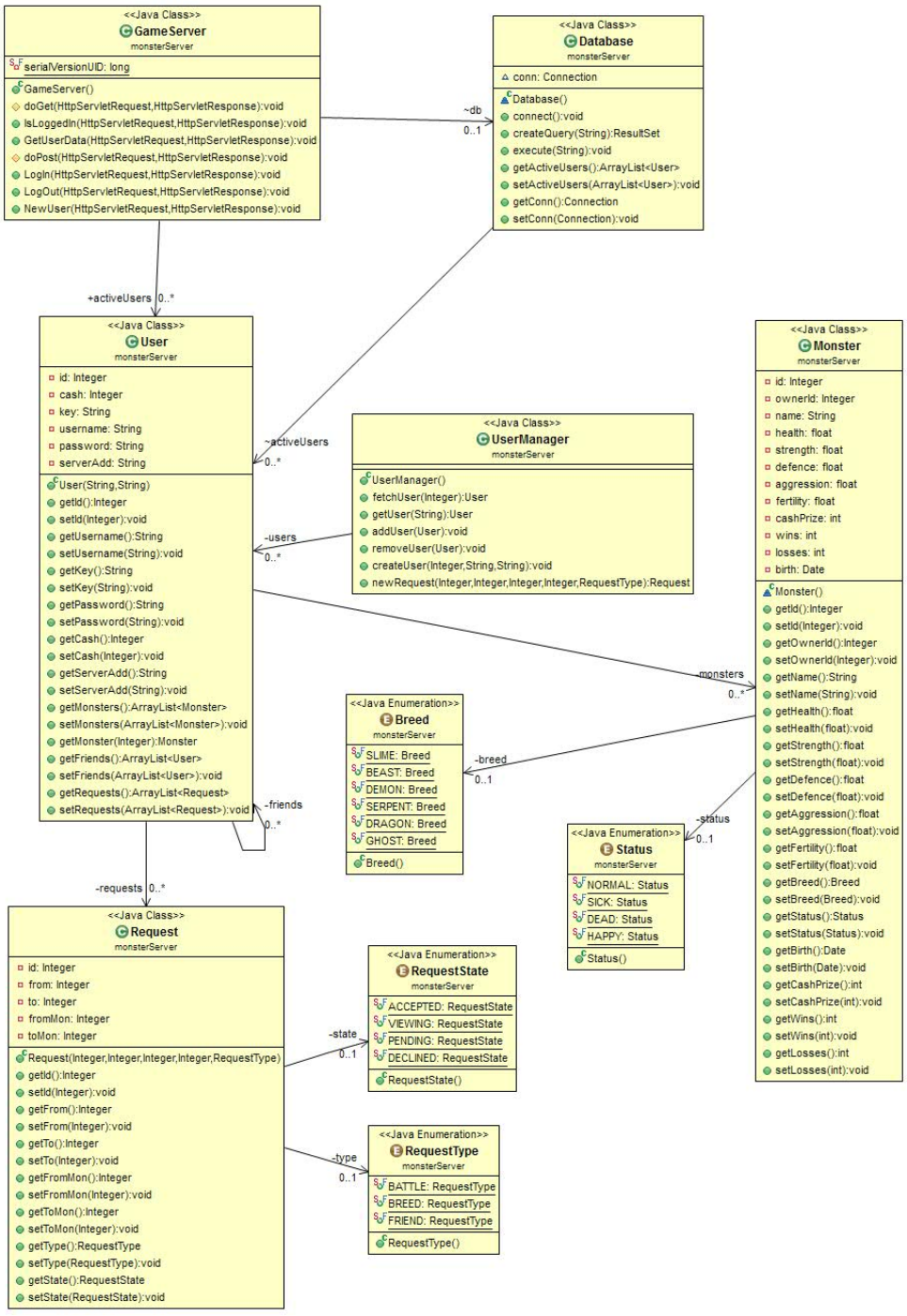
4 JSON Table

Page	Event	Description	Action	Data	Response
Profile	On Load	Request a list of the users monsters	getMonsters	N/a	N/a
Battle	On Load	Request a list of users monsters and a list of the users friends	getMonsters,getFriends	N/a	N/a
Battle	On clicking a friend	Request a list of a friends monsters	getFriendsMonsters	friendId : Int	ID representing a friend.
Battle	On clicking battle	Create a new battle request	newBattleRequest	userMonsterID:Int, friendId:Int, monsterId:int	ID of the selected monster, ID of the friend we are battling with, ID of our friends monster
Breed	On Load	Request a list of users monsters and a list of the users friends	getMonsters,getFriends	N/a	N/a
Breed	On clicking a friend	Request a list of friends monsters	getFriendsMonsters	friendId:Int	ID representing a friend
Breed	On clicking Breed	newBreedRequest	userMonsterID:Int, friendId:Int, monsterId:Int	ID of the selected monster, ID of the friend we are breeding with, ID of our friends monster	N/a
Friends	On Load	Request a list of friends	getFriends	N/a	N/a
Friends	On Load	Request a list of pending friends	getAllNotifications	N/a	N/a
Friends	Accept Friend Click	Accept a pending friend request	acceptRequest id: Int	ID representing a friend	N/a
Friends	Decline Friend Click	Decline a pending friend request	declineRequest id:Int	ID representing a friend	N/a
Friends	Add Friend Click	Send a request to connect to another user as a friend	addFriend	username: email	The users email address
Notifications Menu	On Load	Request a list of all notifications for the current user	getAllNotifications	N/a	”Notifications”: [”Type”:”BATTLE” , ”ID”:”1” , ”From”:”email” , ”Type”:” BREED ” , ”ID”:”2” , ”From”:”email” , ”Type”:” FRIEND ” , ”ID”:”3” , ”From”:”email” ,] Type can be BATTLE BREED or FRIEND
Notifications Menu	Click accept request	Accept the notification	acceptRequest	id :Int	ID of the notification
Notifications Menu	Click decline request	Decline the notification	declineRequest	Id :Int	ID of the notification

