The Trading Group Project

to be completed by 11th June 2004

Second Year Computing Laboratory (CS2)

Department of Computing, Imperial College

Ian Moor <iwm@doc.ic.ac.uk>

Revision History

Revision 4 Last modified: Fri May 14 10:33:40 BST 2004

Purpose

To get experience in web programming 'E-commerce' using both client-side and server-side techniques in conjunction with a database by constructing a multi-user space trading game to be played on the web.

Summary

Each group must implement a web page allowing several players to play concurrently on different browsers on different screens. Each player sees the control panel of his spaceship and travels and contacts other players to negotiate a trade. The information about the game will be stored in a database allowing players to leave the game and return later.

Timetable

1. You should have set up a group of no more than three people and entered the group members' names in the second year lab page [http://www.doc.ic.ac.uk/lab/secondyear/trader] by the start of the exercise (17th May). A shared group directory and database will be allocated to each group.
2. By the start of the second week (24th May) each group must produce a simple working prototype of the game.
3. By the start of the fourth week (7th June) each group must hand in a short report on their project to the SGO.
4. In the fourth week (7th-11th June) groups will give a combined demonstration of their program and a presentation about its design and implementation.

Rules of the Game

These rules should be regarded only as a basis for the project and can be changed or extended.

The game is played among the 30 stars nearest to Earth, we assume each star has one habitable planet. Each player starts on a planet round one of the stars with one space ship. The game involves trading between stars using spaceships to transfer the goods. Players lose if they run out of food and are unable to buy any (a player needs 1 unit of food per turn). The winner is the last player or the richest player when the players agree to end the game.

A turn is complete when all players have made a move: a move is defined as a player having made a transaction or moved (one of) his space ships. To buy or sell something both players involved have to be playing.

Any player can send a message to one or more other players. The messages are stored, and it is up to a player to check to see if there are any messages addressed to him. Messages can be used to make offers and negotiate prices.

A player can move, send or read a message even if nobody else is playing at the time. A space ship uses fuel when it travels but doesn't take time to move (the player doesn't have to wait at the screen while a ship moves). Planets are specialised in producing certain goods, but they can trade in anything they have.

1. Agricultural: the planet grows up to 50 units of food per turn. After a farm tool has been used to produce 100 units of food it breaks down and a replacement has to be bought.
2. Industrial: the planet can make up to 5 farm tools, 5 mining tools, 5 factories, 20 copies of the current Simpsons dvd and 1 spaceship per turn requiring respectively 2, 3,3,3,1 and 10 units of ore for each item. After a factory has made two turns maximum goods it breaks down.
3. Mine: the planet can mine up to 50 units of ore and 100 units of fuel per turn. After producing 300 units of ore or fuel a mining tool breaks down and a replacement has to be bought.

- All planets require 5 units of food and 1 copy of the latest Simpsons dvd per turn.
- A spaceship requires 1 unit of fuel and 1 unit of food per light-year it travels and carries a maximum of 50 units of goods and 50 units of fuel.
- Apart from fuel and dvds, goods can only be transferred between a planet and a spaceship rather than between two ships meeting in space.
- A planet must be occupied (owned to a player) before anybody can trade with it, an unoccupied planet can be colonised by a player with enough resources.
- When a player starts, his planet starts with 10 units of food and 2 tools which it needs to produce its exports. Prices of goods are not fixed, it is up to the buyer and seller to make offers and negotiate the price.

The Database

Which database

The defined database called 'trader' described below is a starting point, its structure allows players can play in more than one game at a time. Each group will be allocated a database in which you make a copy of the tables which you will own and can change, or define your own tables.

The Game table

This table contains the general state of a game. There will be one row for each player in each game he has joined, with the columns:

- `Game`: the number of the game.
- `PlayerNum`: the number of a player in the game.
- `Turn`: the number of turns the player has been in the game.
- `Playing`: is the player active at the moment (type boolean).

The Players table

There is a row for each player. Columns are:

- `PlayerNum`: identifies a player.
- `PlayerName`: what the player wants to be called (type text).
- `Playing`: is the player logged on (type boolean).

Goods

This is information about things can be traded such as money,fuel or ore.

- `Name` : of the item (text).
- `Cargo` : number of the item.

- `Size` : the size of the item.

The Universe table

This is game independent information about the stars.

- `Star`: the name of a star (type text).
- `StarNum` : a unique number for each star.
- `x`,`y`,`z` : the coordinates of the star in light years (type float8), the earth is at 0,0,0 .

StarState

Contains the game related information about a star:

- `StarNum`: unique star number.
- `Game`: unique game number.
- `Owner`: the number of the player owning the planet around the star.
- `Civilisation` : what kind of world it is: agricultural, industrial or mine (type text).

Warehouse

Information about goods stored on a planet

- `StarNum` : star number.
- `Game` : game number.
- `Owner`: the number of the player owning the planet.
- `Item`: a number identifying a line of the Goods table .
- `Count`: the number of the item in the warehouse.

