Software Development Life cycle
Design Specification

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:	2012-12-5
Version:	1.3
Status:	draft

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

CONTENTS

1 Introduction

1.1 Purpose of this Document

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:

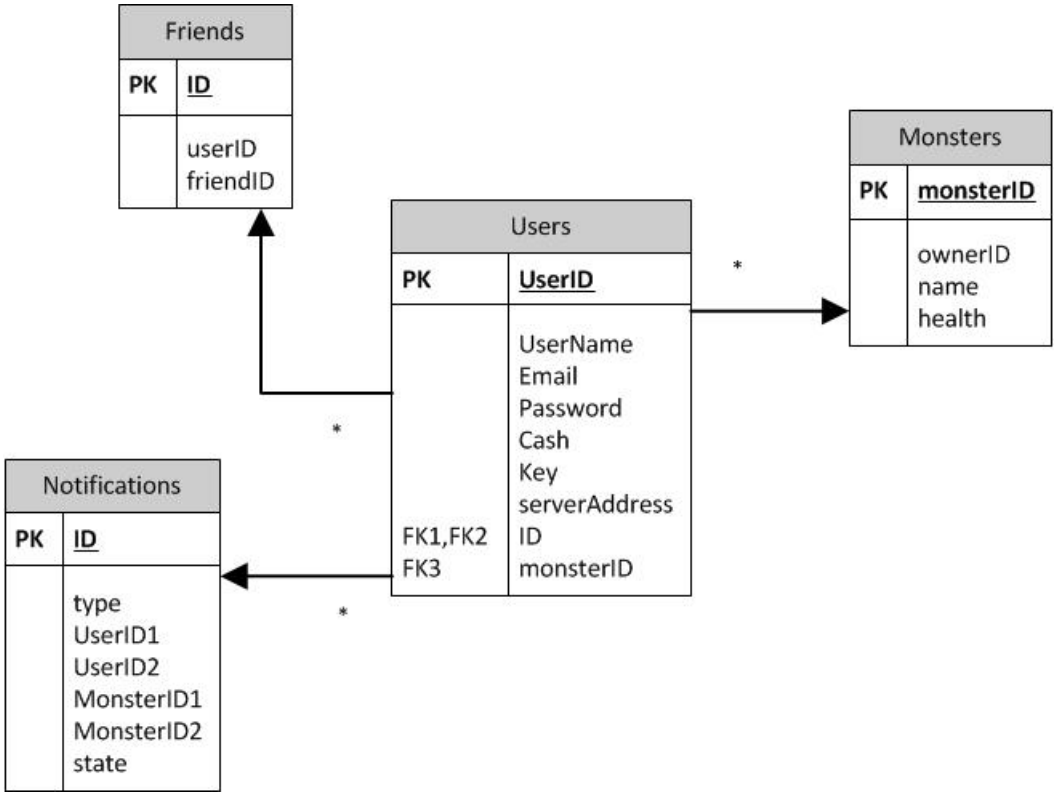
- Java Classes explanations
- Sequence diagrams
- Relational Database diagram
- JSON Table
- Algorithms

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	
Battle	On Load	Request a list of users monsters and a list of the users friends.	getMonsters	N/a	
			getFriends	N/a	
	On clicking a friend	Request a list of a friends monsters.	getFriendsMonsters	friendId : Int	ID representing a friend.
	On clicking battle	Create a new battle request.	newBattleRequest	userMonsterId: Int	ID of the selected monster.
				friendId: Int	ID of the friend we're battling with.
				monsterId: Int	ID of our friends monster.
Breed	On Load	Request a list of users monsters and a list of the users friends.	getMonsters	N/a	
			getFriends	N/a	
	On clicking a friend	Request a list of a friends monsters.	getFriendsMonsters	friendId : Int	ID representing a friend.
	On clicking breed	Create a new breed request.	newBreedRequest	userMonsterId: Int	ID of the selected monster.
				friendId: Int	ID of the friend we're battling with.
				monsterId: Int	ID of our friends monster.