4.1 JSON Explanation

JSON (also known as JavaScript Object Notation) is a form of data interchange that is designed to be "human readable". Derived from the JavaScript scripting language, it shows simple data structures and associative arrays, which they call "objects". This table shows us the various data interchanges that take place within the design of our project. JSON is used as an alternative to XML.

5 Class Diagram



6 Significant Data Classes

6.1 Java Data Classes

There are four main classes within the Java designed to be implemented within the project, these are Monster, User, User-Manager, and Request.

6.2 User-Manager class

The User-Manager class handles many different tasks related to how each of the User profiles are created and organised. This class has creates the ability to add and remove a user from a database, this can be seen in the "addUser" and "removeUser" methods located within the class. This class can also perform other tasks that are important to the functionality of the system, such as the ability to fetch particular User data, which can either be the Users ID, or the Users name (as depicted by a String). Without this class, it would be impossible to create a usable log in page, as the system would not be able to get (fetch) the data that is stored in the database to validate their log in. This class would be absolutely necessary for a functioning registration form for the new users, as without it, no Users could be added to the system.

6.3 User class

The User class handles the more specific information appropriate each user that is registered to the system. This includes information such as the Username and password of each of the registered Users. But this is not all this class holds. The User Class also holds details that are necessary for the management of the user within the game. This includes a method to get how many monsters the User owns, and another method which calculates how much cash the User owns. This class also contains information on which Server the User is registered to, without this method within this class, the game would simply not work, as the online features, such as the battling, mating and friend requests would not know where to look for the other users also wanting to play Monster Mash.

6.4 Monster class

The Monster class will handle all of the functionality regarding monsters that the registered Users may own. This can include many basic things, such as how many battles the monster has won or lost, which User owns the monster, the name of the monster and its birthday. However, other more complex stats of the monster will be included within the MonsterStats class, this includes personality attributes, such as

how aggressive each particular monster is, and also includes the battle statistics (the health, attack power and the defensive power of each monster). This MonsterStats class will also contain the fertility rate of each monster, which will be important in the breeding stage of the game. The idea behind splitting the information between each of the classes, is for added simplicity when coding, it would have been a shame not to include extra features such as the age of each of the monsters, as these extra stats add greater depth to the game.

6.5 Request class

The Request class, as the name would suggest, handles the various requests that each of the registered Users will send to each other while playing the game, these include the options to be able to fight and breed with another Users monsters, and also the ability to become friends with another registered User. Aside from these requests, this class also deals with the notification feature we are implementing within our game, this class will be able to tell the User from these notifications what sort of request another User has sent to them, who has sent it, and the ability to respond to the request.

6.6 Functional Requirements

Monster	FR3, FR4, FR10
User	FR1, FR6, FR7, FR8, F11
User-Manager	FR1, FR2, FR3, FR7
Request	FR6, FR9

Table 1: This table shows the functional of Java classes

7 Breed Class

The Breed class is a Private class that deals with the breeding process of the particular User monsters that are in question. The name of this class is Breed.class. There is an ENUM and it is also the only Public Method located within this class, this ENUM has no parameters. The Breed class is basically a set list of monster types that will be able to breed. We used ENUM so that you can only select a type from that particular list.

7.1 Public Methods

The only public method this class has is ENUM Breed and has no parameters. This is a list of set monster types that the monster will be of type. Used ENUM so can

only select a type from that list.

8 Monster Class

The Monster class is named Monster.class and is a public class. This class contains getters and setters for the monsters attributes. These methods can set and get attributes such as the ID of the monster, the owners (the Users) ID, the name of the monster, the stats of the monster (which include attributes such as the attack, the defence and the aggression of the monster), the breed of the monster, the status of the monster, the cash prize that would be attained by defeating the particular monster and finally the wins and losses each monster has achieved. These methods will set the value for each of these, and will be able to return the value for the monster.

8.1 Public Methods

This class contains getter and setters for monster attributes. The getters and setters are for id, ownerId, name, stats, breed, status, birth, cashPrize, wins and losses. They set the value and return the value for the monster.

9 MySQLDatabase Class

The MySQLDatabase class is called MySQLDatabase.class and is a public class. The MySQLDatabase doesn't contain any public methods. The MySQLDatabase class's goal is to manage the data in our relational database system. This class will store all the information that is important to each particular User. This includes the Username and Password of each of the Users, the monsters they own, and what friends they are with. Without this class the game would simply break, as there would be nowhere to store and retrieve the relevant information for the User.

9.1 Public Methods

10 Request

The Request class is called Request.class and is a public class.

