The University of Auckland

compsys 302 group 21

Jack McIvor 5999650

Nayoung (Monica) Jung 6328365

Table of Contents

[Introduction 2](#_Toc453031422)

[Requirements 2](#_Toc453031423)

[Top-level view 2](#_Toc453031424)

[Significant Issues 3](#_Toc453031425)

[Features 4](#_Toc453031426)

[Network Model 5](#_Toc453031427)

[Protocol Discussion 5](#_Toc453031428)

[Tools 6](#_Toc453031429)

[Further Improvements 6](#_Toc453031430)

[Phase 2 Project Marking CheckList 7](#_Toc453031431)

[Figure 1 Top Level View Diagram 2](file:///C:\Users\나영\Downloads\group21.docx#_Toc453031224)

Figure 2 Front Page ……………………………………………………………………………………………….4  
Figure 3 Main Page ………………………………………………………………………………………………..4  
Figure 4 Network Top Level View Diagram …………………………………………………………..5

# Introduction

The first phase of the project was to make a simple arcade style game with the goal of entertaining an 8 year-old and their friends. It was developed using Java and other programs to make modern graphics and a better user interface (UI). We succeeded in making a game that not only true to the original, but also fun and age appropriate.

However, the replayability of the phase one game is questionable. There is only so many times a child can ask his Dad to play computer games with him, and the AI player can be bested with a little patience. So, the second phase of this project is putting the game online, so that the child may play with his friends.

We had to create a web server for an online multiplayer game using Cherrypy. The purpose of this project was to manage communication and networking so the game “just worked” for the end user. To do this, we had to interface with a loginserver, connect to a remote parallel server as a master or slave, and send information to the game across this network so computation or rendering occurred appropriately.

Not only did this involve the use of Python(Cherrypy) and Java, but HTML, CSS and few JavaScript are used to develop the web user interface. Low level web sockets were implemented using Java and Python wrappers for OS system calls.

# Requirements

There are minimum requirements to play the online multiplayer mode. We have completed Login Server Protocol, so we can login to our server and see who is online. We use our UPI as an ID and our id number as the password, then hash the user’s password to log in. The password is marked to certain other people around the user are not able to see his password. After logging in, it redirects to the main page and you are able to see list of people who are online.

Most of the protocols in Application Protocol are done. When you start the java game and press enter, the python script starts to run, followed by the webserver. It is able to send request to other online user and receive challenge request. When the game starts, two users are not getting or sending information through the server, they communicate through the socket. That data goes to java, and java is resending the data to python.

The main webpage automatically refreshes every 10 seconds to get an updated list of online users or data. We used multi-threading, so the game can listen while running which make game update faster. Also the web-server can listen to java while also listening to connections from another web-server. Also, report the user every 30 seconds to make sure the user is logging in to avoid the user automatically logs off also occurs in a threaded manner. When the user is sending request to other user, it used JSON encoding to send data.

# Top-level view

Play Game (Java)

Select “online multi-play” mode

Online

Log in to server

Challenge

Request other online user

Friend List

Figure 1 Top Level View Diagram

Open the Server

Looking at Figure 1, in order to open the webserver, one has to start the java game and select the “online multi-play” mode. When chosen, Java runs the required Python scripts automatically, which in turn opens the web server with the users default web browser (either Firefox or Chrome). To login to the online server, the user requires their University of Auckland UPI as their ID and the University ID number as their password. When they login successfully, the player may request to challenge other online users. If the other user accepts the challenge, the sender becomes the master, and the receiver automatically becomes the slave. The master sends the information to the slave, and in return, receives information from the slave as well. All the information being sent goes through the Java application, which receives and updates the changing position and state of the slave. A friends list was also implemented, along with a section detailer the developers of the game, which are only accessible when the user is in the “online multi play” mode.

The python/ python webserver connection is managed by the popular package Cherrypy, while the python/ java connection uses low level web sockets, bound to the local machine. For robustness, ports are dynamically allocated (by the OS). Both java and python bind a socket server and both application sends a message to the other to let it know what port it is listening on and test the connection.