Friends	On Load	Request a list of friends.	getFriends	N/a	
		Request a list of pending friends.	getAllNotifications	N/a	
	Accept Friend Click	Accept a pending friend request.	acceptRequest	id: Int	ID representing a friend
	Decline Friend Click	Decline a pending friend request.	declineRequest	id: Int	ID representing a friend
	Add Friend Click	Send a request to connect to another user as a friend.	addFriend	username: email	The users email address.

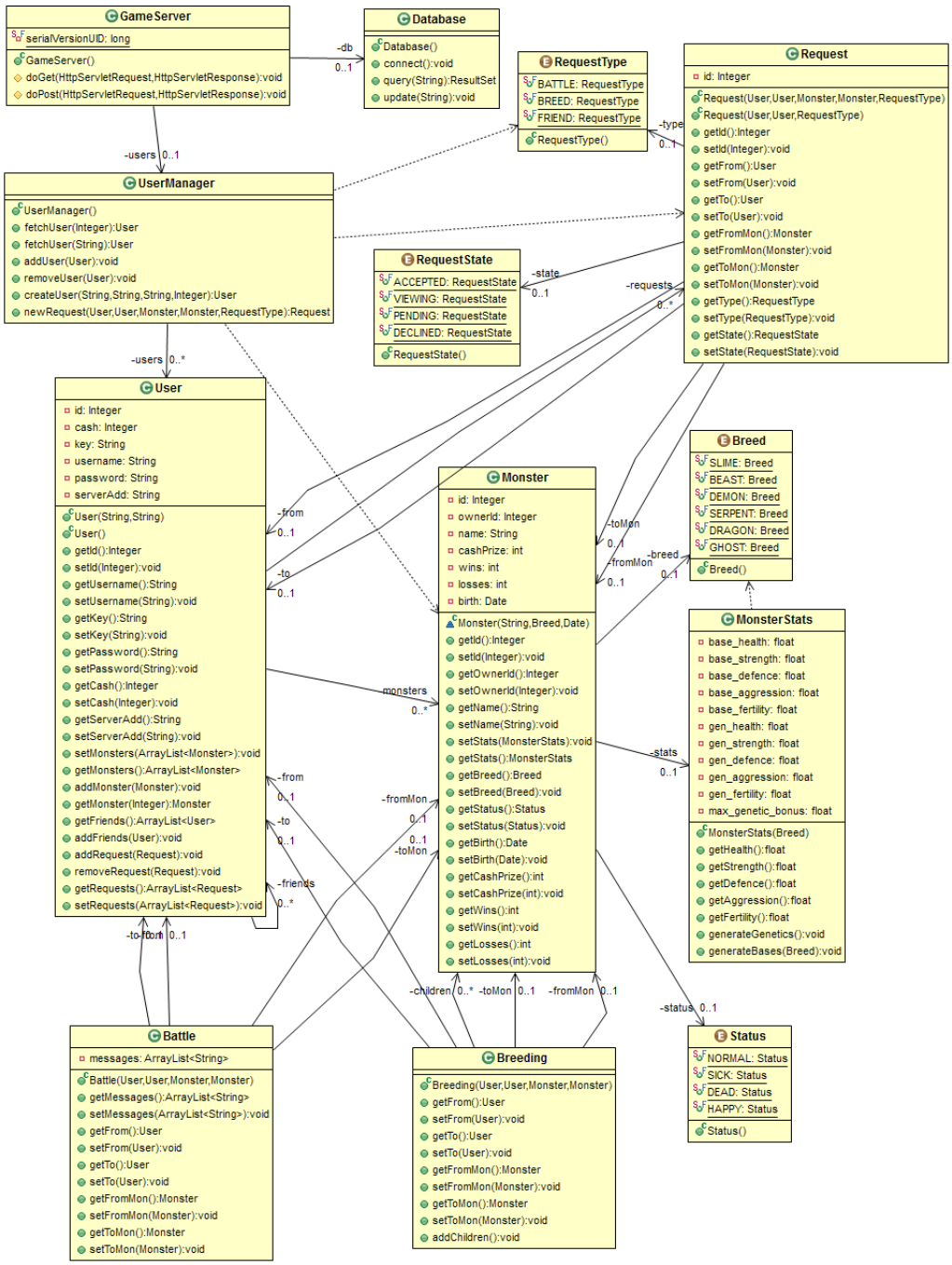
Notifications Menu	On Load	Request a list of all notifications for the current user.	getAllNotifications	N/a	{
					"Notifications": [
					{ "Type":"BATTLE" , "ID":"1" , "From":"email" },
					{ "Type":" BREED " , "ID":"2" ,
					"From":"email" },
					{ "Type":" FRIEND " , "ID":"3" ,
					"From":"email" },
					]
					}
					