10.1 Public Methods

public User getFrom() - This will return from which is of type User. public void setFrom(User from) - This will set the User from and take User from as a parameter.

public User getTo() - This will return to which is of type User. public void setTo(User to) - This will set the User to value and take User to as a parameter. public Monster getFromMon() - This will return fromMon which is of type Monster. public void setFromMon(Monster fromMon) - This will set fromMon and takes Monster fromMon as a parameter. public Monster getToMon() - This will return a toMon which is of type Monster. public void setToMon(Monster toMon) - This will set toMon and takes Monster toMon as a parameter. public RequestType getType() - This will return type which is of type request. public void setType(RequestType type) - This will set type and take RequestType type as a parameter. These will be used to determine whether it is pending, accepted and declined. public RequestState getState() - This will return state which is of type RequestState. public void setState(RequestState state) - This will set state which is of type RequestState and takes RequestState state as a parameter.

11 Request state

The request state class is called RequestState.class and is a public ENUM class.

11.1 Public Methods

The only method in this class is enum RequestState. This is used to allow 3 states to be identified of which are ACCEPTED, PENDING and DECLINED. These will be used for requests such as battle and breed.

12 UserManager

The User-Manager class handles many different tasks related to how each of the User profiles are created and organised. This class has creates the ability to add and remove a user from a database, this can be seen in the "addUser" and "removeUser" methods located within the class. This class can also perform other tasks that are important to the functionality of the system, such as the ability to fetch particular User data, which can either be the Users ID, or the Users name (as depicted by a String). Without this class, it would be impossible to create a usable log in page, as the system would not be able to get (fetch) the data that is stored in the database to validate their log in. This class would be absolutely necessary for a functioning registration form for the new users, as without it, no Users could be added to the system.

12.1 Public Methods

public UserManager() - public User fetchUser(Integer id) public User getUser(String name) public void addUser(User user) public void removeUser(User user) public void createUser(Integer id, String username, String email, String password)

13 User

The User class handles the more specific information appropriate each user that is registered to the system. This includes information such as the Username and password of each of the registered Users. But this is not all this class holds. The User Class also holds details that are necessary for the management of the user within the game. This includes a method to get how many monsters the User owns, and another method which calculates how much cash the User owns. This class also contains information on which Server the User is registered to, without this method within this class, the game would simply not work, as the online features, such as the battling, mating and friend requests would not know where to look for the other users also wanting to play Monster Mash.

13.1 Public Methods

public Integer getId() - This will return id which is of type Integer. public void setId(Integer id) - This will set id and takes Integer id as a parameter. public String getUsername() - This will return username which is of type String. public void setUsername(String username) - This will set username and takes String username as a parameter. public Integer getKey() - This will return key and is of type Integer. public void setKey(Integer key) - This will set key and takes Ineger key as a parameter. public String getEmail() - This will return email and is of type String. public void setEmail(String email) - This will set email and takes String email as a parameter. public String getPassword() - This returns password and is of type String. public void setPassword(String password) - This will set the password and takes String password as a parameter. public Integer getCash() - This will return cash and is of type Integer. public void setCash(Integer cash) - This sets cash and takes Integer cash as a parameter. public String getServerAdd() - This will return serverAdd and is of type String. public void setServerAdd(String serverAdd) - This sets serverAdd and takes String serverAdd as a parameter. public ArrayList<Monster>getMonsters() - This will return a list of monsters of type ArrayList<Monster>. public void setMonsters(ArrayList<Monster>monsters) - This sets monsters and takes ArrayList<Monsters>monsters as a parameter. public ArrayList<User>getFriends() - This will return a list of friends of type ArrayList<User>. public void setFriends(ArrayList<User>friends) - This sets friends and takes ArrayList<User>friends as a parameter. public ArrayList<Request>getRequests()

- This will get a list of requests of type ArrayList<Request>. public void setRequests(ArrayList<Request>requests) - This will set requests and takes ArrayList<Request>requests as a parameter.

14 Status class

The Status class is called Status.java and is a public enum class.

14.1 Public methods

This class is an ENUM class and defines a set of 4 statuses that monsters can have. NORMAL, SICK, DEAD and HAPPY.

15 RequestType class

The Request class, as the name would suggest, handles the various requests that each of the registered Users will send to each other while playing the game, these include the options to be able to fight and breed with another Users monsters, and also the ability to become friends with another registered User. Aside from these requests, this class also deals with the notification feature we are implementing within our game, this class will be able to tell the User from these notifications what sort of request another User has sent to them, who has sent it, and the ability to respond to the request.

15.1 Public methods

This class is an ENUM class and defines a set of 3 types for request. BATTLE, BREED and FRIEND.