The Ships Table

- `ShipNumber`: the number of the ship.
- `Name`: the name the ship (type text).

Ship State

- `ShipNumber`: the number of the ship.
- `Game`: the number of the game.
- `Owner`: the number of the player who owns the ship in the game.
- `x`,`y`,`z` the coordinates in light years of the ship (type float8).
- `Capacity`: the size of the ship's hold. Ships have a default capacity of 100 units of goods (including fuel).

The Hold

Info about the goods in each ship's hold.

- `ShipNumber`: the number of the ship.
- `Game`: the number of the game.
- `Cargo`: number of the item in goods.
- `Count` : how much of the item is in the hold.

The Messages Table

The messages and mailbox tables are used for communicating with other players: Messages are not removed

from the database.

- `MsgNum`: an integer identifies the message.
- `Sender`: the `PlayerNum` who sent the message.
- `Game`: the number of the game
- `Sent`: the current time when the message was sent (type datetime).
- `Subject`: (type text).
- `Message` : the text.

Mailbox

- `MsgNum`: the number of the message
- `Game`: the number of the game
- `Dest`: (type integer) the player the message is addressed to (player 0 is everybody).

Database use

- SQL ignores the case of column names, but comparison of values in text columns is case sensitive.
- Communication between your program and the database should be done by using the database interface appropriate for your server-side code, for example php defines functions for operations on different database systems, perl uses the DBIinterface, JSP uses the jdbc interface, NET.ASP supports `ODBC`.
- You should assume that players are only able to access the database using the user interface of your game.
- Postgresql has a programmers interface for performing things which are difficult to write in SQL. It is possible to write procedures in the imperative language PL/pgSQL.
- You can add rules to a database to control access to the data. For example this rule restricts changes to amount of fuel in a ship by checking the new value that an update statement would store:

```
CREATE RULE only50 AS ON UPDATE  TO ships
WHERE new.fuel > 50 OR new.fuel < 0 DO INSTEAD NOTHING;
```

  (**NOTHING** is the do-nothing SQL statement). Alternatively the game code can check the value before updating the database.
- You can add triggers to a database which are functions called automatically when specified conditions are matched. For example the PL/pgSQL procedure `check_positions` is called each time a row of the ShipState is updated to prevent a collision.

```
CREATE TRIGGER no_crash BEFORE UPDATE ON ShipsState FOR EACH ROW
EXECUTE PROCEDURE check_positions();
```

- The commands **LISTEN** and **NOTIFY** can be used for asynchronous communication between processes using the same database. They are usually used from PL/pgSQL procedures.
- Different players may access the database at the same time. You have to make sure that the result is correct.

  1. For example: One player uses a **select** to get the position of another player who using an **update** to move at the same time. Postgresql makes sure that the statements are not done currently, one statement is completed before the other starts.
  2. However if a value is read from a table and updated later two different processes may overlap for example with the sequence

```
player 1: SELECT turn FROM Game WHERE PlayerNum=1 INTO TABLE a;
player 2: SELECT turn FROM Game WHERE PlayerNum=1  INTO TABLE b;
player 1: UPDATE Game SET turn=1 + a.turn WHERE PlayerNum=1 ;
player 2: UPDATE Game SET turn=1 + b.turn WHERE PlayerNum=1 ;
```

     the value of turn is only incremented by 1. To prevent this in SQL the two statements should be put in a single transaction. for example:

```
BEGIN TRANSACTION;
```

```
SELECT turn FROM Game INTO TABLE a FOR UPDATE ;
/* stored in a and then */
UPDATE Game SET turn = 1+a.turn ;
END TRANSACTION;
```

during the transaction no other updates can occur (note the **FOR UPDATE** option on select which locks the row).

3. The **lock table** command can prevent other uses accessing or updating a table while the transaction is running. Think carefully about locking of database tables when users are involved for example:

```
player 1 gets information about food on a planet and goes to lunch.
player 2 who owns the planet sells the food to a third player
player 1 returns, tries to buy the food and finds it has gone.
```

The information player 1 has accessed is changed before he uses it, but locking the table (or even the relevant row) prevents player 2 from selling the food for a long time. Instead use a conditional update

```
UPDATE warehouse SET count = count - 5
  WHERE count > 5 AND Item = 'food' AND Owner = 1
```

The Group database

A database will be allocated for each group, use the same database password that was provided earlier for each user. To create a version of the trader database, first get the definition and initial data for the 'trader' database by typing **exercise trader**. The file `trader.sql` contains definitions of the tables and commands to load data into the tables and set initial permission. Read the file to see what changes you want to make on the file and edit it. Start **psql** specifying your database name and and from the **psql** prompt type **\i trader.sql** to load the table definitions and initial data into the database.