					Type can be BATTLE BREED or FRIEND

	Click accept request	Accept the notification.	acceptRequest	id :Int	ID of the notification
	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

User-Manager handles the basic ability to get a user from the database and find out if they exist, as well as being able to add and remove a user from the database. This class is therefore key to having a usable log in page, as well as being absolutely necessary for a functioning registration form for new users.

6.3 User class

The User class handles more specific details appropriate to a user, including both details necessary for the user logging in, such as getting the user-name and password, as well as details necessary for the management of the user within the game, such as methods to get which monsters the user has. There will also be parts of this class that handle setting the user-name and password and other key parts of the users details, so this is a class key to many functions of the website; from creating a new user and logging in, to finding out what monsters and friends that user has.

6.4 Monster class

There is then the Monster class which will handle all the functionality regarding monsters that a user might own, including basic things such as who owns the monster, its stats, and what its name is, to key details that add depth to the game, such as the monsters age, how many battles it has won, and how fertile it is.

6.5 Request class

Finally, there is the Request class, which will handle requests from other players to breed, fight, or become friends. This will handle everything to do with notifying the recipient of these requests, such as noting who the request is from, and responding to it.

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

6.6 Functional Requirements

7 Breed Class

The breed class is called Breed.class and is a private class.

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 called Monster.class and is a public class.

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.

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 is called UserManager.class and is a public class.

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 is called User.class and is a public class.

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 type class is called RequestType.class and is a public ENUM class.

