Group 06 - Design Specification

3rd December 2012

Amy Rebecca James, James Slater, Dan Mcguckin, Samuel Mills, Felix Farquharson,
Ben Brooks, Aiman Arafat, Chris Arom
Project Coordinator: Nigel Hardy

Version: 2.0
Status: Draft

Computer Science Department, University of Wales, Aberystwyth

Contents

1 Introduction

1.1 Purpose of the Document

The main purpose of this document is to identify and describe the format of information which is given in the design specifications document produced by CS22120 Group 06. Each section will be individually described and will explain the design aspect of each feature.

1.2 Scope

The design specification shows the individual features which relate to the Monster Mash game. This document specifies the description of each component within the design and defines the details of each feature. It is important that this document is read by all members of the project group, especially the Design team.

1.3 Objectives

The main objective of this document is to reinforce the production of a design specification. It has to be complete, up to date and an accurate interpretation of the users requirements into a description of the design elements necessary for the implementation and development phase. These will generally include the outline structure, decomposition description, dependency description, interface description and a final detailed design.

2 Outline Structure

The outline structure for the design specification is presented in Figure 1. The structure consists of 5 features which are listed and described within the document. Firstly the introduction briefs the reader on what the intention of the document is, who it is aimed at and what its main objectives are as a whole. The decomposition description describes the applications within the system and the classes of the applications. The dependency description explains how the parts of the system come together and how they are dependent on each other. The interface description describes each class and function and what their roles are. Finally the detailed design shows the Monster Mash class diagram and the sequence diagram.

2.1 Figure 1 - Structure for Design Specification

1 Introduction
- 1.1 Purpose of the Document
- 1.2 Scope
- 1.3 Objectives

2 Outline Structure
- 2.1 Figure 1 - Structure for Design Specification

3 Decomposition Description
- 3.1 Applications in System

4 Significant classes in each package
- 4.1 Significant classes in the Model package
  - 4.1.1 User
  - 4.1.2 Monster
  - 4.1.3 PersistManager
- 4.2 Significant classes in the Controller package
  - 4.2.1 Login
  - 4.2.2 Shop
  - 4.2.3 FriendsList
  - 4.2.4 TradeMonsters

3 Decomposition Description

3.1 Applications in System

In our Monster Mash design, there will be only one application. This handles everything from serving JSPs that a user can see in their browser, to the database connections, inter-server connections and data processing.

4 Significant classes in each package

4.1 Significant classes in the Model package

4.1.1 User

The User class contains fields to hold the user's data for things such as Name, Password, Money, Friends, etc. Each user can have between 1 and 6 monsters. You can also set the user's default battle monster from the changeBattleMonster method in this class. A user is created using the UserFactory class.

4.1.2 Monster

This class contains all of the monster's stats. For example, height, age, strength, sex, etc. A monster is created via the MonsterFactory class.

4.1.3 PersistManager

PersistManager is the class which can create, update or delete a user or monster. Search for users or monsters and start or stop the main application. It is also this class which talks to the User database, Monster database and shop database.

4.2 Significant classes in the Controller package

4.2.1 Login

Login contains methods that will allow a user to login or register themselves.

4.2.2 Shop

Shop contains methods that let a user buy or sell monsters.

4.2.3 FriendsList

FriendsList contains methods for managing a user's friends. E.g. addFriend, removeFriend, getFriend, etc.

4.2.4 TradeMonsters

TradeMonster contains a method for trading two monsters.

4.2.5 Breed

The Breed class has methods which calculate a cost of a monster, creating a child and checking if monsters are of the same sex.

4.2.6 SimulateFight

SimulateFight has methods to calculate a winner of a fight between two monsters and calculate the prize amount of a fight.