16 Data Storage

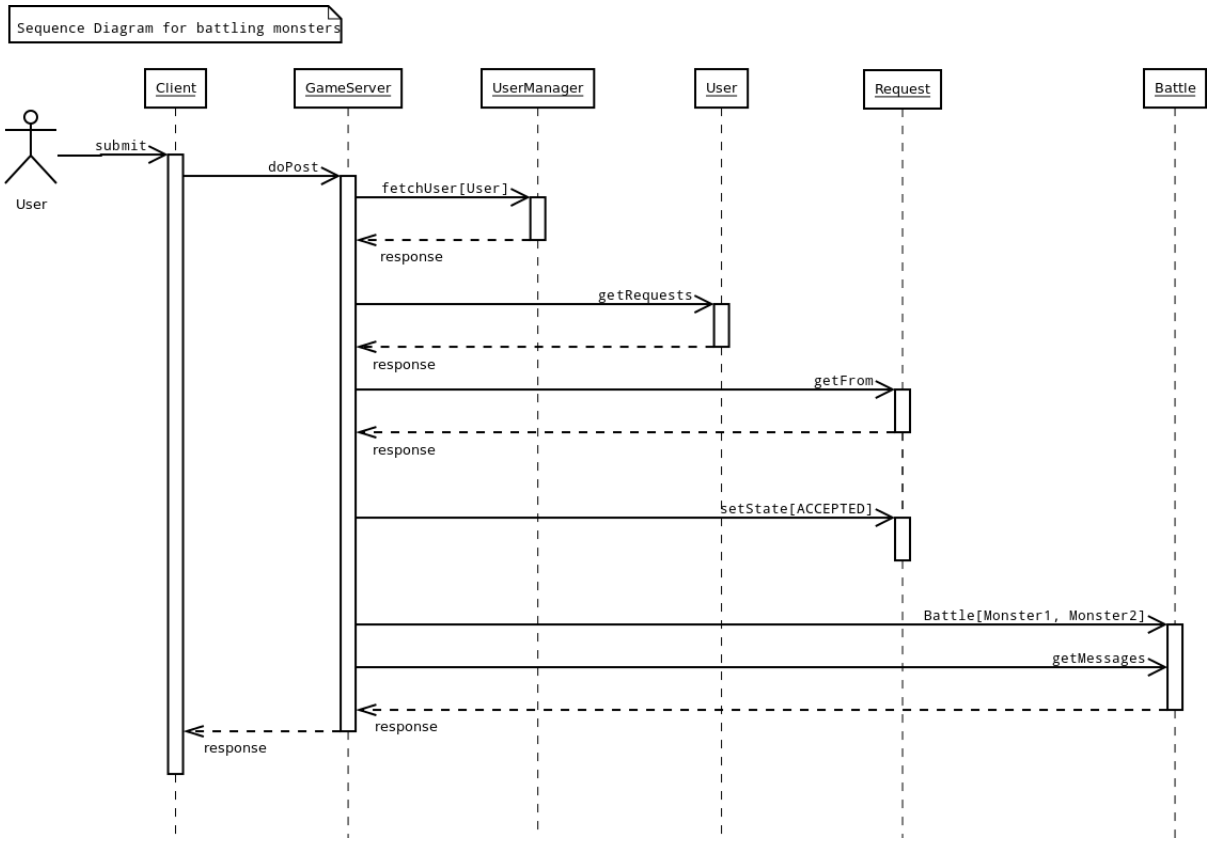
Within the java programing the data is stored in a number of ways, a major way that data will be handled is through enums; requests will be handled through this and will be stored as either battle, breed, or friend, and the states of these requests will also be stored as enums, being either accepted, viewing, pending, or declined. Also stored as enums will be the statuses of monsters, thusly each monster status will be handled as normal, sick, dead, or happy. Breed considers the different types of monster that we plan to be breeding with each other, and this therefore will be stored as an enum with the value of slime, beast, demon, dragon, serpent, or ghost.

Array Lists will also be used, but only, as far as designed, in private instances, so there will be no public instances that need to be explained in class diagrams. Otherwise, usernames, passwords, and the like will be handled in private strings and variable types as are appropriate.

17 Sequence Diagrams

17.1 Accept Battle

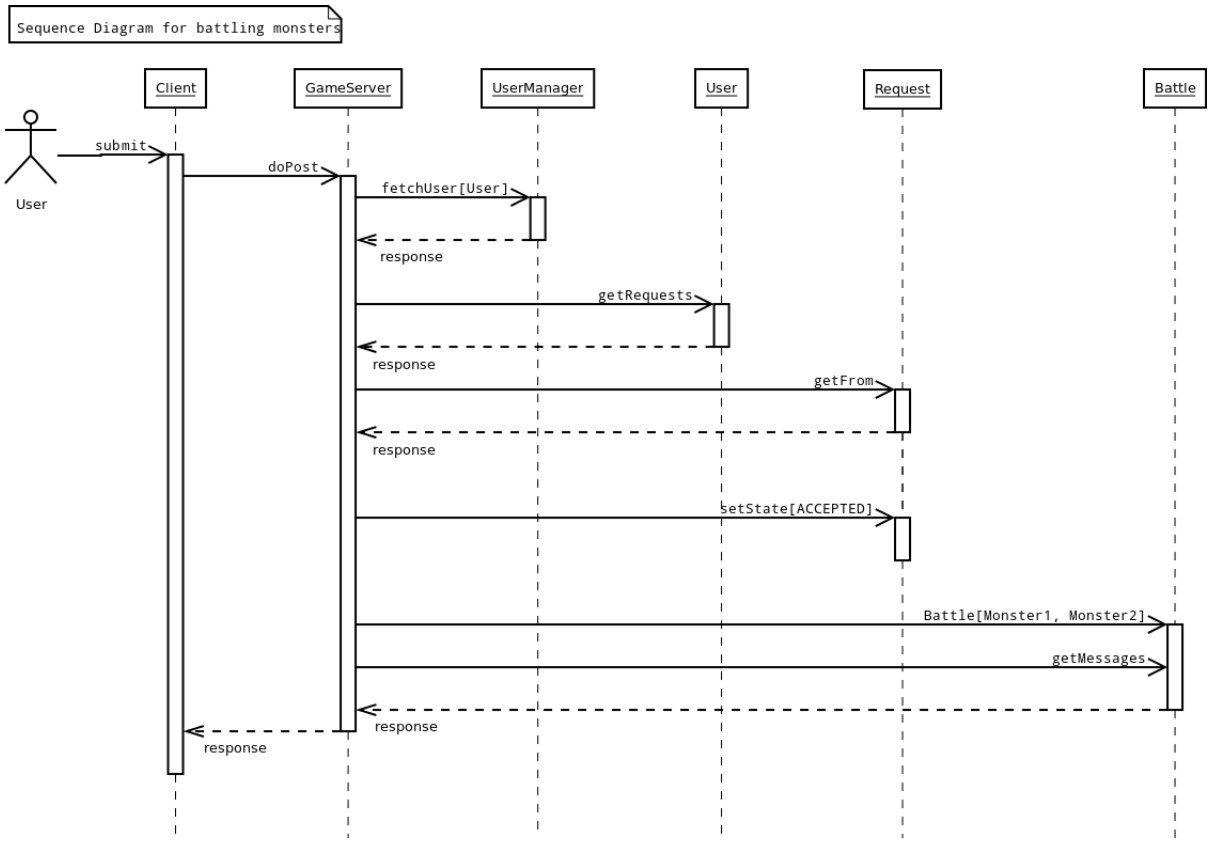
This sequence diagram shows a user accepting a battle request that has been sent to them by a friend. The diagram shows the users response being sent to the server using the doPost method. The request then runs through the relevant classes in order to gain the information required.