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

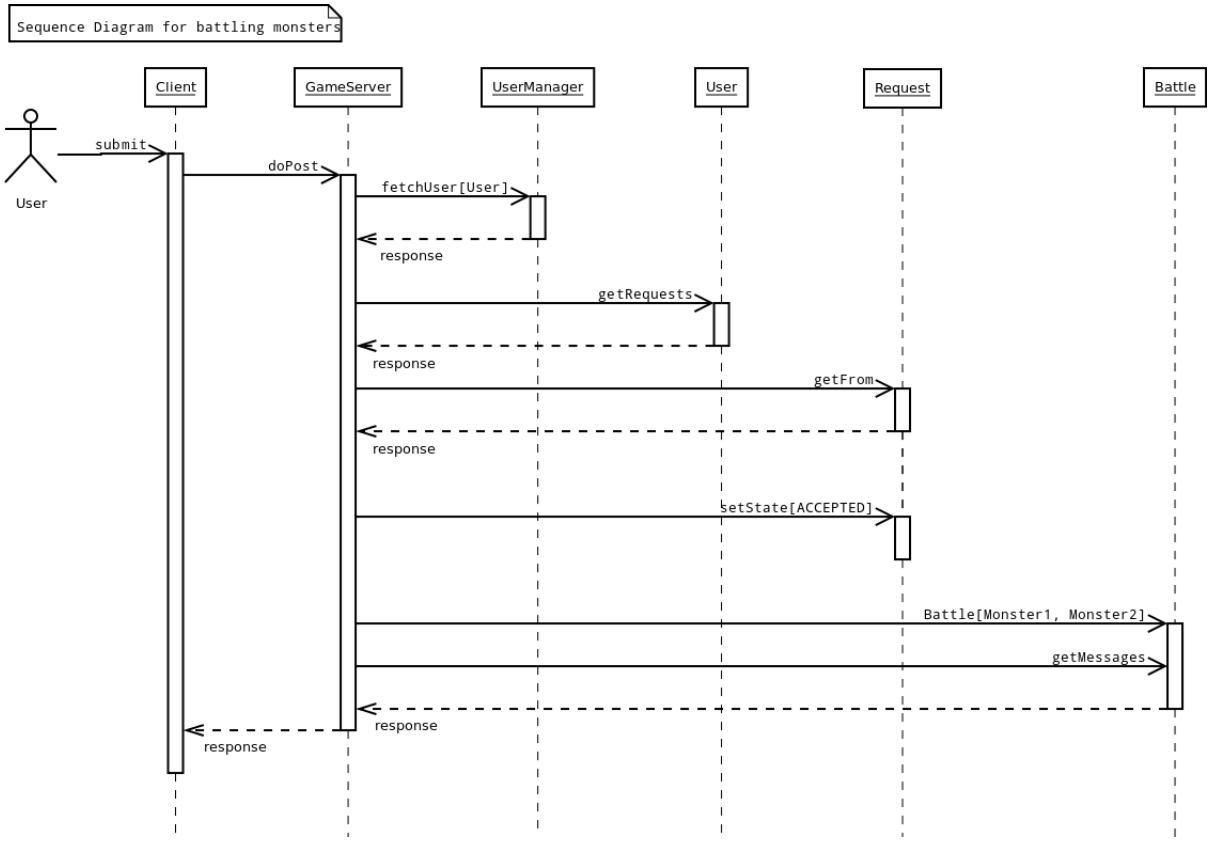
Within the java programing the data is stored in a number of ways, a major way that data will be handled is through enums; requests will be handled through this and will be stored as either battle, breed, or friend, and the states of these requests will also be stored as enums, being either accepted, 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