4.3 Mapping from requirements to classes

Requirement	Classes Providing Requirement
FR1 (Server-based authentication)	Login, CreateMonsterAndUser, PersistManager, User and UserFactory
FR2 (Server friends list)	FriendsList and PersistManager
FR3 (Server monster list)	CreateMonsterAndUser, PersistManager, User and Monster
FR4 (Server monster mash management)	SimulateFight, FriendsList, PersistManager, User and Monster
FR5 (Server-server communication)	PersistManager
FR6 (Client options)	Login, Shop, FriendsList, Trade, Breed, SimulateFight and PersistManager
FR7 (Startup of software in browser)	Login and PersistManager
FR8 (Game display in browser)	FriendsList, TradeMonsters, Breed, SimulateFight and PersistManager
FR9 (Friend matching)	FriendsList, PersistManager
FR10 (Fight notifications)	SimulateFight and PersistManager
FR11 (Friends rich list)	FriendsList

5 Dependency Description

5.1 Outline

The following section of the document will explain how the parts of the system come together and how they are dependent on each other.All parts of the system will be dependent heavily on the database, as all user details will be stored in it, such as their login details, monsters, cash and a list of friends.

5.2 User

In order for the user to play the game and access its internal components, it is a requirement to be logged in. User will not be able to create a monster, add friends or sell/buy monsters if login procedure was not successful, if this is the case, an error would be returned and a user would be asked to register his username and password to the database, which would then allow access to the game itself.

5.3 Login/Register

Login and register methods will be entirely dependent on the database classes, whose task is to ensure that the details are entered and retrieved correctly allowing user authentication. TradeMonsters is dependent on the Shop class which provides essential methods for the procedure to complete, also supplying a list of monsters, which needs to be populated either with local or external data.

5.4 Trading/Shop

Buying a monster will not work when the users cash is below the selling price. Breed class is dependent on two monsters, when less than two monsters are supplied, the class will return an error and would not proceed further. SimulateFight class also needs two monsters as a minimal requirement for it to work, otherwise producing an error and not proceeding further.

5.5 Database

PersistManager is a class that will communicate with the database itself, from which it is going to retrieve the data. The Persist Manager is dependent on the user and monster classes.

6 Interface Description

6.1 Model Package

The Model package contains the classes that deal with data representation. It will also contain all connections to the database, and methods of using all of the data types we may need to use in the controller class later.

6.1.1 PersistManager Class

Public class that handles transferring the data currently active in the application to persistent data in a database. It also handles retrieving the data from the database. There are 6 public functions, that are listed below. It uses the Monster class and it works by translating a Monster instance into a set of database friendly fields. It is used by a lot of the classes in the Controller package, so that they can retain information and process it later.

init Function A public function that when called with initialize the class and instantiates the Persistence class to deal with the database. It takes no arguments and returns nothing.

create Function A public function that will create all of the fields in the database for a Monster instance. Takes a Monster instance as an argument.

update Function A public function to update an entry for a monster or a user in the database, takes a Monster instance or a User instance and returns nothing.

search (monster) Function A public function for searching the database to find a list of a user's Monster instances. Takes a User instance and returns an array of Monster instances.

search (user) Function A public function for finding a user in the database takes a nothing and returns a User instance and returns nothing.

delete(monster) Function A public function to delete the representations of the Monster instances from the database. Takes a Monster instance and returns nothing.

delete (user) Function A public function to delete the representations of the User instances from the database. Takes a User instance and returns nothing.

isReal Function A public function to check if the user is stored in the database. Takes two Strings a username and a password. Returns a boolean.

shutdown Function A public function for closing a connection to the database. This function requires nothing and returns nothing.

6.1.2 User Class

A public class to represent a user, it does not inherit from any classes. There are no public variables because everything is dealt with by the PersistManager class.

ChangeBattleMonster Function A public function to change which user's Monster is set for use in a battle situation. Takes a Monster instance to set as the users new battle Monster.

6.1.3 UserFactory Class

A public class designed to create new User instances when called from somewhere. It has one public function, to create the User.

makeIt Function This public function will create a User instance, that will later be stored by the PersistManager. Likely called by the Login Class from the Controller package. It requires the following arguments:

- name: String - The name of the new user.
- password: String - The password of the new user.
- vMoney: int - The amount of money the new user has.
- type: Type - The user's type, ie. regular user or superuser.

This function will return a User instance.

6.1.4 Monster Class

The Monster class only has public getters and setters, it represents each monster and contains everything that is necessary to represent one. The PersistManager uses the Monster class when it converts each monster into a database friendly form.

