**CS22120 Software Development Life Cycle**

**Group 05 Final Report**

**Author:** Chris Savill – chs17

**Config Ref:** SE.05.DS Group 5

**Date:** 28th January 2013

**Version:** 1.0

**Status:** Draft

Group 05

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2012

Table of Contents

**Project Plan:**

**Test Specification:**

**Design Specification:**

**End-Of-Project Report:**

[Management Summary 8](#_Toc347222418)

[Historical Account of Project 8](#_Toc347222419)

[Performance of Team 8](#_Toc347222420)

[Critical Evaluation of Team and Project 8](#_Toc347222421)

**Appendices:**

[Requirements 9](#_Toc347222422)

[Test Report 10](#_Toc347222423)

[Test table 10](#_Toc347222424)

[Failed tests 11](#_Toc347222425)

[Maintenance Manual 11](#_Toc347222426)

[Program description 11](#_Toc347222427)

[Program structure 11](#_Toc347222428)

[Algorithms 11](#_Toc347222429)

[Main data areas 11](#_Toc347222430)

[Files 11](#_Toc347222431)

[Interfaces 11](#_Toc347222432)

[Suggestions for improvements 11](#_Toc347222433)

[Things to watch when making changes 11](#_Toc347222434)

[Physical limitations of program 11](#_Toc347222435)

[Rebuilding and testing 11](#_Toc347222436)

[Personal Reflective Reports 12](#_Toc347222437)

[Chris Savill – chs17 12](#_Toc347222438)

[Richard Gray – rig6 13](#_Toc347222439)

[Edward Davies – edd14 14](#_Toc347222440)

[Sam Morrison – sjm16 15](#_Toc347222441)

[Jacob Smith – jas32 16](#_Toc347222442)

[Ivan Cholakov – ivc 17](#_Toc347222443)

[Katherine Rose Farmer – krf 18](#_Toc347222444)

[Oliver Roe – olr1 19](#_Toc347222445)

Project Plan

Test Specification

Design Specification

End-Of-Project Report

# Management Summary

# Historical Account of Project

# Performance of Team

# Critical Evaluation of Team and Project

Appendices

# Requirements

**Functional Requirements**

**FR1 Server-based authentication**

The server will be used to authenticate a user, allowing them to log-in or register from their browser.

**FR2 Server friends list**

The server will maintain a list of friends for each user. Users will only be able to interact directly with their friends. Friends will be identified by their email address and added by a request-confirm mechanism.

**FR3 Server monster list**

The server will maintain a list of the monsters owned by each player and their attributes. These include genetic attributes and phenotypic attributes (such as age, health etc). The server will manage the monster lifecycle i.e. mating, birth, ageing, illness, injury and death. New users should be allocated a basic (random) monster and a small pot of virtual money.

**FR4 Server monster fights**

The server will handle monster fights with a (virtual) cash prize available. The system will provide a fixed value prize to the winner. Users can select one of their monsters and challenge one of their friend’s monsters to a match. The friend can accept or decline the challenge. If they accept, the server will decide the winner based on the characteristics of the monsters along with an element of random chance (see Appendix A for an outline suggested algorithm). The server “pays” the winner the prize value and the loser’s monster should die.

**FR5 Server-server communication**

The server should be able to communicate with other servers using a standard protocol (agreed between groups) in order to play the game (add friends, buy/sell monsters, arrange monster breeding, manage fights, etc).

**FR6 Client options**

The client will allow users to interact with the system i.e. register/unregister, add/remove friends, offer for sale/buy monsters, offer for breeding / purchase breeding, etc. The sale and breeding of monsters will be managed in a similar way. If a user wishes to offer a monster for sale or breeding they can assign a value to the monster.

Any of their friends can view the monster’s price and purchase it or hire it for breeding. When purchasing the monster is transferred to the purchaser, when breeding the offspring are transferred to the purchaser. In both cases the sale price is transferred from the buyer to the seller. If the buyer does not have sufficient funds the transaction should not take place.

**FR7 Startup of software in browser**

When the software first starts, it will display a set of choices for the user as follows:

• Log in

• Create new account

Once logged in the system should provide an option to log-out. This will take the user back to the initial log-in/register screen.

**FR8 Game display in browser**

When the player has logged in they should be able to see a list of their monsters (with status info), their friends (with offers of monsters for sale and for breeding), challenge requests (with prize money etc) and have options to interact with these options as described in FR6.

**FR9 Friend matching**

The system should allow users to send a friend request to other users of the system (identified e.g. by their email) and to accept or reject requests sent to them. On accept the friend would be added to the friend list.