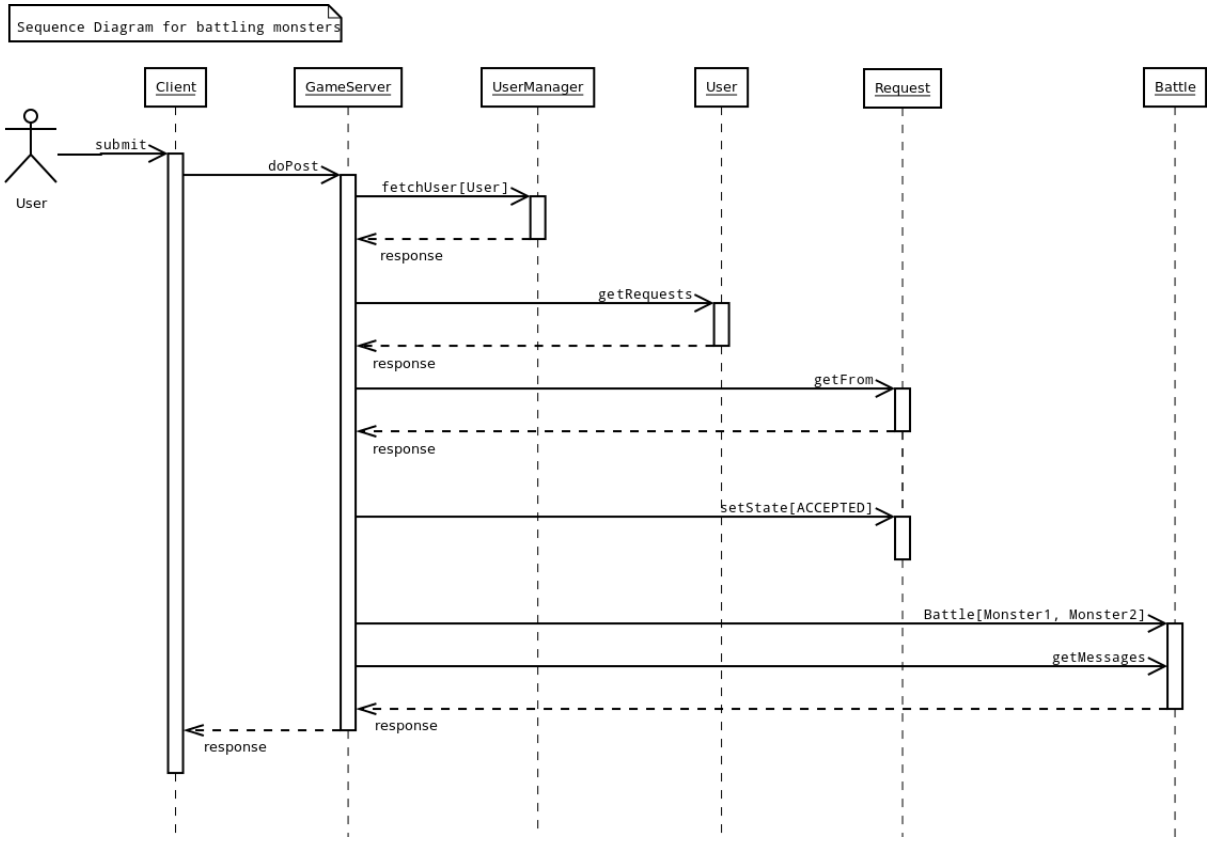
17.1 Accept Battle

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.



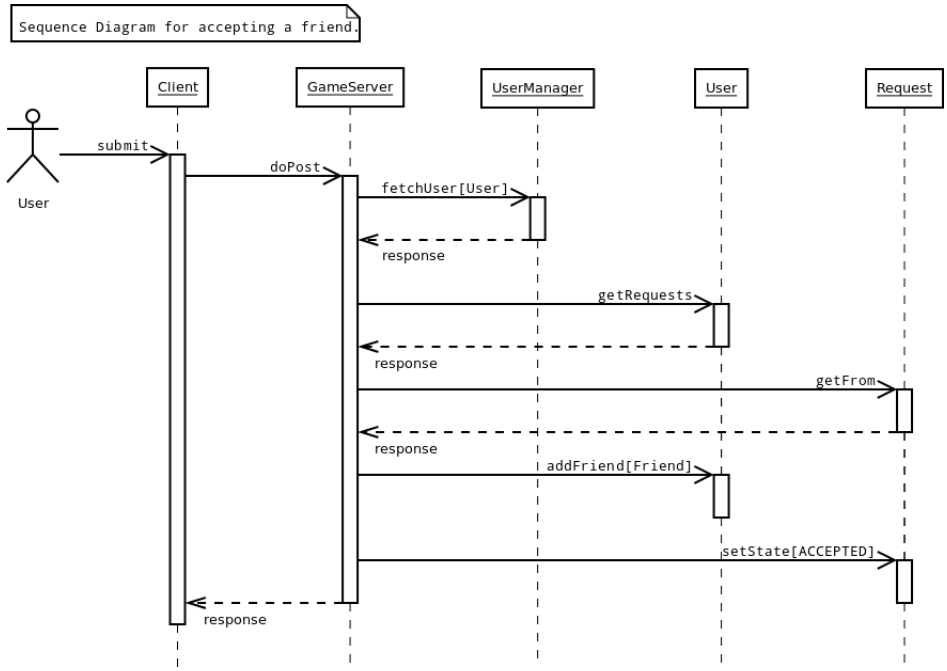
17.2 Accept Breed

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.