6.1.5 MonsterFactory Class

A public class to create Monster instances that will later be stored by the PersistManager as part of a User instance. It has one public function.

makeIt Function This public function will create a Monster instance. It may be called from the UserFactory class when a new user is created to make their first Monster. It might also be called from anywhere else that needs to create a new Monster instance, such as the Breed (servlet) class. It needs to be called with the following arguments:

- name: String - The name of the new monster.
- height: int - The new monster's height.
- age: int - The new monster's age (in days I expect).
- strength: int - The new monster's strength.
- health: int - The new monster's health.
- sex: Boolean - The new monster's sex as an isMale boolean.
- aggression: int - The aggression of the new monster.
- dob: int - The new monster's Date of Birth.
- battleMonster: Boolean - States if the new monster is the monster used for battle.

This function returns a Monster instance.

6.2 Controller Package

The Controller package contains all the classes that deal with the logic behind the back end of the application. It contains all the servlets and uses the Model to represent data for it.

6.2.1 CreateMonsterAndUser Class

A public class used by the Login (servlet) Class to handle creating a user and the user's first monster monster. It will rely on the MonsterFactory and UserFactory in the Model. There are two public functions.

CreateMonsterAndUser (Constructor) Function This function will set up the class, by creating a User and with it, it's first Monster. It takes no arguments and returns nothing.

CreateMonster Function A public function to create a Monster instance, it takes no arguments and returns a Monster instance. Presumably used by the constructor.

6.2.2 Login (Servlet) Class

The public class login handles the authentication of the users it is a servlet class and is used over http. There are a number of public functions to deal with users.

checkPass Function A public function for checking a password is correct. Takes a String and returns a Boolean value that reflects whether the password is correct.

checkUserName Function A public function for checking if a user name is correct. It takes a string and returns a boolean value that reflects if the user name is correct.

register Function A public function to register a new user into the system. It needs to be passed two variables; username and password both as strings. it returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.3 Shop (Servlet) Class

This servlet class handles all of the information required to deal with the shop, it allows users to buy and sell Monsters and to find out prices for other listed items.

calcWorth Function A public function that calculates how much a monster is worth so that it can be displayed in the shop. Uses a local variable of Monsters in the shop to find the monsters, and returns an integer that represents the price.

buy Function A public function that allows the user to buy the selected monster from the shop if they can afford it. It takes no arguments and returns nothing.

sell Function Public function that allows the user to sell the selected monster from the user's monster list for the calculated worth of the monster. It takes no arguments and returns nothing.

viewVMoney Function A public function for finding out how much money the user has, because this is needed for display in the shop. There are no arguments and nothing is returned.

addMonster Function A public function that adds a monster to the shop, it takes no arguments and returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.4 FriendsList (Servlet) Class

A public class to deal with the Friends List and all the information required, it also allows users to add, view, and remove from their friends list.

addFriend Function Public function that allows the user to send a friend request and if accepted friend ID will be added to friends array.

removeFriend Function Public function that asks the user to confirm the removal of the specified friend ID, if confirmed, the specified friend ID will be removed from the friend array.

requestFight Function Public function that allows the user to pick a friend from the friend list and to start a battle against the friend's preset battle monster.

getFriend Function A public function that allows the user to pull out a friends profile. Takes no arguments and returns nothing.

findUserMonsters Function A public function to find all of another user's monsters. This function takes no arguments and returns nothing.

updateList Function A public function to update a user's friend list from the server. Takes no arguments and returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.5 TradeMonsters (Servlet) Class

A public (servlet) class to handle monsters being traded.

TradeMonster Function A public constructor function that takes two Monster class instances as an argument, it then swaps them over and returns nothing.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.6 Breed (Servlet) Class

A public (servlet) class to handle how you might breed monsters in the game. It allows you to perform all relevant checks to the data. There are a few public functions made available. There are no public variables. The only important things are the two parent Monsters, they are passed in as parameters to the createChild function.

isSameSex Function This is a public function to work out if two given Monsters are of the same sex and therefore cannot breed. It takes two Monster instances and returns a boolean that reflects if they are of the same sex or not.

calcCost Function This is a public function to work out what it would cost to buy a particular monster, takes a Monster instance and returns an int that is the price to buy the given monster.

createChild Function A public function that allows the user to breed a monster, takes two Monster instances with different sexes and returns a Monster instance that is a child of the two given monsters.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

6.2.7 SimulateFight (Servlet) Class

A public (servlet) class that represents two monsters fighting in the game. There is only one public function available and no public variables.

calcWinner Function A public function to work out which Monster is going to come off better in a fight. Takes two Monster instances, returns the winning Monster instance.

isPrized Function A public function to work out if a match is prized, takes nothing as an argument, but returns a boolean.