# Significant Issues

* Time management: Compared to the difficulty and number of minimum requirements, there was not enough time to accomplish all of them. Also, it took a lot longer than expected to specify the protocol. This is probably because we had no experience in web development or networking and so overestimated what we could do.
* Linux Quota: One of the requirements was for the protocol to run on Linux. To prevent unexpected errors while running the protocol later, it was deemed better to use Linux from the beginning of the project in regards to the protocol. However, a shortfall of Linux is that there is a limited quota to the amount of data that can be stored. When the quota is exceeded, it is not possible to store any of our code to push our repository.
* Testing with the other port: It was not hard to display the list of online users. However, the main server constantly wanted to change the order of users on the list, which meant that our servers list was changing as well. E.g. when we send a challenge to user A, it sends a challenge message to user B instead of A. It was hard to reliably test if user A was receiving any information from the sender.
* Testing in general: It was a lot harder to manage testing in this phase of the project due the different applications running. A successful test requires five elements working seamlessly (see Network model overpage) and a simple port error would break the whole system. Thus when we were developing the system we had to be extra sure things were being sent properly and developed optional system arguments for scripts to test. Also, we sent test messages as soon as each element is initialised.
* Slowing down the game: The way we initially implemented the java/python interface was very slow as the programme blocked while connected to web sockets and also polling messages from the incoming queue. To deal with this, we threaded sending messages on both sides (java/ python). The queue is also continually polled and bad messages dropped instantly. Furthermore, java and python implement their server sockets in a separate process (for python) or thread (for java).
* Displaying python stdin, stdout and err streams in java console: Because we run the python script from java, we would like to see what python is printing to these streams in the java console. This required threading over the different streams to read them simultaneously, and also forcing python to flush it’s stdout and err streams in an unbuffered manner.

# Features

Figure 2 Front Page

Figure 3 Main Page

One of the features of this game is a simple web-user-interface, suitable for 8-year-old children. The web page theme is easily recognizable, and it is very easy and intuitive to challenge other users online.

If the “add friend” button is clicked, the specific user is added to the “Friend List”. When you visit the Friends List, it is possible to see the new user that was added, and whether they are online or offline. To prevent the loss of the users in the friends list, a text file was used to store the information on the local drive.

Because the list of online users keeps on changing, it was very easy to obtain the wrong users information instead of the intended user from the list. To avoid this situation, the main page automatically refreshed every 10 seconds, allowing the user to view the most recent version of the online users list.

When the player runs java game, it automatically runs the python script and opens the web-server with the default web driver.

We have good operating P2P networking system. It is needed to prevent when the server goes down because server is not operating. So the slave is sending the correct key pressing data to master and the master is computing game logic and sending the object position to slave well.

Low level socket connections are used to send messages fast and directly between Java and Python. Also, used an ArrayBlockingQueue in java for interthread communication between incoming messages and the game loop. This robust implementation means that it would be possible, with minimal changes, to have the java game communicate with another java game directly.

# Network Model

Figure 4 Network Top Level View Diagram

The diagram above is a brief explanation of the concept we developed. Information is sent and received between Python and Java, and information between users is transmitted through Python and the Web Server.

The user has to login to Andrew’s server which has all the information about user’s account with their Auckland UPI and ID number. After the player logs in, he may request challenge to other user who are online. When the game starts, the information between two users go through in P2P manner. This will need two users Python code to communicate to each other directly through a socket which needs send game rules (key presses, the object positions and the game state). Each webserver/ game pair implements their own protocol to update the game in a master/ slave state.

# Protocol Discussion

In this project, there are 3 types of protocol which has been discussed throughout second half of this semester are login protocol, class protocol and application protocol.

First, the login protocol which had to use Andrew Chen’s server in order to get information about users. It was very straight forward and quite easy to implement. They had few changes until week 12, but it was not a big issue.

With the class protocol, our class choose to implement master-slave system instead of master-master system. The master is the user who sends the challenge request and the slave is the user who receives the request. Then, the slave only sends the key-pressed data to the master, and master is handling the game rule and sends the slave the position of objects.

Last, the application protocol had all the major functions to implement. There was constant changing until end of week13, but the change was needed in order to run programme with the server. It was one of the major issue to communicate Java and Python code.

The java/ python plaintext protocol we used is a bastardised form of URL’s. eg “0?3&4&5”. The first number represents a function call to handle the message, and the remaining integers after this are arguments defined in the order of the inter-app API defined by Andrew. This way messages are kept small, but human readable.

# Tools

In phase 2, Python(Cherrypy), HTML, CSS and Java were used to develop the project.

Python Python was mainly used for the functionality of the server, which sends and receives information between the web server and Java application. It is one of the better tools to use in the development of a web server, as Cherrypy itself has the required fundamental code. This means the developer only needs to concern themselves with the main functions that are required.

HTML & CSS HTML and CSS had a very positive impact on the design of the web page interface. They are well known tools used in the process of designing web pages, and are very intuitive to use.

Java To ultimately play the game and interact between the server and the game, Java was used. There had to be a few modifications and additions to the already existing functions in order to properly communicate with Python.

# Further Improvements

* Add more functions in the webserver e.g. instant message or mini-clip of our java game, so player can play game without accessing java programme to run. Also get improved web design would be considered.
* Use of database can make easy to handle user’s information and more efficient to get needed data. It saves time because it minimises the overlapping data and it gives more accessible information.
* Make sure challenge and respond functions work properly. It is necessary to get done with request challenge and respond to received challenge in order to play the game with other online users.
* Work with Firefox. HTML codes are running fine and do not calls any error when it complies. Everything is working fine with Chrome but some buttons are does not work with Firefox. Get the working buttons for Firefox.