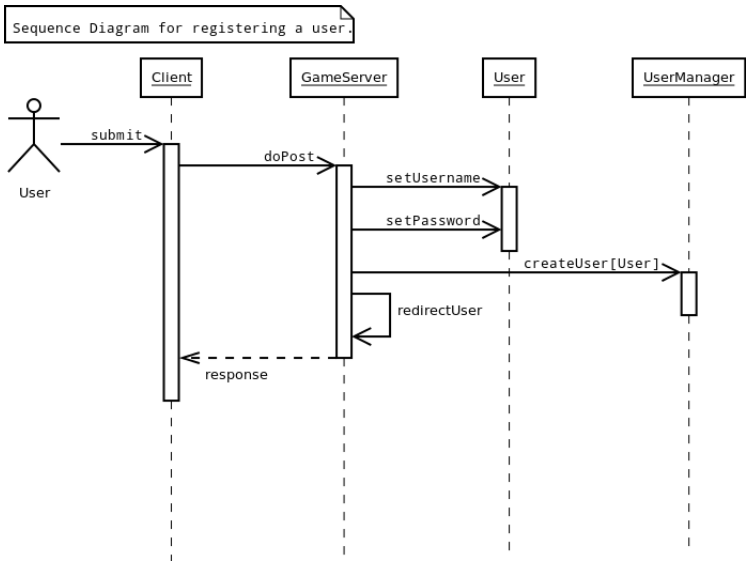
17.3 Add Friend

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.



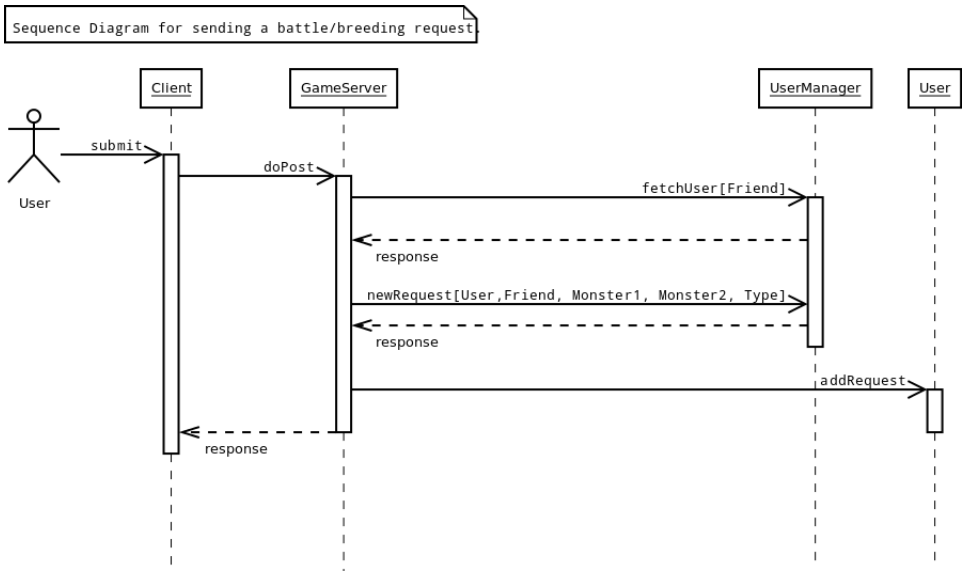
17.4 Register User

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.