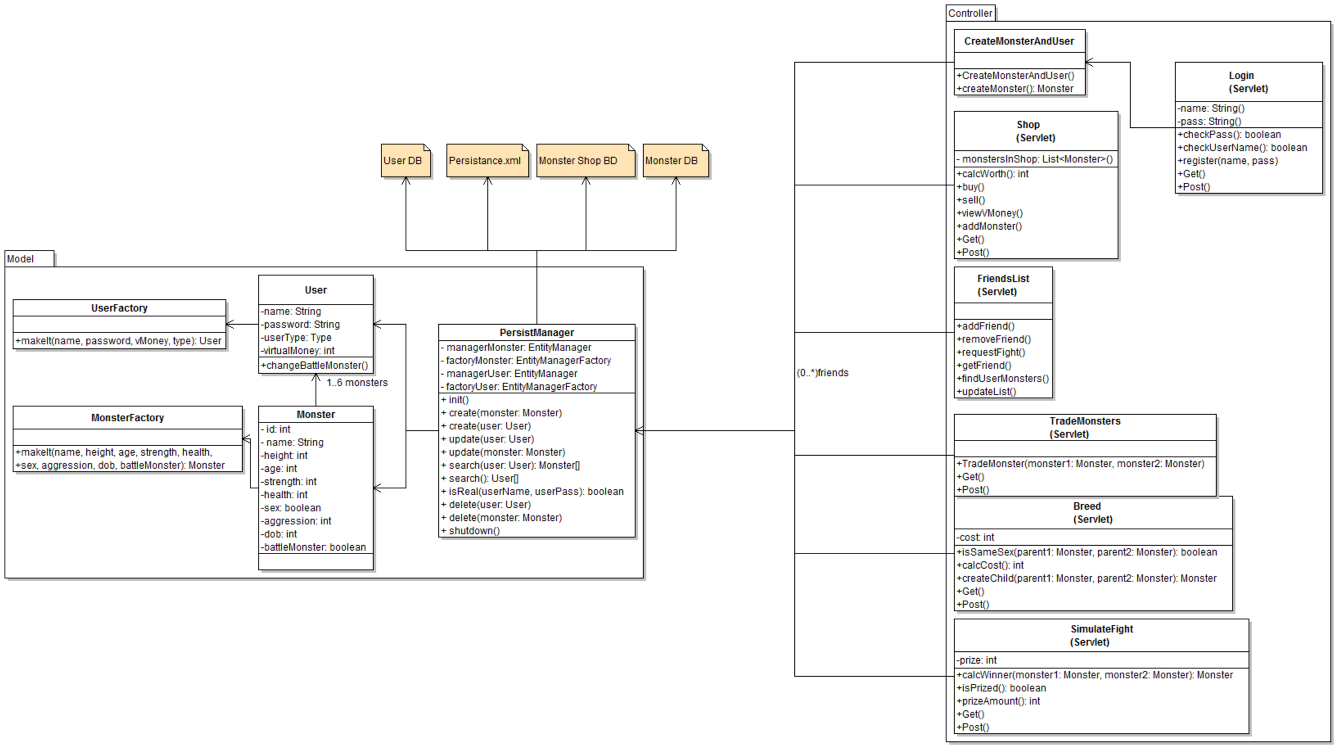
prizeAmount Function A public function to return the amount that is given as a prize, takes nothing as an argument and returns an int that represents the prize money.