**FR10 Fight notifications**

Following a fight that the user has entered, the monster lists off all competitors should be updated. Loser’s monsters should be removed from their list, the winner will have the prize money added to his account and the monster’s status will be updated (accounting for injuries etc).

**FR11 Friends rich list**

A user should be able to see a list of his friends (including himself) and the wealth of each, ordered by wealth.

**External Interface Requirements**

**EIR1 Appearance of Interface**

The program should conform to usual look and feel guidelines for web-based applications.

**Performance Requirements**

**PR1 Response of program to user input**

The user should feel like the system is responding to them at all times during game play. There should not be any perceptible lag between attempting a game action and the system responding.

**PR2 Target computer for system**

The client software produced should run correctly on standard browsers (i.e. one of the browsers installed on the IS desktop). The servers should also run either on the Department’s or University’s systems or a third-party system, but should be accessible from the department for testing.

# Test Report

## Test table

## Failed tests

# Maintenance Manual

## Program description

This is a browser based game about fighting monsters, with the aim to educate people on the evolution of breeding. Each player has their own monster(s) that they can use to fight, breed or sell with other users. Each monster has some generic attributes, these are (don’t know). It accomplishes this by using a web based user interface that uses a supporting server program to store user information and to handle the server to server interaction.

## Program structure

## Algorithms

Reference to design spec

## Main data areas

## Files

## Interfaces

## Suggestions for improvements

As with most software, on reflection there are many features and processes that can be improved, such as:

Attributes – by giving the monsters more meaning full attributes the game will be able to appeal to a wider range of users that enjoy combat games, here is a list of attributes that could be used:

* Strength – the number of attacks is dependent on how much strength the monster has before it is too weak to fight, this will reset after a period of time, maybe 1 point every 3 minutes.
* Attack – this is the number of point inflicted on the other monster when an attack is instigated.
* Armour – this is the number of points the monster can use to defended is self against an attack, the armour does not neutralise the attack points but is does reduce its effectiveness
* Intelligence – the intelligence attribute will factor in to what type of attacks and how effective those attacks are
* Speed – the intelligence attribute will factor in to what type of attacks and how effective those attacks are
* Type –the ‘type’ attribute will factor in to what type of attacks and how effective those attacks are, it will also give an advantage or disadvantage if the other monsters ‘type’ is opposite to the users monster e.g. water will win against fire

Shop – through adding a shop to the game it wound enable users to purchase items that can be used to improve their monsters, items such as;

* Health potions – to increase the health of the monsters in order to increase the chances of a successful win.
* Weapons – to increase the attack of the monster in order to increase the chances of a successful win.
* Armour – to increase the armour of the monster in order to increase the chances of a successful win.
* Monsters – to purchase a new monster or additional monster for additional fights or breeding.

Monster evolution – by winning a fight each monster will earn a number of points that can be exchanges for health, weapon or amour points. This will make the monsters more valuable and a better pedigree for breeding.

Combat evolution – when a fight is started the user can input commands that can change the outcome of the fight, such as special attacks that would reduce the monsters strength, these special attack effectiveness is calculated by using the attributes the monsters have, mainly attack, intelligence, speed and type.

Special attacks can be combated by armour but it will not neutralise attacks only reduce them by a parentage which will be calculated using the armour, speed and intelligence

Graphics –

Gender/breeding – by giving the monsters a gender, it would give the game a better way of educating people on how evolution works on a small scale where the monsters can have bread to create a better monster with higher attributes, it also makes the game more enjoyable and give it another way to play, where as some may want to just fight, others may want to breed and make money or even just breed a super monster.

Needs someone more technical to comment on the technical side of stuff

Colour blind people

Mobile friendly

## Things to watch when making changes

Through using a programming language like Java, programmers can write software to do almost anything in many different ways using different variables and methods. One problem that this could cause is confusion between programmers.

If the software were to be written using a standard set of methods and those methods were change or discarded and not all the programmers were told, then the software (where it is vital that they integrate together with very little or no problems) would not work and would have to be ether re written or a new piece of code would have to be written like a bridge to the other codes.

Through using ‘Git’ programmers are able to all work on the software and able to have up to date copies of all the documents, however if the data is not committed, pushed or pulled correctly then problems can occur and it can be difficult to uncover the mistake and sometimes data can be lost and have to be re written.

Correct procedure

## Physical limitations of program

Hardware requirements

Server limitations

## Rebuilding and testing

# Personal Reflective Reports

## Chris Savill – chs17

## Richard Gray – rig6

## Edward Davies – edd14

## Sam Morrison – sjm16

## Jacob Smith – jas32

## Ivan Cholakov – ivc

## Katherine Rose Farmer – krf

## Oliver Roe – olr1