17.2 Accept Breed

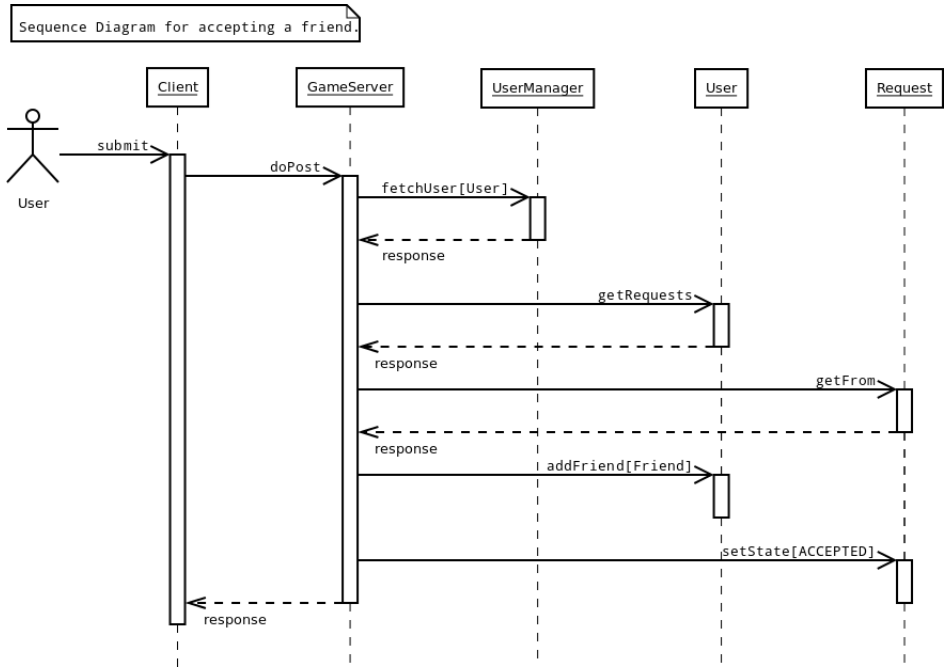
This sequence diagram shows a user accepting a breed request that has been sent to them by a friend. The diagram shows the users response being sent to the server

using the doPost method. The request then runs through the relevant classes in order to gain the information required.



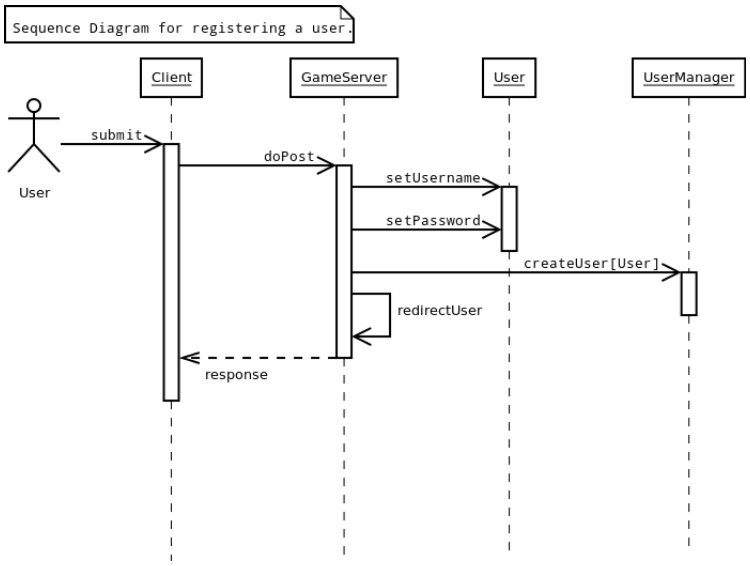
17.3 Add Friend

This sequence diagram shows a user adding a friend. The users request is sent to the server using the doPost method. The request then runs through the relevant classes in order to gain the information required. A request is then sent to the corresponding user using the addRequest method and User class.