Get Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

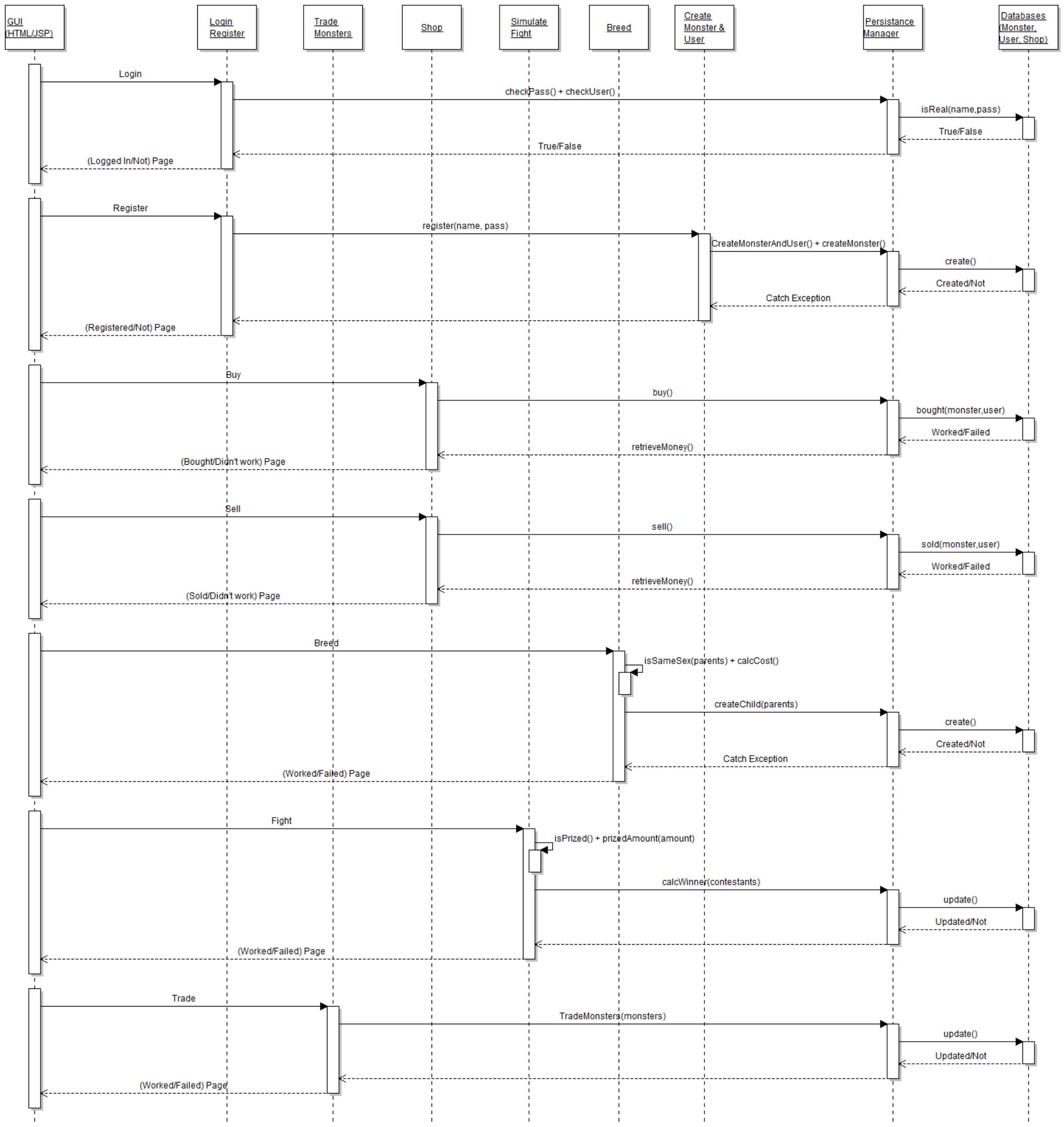
Post Function A public function so that this class can be used by Glassfish to deal with web requests. Takes no arguments, returns nothing.

7 Detailed Design

7.1 Monster Mash Class Diagram



7.2 Monster Mash Sequence Diagram



8 References

1] Software Engineering Group Project Plan Design Specification Standards. C. J. Price, N. W. Hardy. SE.QA.05a. 1.6

9 Document History

Table 1: Document History

Version	CCF No.	Date	Changes made to document	Changed by:
1.0	N/A	06/12/2012	Created Design Specification	sam39