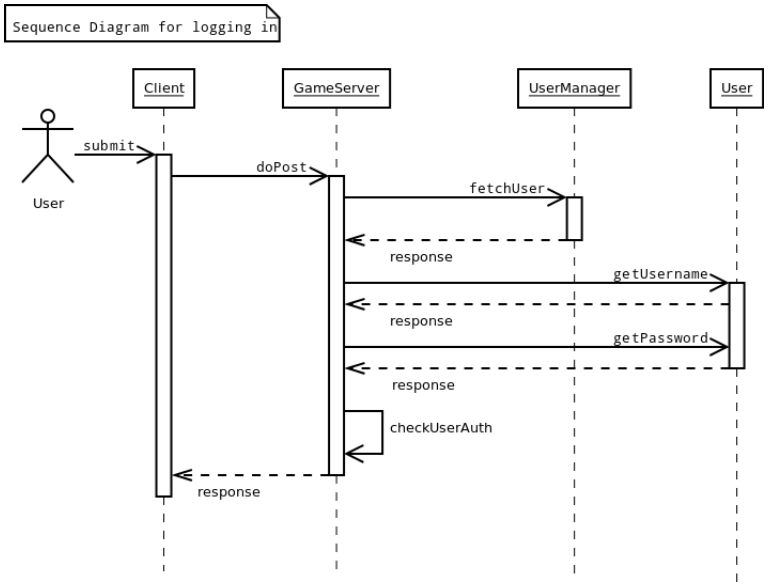
17.5 Send Battle/Breed Request

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.

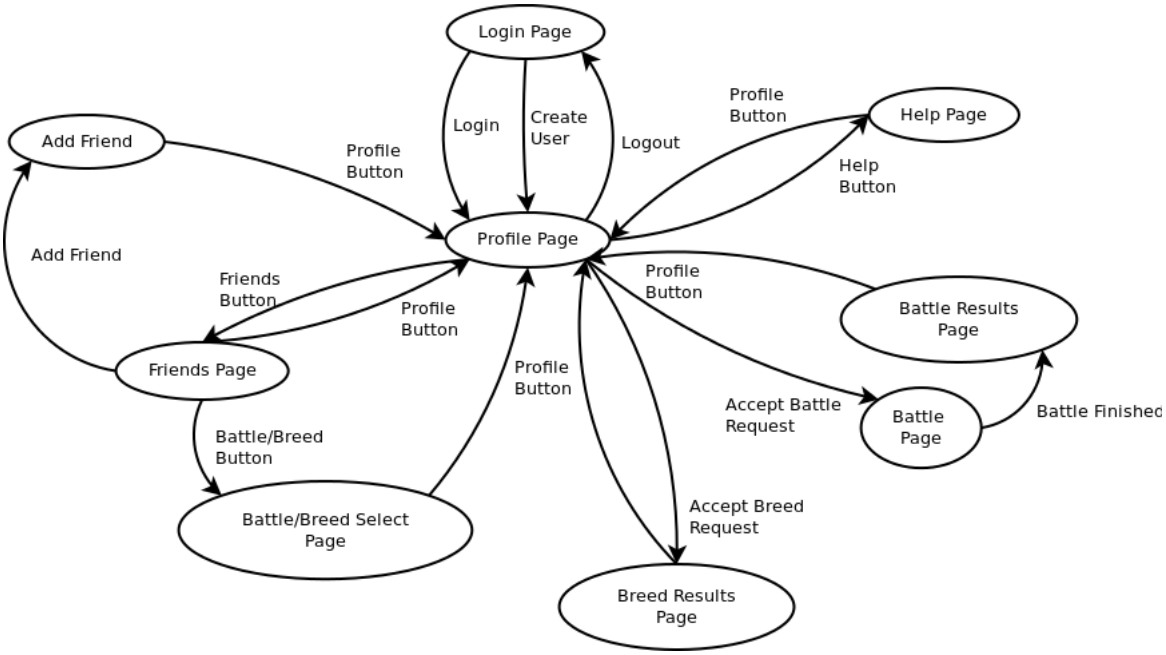


17.6 User Log In

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



REFERENCES

[1] *N/A*

DOCUMENT HISTORY

Version	CCF No.	Date	Changes made to Document	Changed by
1.0	N/A	2012-10-31	Initial creation	CPM4
1.1	N/A	2012-11-2	Added information from Mike	CPM4
1.2	N/A	2012-12-5	Updated config ref and added other documents	CPM4
1.3	N/A	2012-12-6	Added missing data and fixed few mistakes	CPM4