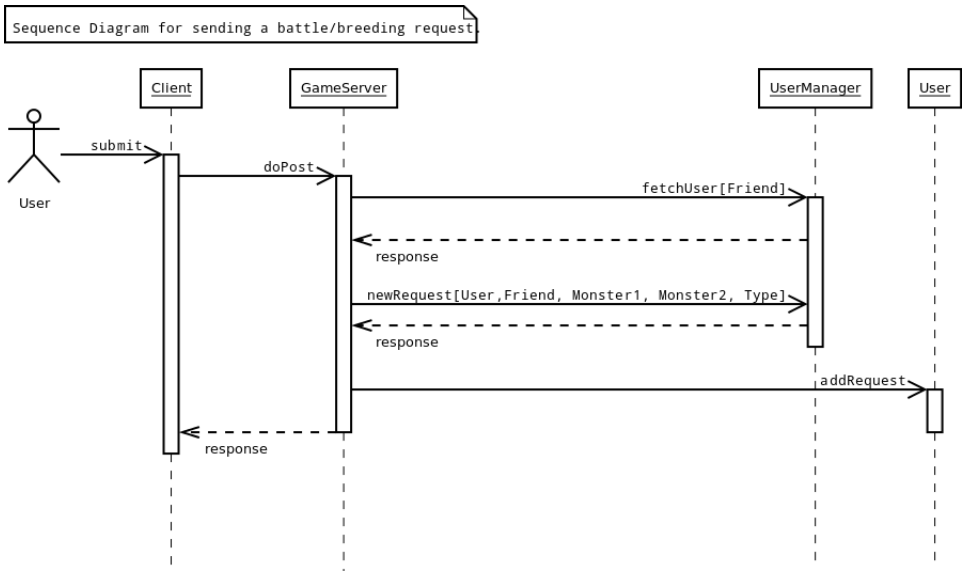
17.4 Register User

This sequence diagram shows a user submitting their password and username to create a monstermash account. The diagram shows the users password and username being sent to the server using the doPost method. The request then runs through the relevant classes in order to process the users details and create the new account.



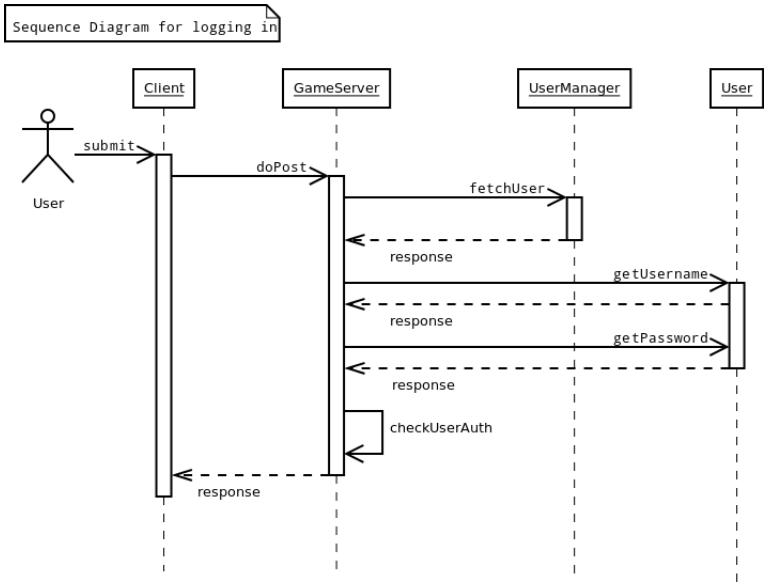
17.5 Send Battle/Breed Request

This sequence diagram shows a user sending a breed or battle request to a friend. The diagram shows the users request being sent to the server using the doPost method. A request is then sent to the corresponding user using the addRequest method and User class.

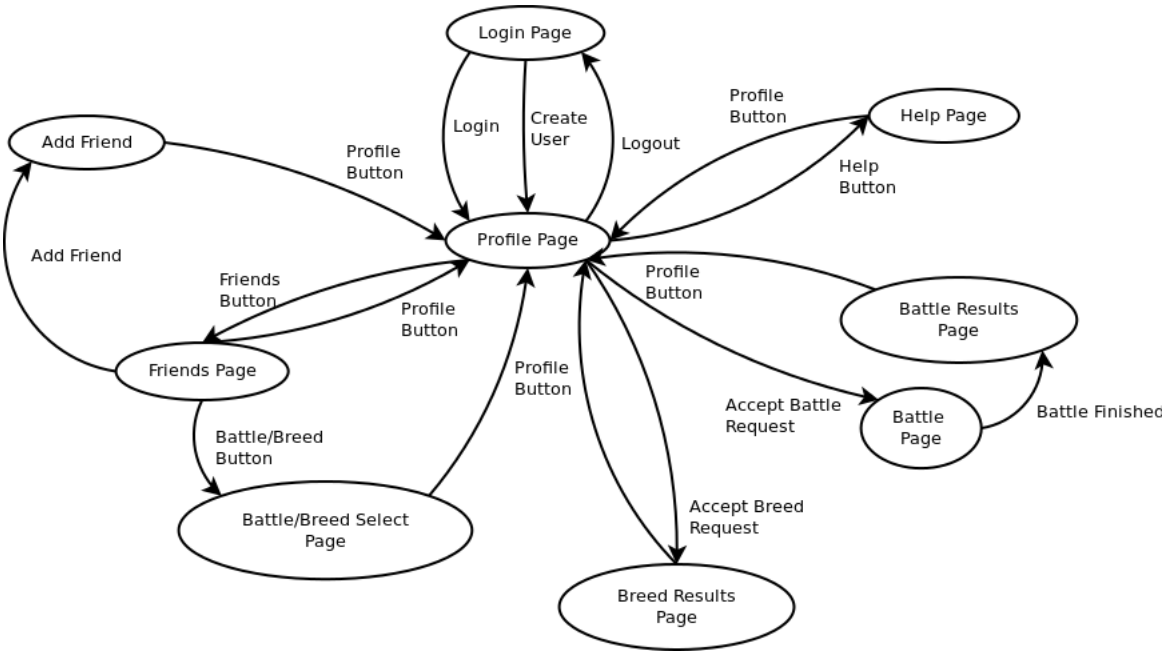


17.6 User Log In

This sequence diagram shows a user logging in to monstermash. The user submits their login information and it is posted to the server using the doPost method. The request then runs the the relevant classes, checking the login details. A response is generated if the login details are invalid or the user is logged in if they details are valid.



18 State Diagram



18.1 State Diagram Description

As you can see, the diagram above shows the design structure for how we want to be able to navigate each of our web pages. I will go through each connection individually, and explain the reasoning behind each.

18.1.1 Login Page to Profile Page

This is the first page that the user will be able to see when going onto our website, the page will require the user to enter their username and password into the allocated areas, once this is done, the user will be taken to the profile page, this is because the profile page is regarded as the users most important page, as this is where the user can decide what s/he plans to do.

18.1.2 Profile page to Various pages

The reason we had the profile page link to most of the pages on the website, was simply because this page acts almost as the home page of our structure, and it makes it easier for the user to have just one page in which they can select most of their options. If we separated and spread out each of the links to various pages to other pages in the structure, this would likely confuse and frustrate the user, by having them all in the same place, this avoids most issues about navigation that may arise.

18.1.3 Profile page to Help page

It is most likely that if the user had a problem, they would not be on any page other than the profile page, as this would be the starting point for the user, for this reason, we linked the Help page only too and from the profile page.

18.1.4 Profile page to Battle page

As the Battle page one of the most important pages (Alongside Friends) it was important that we put the link to this page in an obvious place, and there is no place more obvious than on your Profile page, as (as was explained before) this page works as the home page of this website, a page that is constantly referred to.

18.1.5 Battle page to Battle results page

After you have accepted an offer of a fight, the user will be taken to an area called the Battle page, in which you can watch the battle between your monster and the

opponent that challenged you, after the battle has been completed you will be taken to the Battle results page, we structured this so that if you are the user that sent out the battle request, you will automatically be taken to the Battle Results page, as (as we have explained previously) the user sending out battle requests does not get to view the battle, but just sees the results. After the users have seen the results of their battle(s), they have the option to return to the Profile page, so that they can continue to do other tasks within the game. Profile page to Friends page:

The link to this page is very important, it is from the Friends page you will be able to add and accept who you want to be friends with in monster mash, who you would like to battle with, and whos monsters you would like to breed with. Without this page, the game simply would not work. So for this reason, we linked this page to the Profile page, similar to the reason with the Battle page, this was done so the user can find this page easily, as the Profile page will be the most visited page on the website.

18.1.6 Friends page to Various pages

From this page, it is possible to do three separate things, you will be able to manage your friends, select which monsters you wish to breed with, and select which monsters you wish to battle. We decided to put these options on this page and NOT on the Profile page, because you would begin to run the risk of cluttering the Profile page, as important these tasks are to the game, we concluded that it would be much more neat and less confusing to put these options within a separate Friends page, so the user wouldnt be drowned with options on the Profile page.

18.1.7 Friends page to Battle/Breeding page

Originally, we were going to have the Battle and Breeding pages separate here, so the user would be able to select which one they wanted. It was only later on that we realised that both these tasks could be accomplished on the same page, so having them on the same page was unnecessary. Once the user has selected whether they want to breed or battle certain monsters, the user will be directed back to the Profile page, this is a running theme in our structuring, as this is where everything on the website stems from.

18.1.8 Friends page to Add Friend page

It would seem fairly obvious that from the Friends page, we would be able to select which friends we would want to add, so from here, we added an Add Friends page, in which the user can type the recipients email or username, to send a friend request. Once the user has finished adding the friends they wish, they can then link back to the Profile page, to continue with other tasks they may wish to do.

18.1.9 Profile page to Breeding Results page

This link is very similar to the Battle page explained earlier, the only difference being that it does not matter who was the recipient of the offer or not, they would both be linked straight to a breeding results page. Once the user has looked at the results, they can then link back to the Profile page.

18.1.10 Profile page to Logout

For a small while, we did consider having a logout page on every page of the website, but we theorised that this may cause problems that would be difficult to fix, for example, what would happen if you logged out in the middle of a battle on the Battle page. For this reason, to avoid confusion, we only put the logout page on the Profile page, as this guarantees that the user is not in the middle of a task.

1 Management Summary

This project has managed to achieve 25/27 of the functional requirements it was aiming to complete. See (cite requirements doc) for list of functional requirements. The only two that were not managed to be completed on time were, monster aging and server to server. The mosters ageing requirement, although not implemented, there was a formula that we had that worked. Given a bit longer on the project I believe we could have got the monster ageing working. The server to server side however was a bit further off. Silhab Csoma had a look at the server to server but due to the change in requirements and internet down time there was not time to implement server to server.
All the documents are in a good state——elabourate——.

A few difficulties got in our way during the development of the system. One of these issuses was the change of requirements. Originally it was specified that the monster battle and breed would be offered up or requested but as the deadline approached the functional requirements were changed to specify that breed and battle requests had to be offered. This meant that part of the system had to be redesigned which took time off working on bugs and other functional requirements.

Another major issue that arose was that there was internet downtime in the week before the deadline. This caused several issues, the first of which was the inability to test the system properly. This lead to work on the system being slowed down and thus resulted in a less complete system. It also meant more two of the coders(Sam Jackson and Chris 'Tux Lloyd) had to put in more hours to complete what was necessary to get a functional and bug free program. The second issue the internet downtime caused was a loss of work as when uploading the latest copy to git there was an error in doing so. We then tried to access the project which was stored on the M: drive however, this was unaccessable. We decided to leave the university grounds and head down into town to find somewhere with a working internet connection. Here the two coders managed to get access to the temporarily back up internet and M: drive and obtain a copy of the latest code. From here it was then uploaded to GitHub with an issue that meant the latest CSS code would be Lost. This meant Aleksandra Badya had to put more work into finishing of the design of the system.

The team preformed outstandingly overall. However special credit has to go to Chris 'Tux' Lloyd and Samuel Jackson as they worked perfectly as a team. Between them they managed to sort out a large quantity of the bugs and get a large chuck of the system operational. The rest of the team fulfilled there roles and more, they managed to do what was asked of them and more.

This should sum up in one page what the project achieved (what parts of the program work and what parts do not; which documents are in a good state and which are not), what difficulties stood in the way of project completion and how

they were overcome, and how well the team performed.

2 Historical account of project

(look at gannt chart)

This should outline the main events over the lifetime of the project, and how the project team acted to produce a plan and to deliver a product within a constrained lifetime. This should take no more than two pages of A4.

3 Final State of project

This should give a summary of which parts of the project are perceived as correct and which are not. It is as well to be as accurate as possible here - more marks will be deducted for problems that are not declared but are detected by the markers than for problems that are declared in the final report. As well as missing or erroneous features in the software, known problems with documents should be included here.

4 Performance of each team member

4.1 Tom Reed

Tom's duties as Q.A. Manager were to manage other documentors and delegate documenting tasks where he felt appropriate. He preformed brilliantly especially as the deadline approaced, he assigned tasks well and made sure they were completed. He also kept Christopher Marriott(Project Manager) up to date on goings and asked for more work when finished allcurrent work. He was an essential part of the team.

4.2 Chris 'Tux' Lloyd

Tux preformed outstandingly. His time and effort put into the code was brilliant and he worked well with the rest of the team. He informed people of changes and explained things clearly. He worked especially well in tandem with Samuel Jackson. He put countless hours of work into the project and helped solved numerous bugs.

4.3 Samuel Jackson

Sam's preformance was outstanding. Not only was he vital in the functionalality of the overall system but on occasion preformed as a brilliant project leader when Christopher Marriott was away.

4.4 Silhab Csoma

Silhab's work on data structions and work on algorithms was essential to the working of the system. Without these in place there would be no way of storing monster and user info. She worked well in a team and completed all tasks set for her.

4.5 Alexsandra Badya

Alex worked well in the team. She provided vital support to the team with git did great work with the html and css. She also worked well as deputy Q.A. manager when Tom Reed was away. The team were kept up to date with duties they had to complete. She performed very well on the whole project.

4.6 Dave Clark

Dave was a very good documentor in all areas. He mostly focused on Tux's server side documentation however he also work on other documentation. He also helped find bugs with in the system and inform people of what they were. Other areas he worked on were the database diagram, methodology and reviewing functionality. He also did the minutes for several of the team meetings.

4.7 Mike Steel

Mikes main duty was to document Silhab's work. He performed well within the team and helped in other areas. He helped out on the testing side of the system and fixing the LaTeXdocuments.

4.8 Matt Whitmore

Matt's job was to document and test Alex's HTML and CSS. He completed all tasks on time and worked well with the team. He also helped out on the testing of the system and found some bugs of which he informed the appropriate people.

4.9 Christopher Marriott

Chris' main role was project manager. As project manager it was Chris' duty to assign tasks to other team members, review documents and check that people are completing their tasks. He communicated to the group regularly and organised meetings, he also on occasion took the minutes.

The project leader should write a half page description of the duties and performance of each group member, including the group leaders themselves. This should be agreed with the group member if possible, and it should state whether agreement was reached, and if not, should give an explanation why not.

5 Critical evaluation of the team and project

This should be no more than a page in length and should address the following subjects:

5.1 Team performance

The teams performance was brilliant. A few team members also went above and beyond the call of duty to get the system more or less fully functional. Everyone knew their role and were happy with the tasks they were set. If anyone had a problem they spoke to the appropriate person and got it resolved reasonably quickly. Everyone did their roles excelently and fulfilled all that was required and more. Each member of the team communicated clearly with other team members and kept up to date with what had been done and what had not. Each team member was kept up to date and informed of new tasks via a facebook page and a gannt chart. Everyone seemed happy with this approach and there was no major disruptions from within the group.

5.2 Improvements

The project could have been improved in a few ways. The first way would have been to assign Samuel Jackson as the project manager as he was more knowledgeable in the system and would therefore been able to assign more details tasks. The second would have been for Christopher Marriott(Team leader) to have informed the team of new tasks that needed doing and keep up to date with where team members were at with their current task.

5.3 Lessons learnt

Some of the lessons that were learnt during this project were to make sure people understood their role more clearly. To plan the more important and required parts of the system before worrying about how it will look.OTHER GROUP MEMBERS LEARNING EXPERIENCES HERE

End report/1(draft)

REFERENCES

[1] *N/A*

DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2013-01-30	Initial creation	cpm4