The database server is backed up, but you can also use the **pg_dump** command to save the database before a major change.

```
pg_dump -u --host=db --inserts ourdatabase --file saved.sql
```

writes the commands and data to recreate 'ourdatabase' to the file `saved.sql` so it can be restored later.

Changing the definition of your database

The SQL standard does not include any commands to change the definition of an existing table in a database. Postgresql has an extra command **ALTER TABLE**, to add a column to a table or remove or rename a column in a table. Changing the type of a column can only be done by deleting or renaming the column and then recreating it with a different type.

Implementation

The Development Web server

Access your group directory via the web with the url:

```
http://www.doc.ic.ac.uk/project/2003/261/your_group_name
```

A group can use any of the server-size languages CGI, PHP, JSP or NET.ASP and also use java applets, javascript, SVG or Flash on the client-side to improve the interaction and user interface of the game.

CGI

CGI is the protocol communicating between a browser and a server. CGI programs can be run in any program-

ming language that the web server can execute. Perl is often used to write cgi programs, because it has many useful modules such as the `CGI` module to implement the CGI protocol and the `DBI` module to access databases. The web servers in the department will execute a file with suffix '.cgi' from a user's `public_html` directory. when it is referenced (usually by a link from an html file). If the file is only accessible by the owner of the file (type 'chmod 700 my.cgi'), the program is run under the owners account, and will be able to read or write the owner's non-public files and databases. If a cgi program is publicly readable the web server will run it under another account (usually called nobody,or wwwnot) and the program can only use public files and databases. For this project cgi files in the group directory should be readable and executable only by other members of the group so that the result from

```
ls -l our.cgi
```

should be something like

```
-rwx-r-x--- 1 me g0326200 4096 May 20 18:30 our.cgi
```

If group following your username is 'cs2' use the command

```
 chgrp g0326200 our.cgi
```

with your group name in the command to change the group of the file to your project group.

PHP

A PHP web application is written by embedding code in an html file. Like perl there are a large number of useful free libraries available. The departmental web server will only run a Php file as a cgi program, the file must start with the header '#!/usr/bin/php' and have the suffix '.cgi'.

JSP

JSP(Java Server Pages) is java embedded in an html page, similar to a php, but instead of inrepreting the embedded code, the whole page is converted to java, compiled and run by the tomcatserver. It is also possible to build a 'servlet' which is written in java using the 'javax.servlet' package and is run in the tomcat server as a 'permanent' program. To use jsp or a servlet go to /csg's [http://www-jsp.doc.ic.ac.uk]'s page. Jsp pages and servlets use the jdbc database interface described by Sun [http://java.sun.com/docs/books/tutorial/jdbc/basics/].

NET.ASP

NET.ASP supports embedding of JavaScript,C#,VBasic or other NET languages in html pages. Groups wanting to use NET.ASP should mail help asking for access to the server for this project.

Client-side programming

The game must have some server-side component to manage the database, however you can use client-side code to improve performance of graphics and verify the input before it is forwarded to the server.

Javascript

Javascript is an untyped scripting language, the functions are written inside the header of an html page and called by events like a button press in a form (using for example onSubmit) or mouse movement (for example onMouseOver). Read the tutorial at pageresource [http://www.pageresource.com/jscript/jbasics.htm]. Unfortunately, the javascript interpreters on different browsers may behave differently.

Applet

An applet can move computation and graphics to the client, but depends on the browser having a java plugin and

any other required library. Communication between the server and the applet can be done with the applet using the java.net package, send HTTP requests to the server.

Flash

Flash is a commercial program which generates client-side animated and graphic code in SWF. Flash is not available in the departments teaching machines, but plugins for flash version 6 are installed for mozilla and internet explorer, so flash files can be played on a department browser. Look at this flash tutorial [http://www.w3schools.com/flash/default.asp]. A number of free programs for processing SWF written in various languages can be found on the web.

SVG

SVG Scaleable Vector Graphics is an XML based animated graphics format which can be used on the client-side, for example written from javascript or included in a page. The department's browsers support SVG using a plugin. Simple SVG graphics can be generated using xfig or a text editor. For example an introduction to SVG [http://pike.port5.com/prog_svg_intro.html]

Error messages

Detailed error messages from server-side programs are not written on the browser's screen, because the user is unlikely to be the programmer. Instead the errors are recorded in the web server's log. It is a good idea first to run a cgi program from the terminal to see if there are any syntax or other errors before using it from the web. For example a perl cgi can be run with the warning option and initial values for some variables by:

```
perl -w trader_version0.cgi player=me turn=1
```

a php cgi can be tested using the

```
php trader_version0.php
```

Error messages written to the standard error stream are saved in the error log of the web server, output to written stdout may appear on the screen in the page. The department web server writes error logs to the directory `/vol/wwwlogs`, a new log file is created overnight with a name generated from the date.

```
ls  /vol/wwwlogs/error_log.*
```

lists the error log file names, the last file in the list will be the current error log and will contain the error produced by your cgi, as well as all other error messages produced in the last several hours. If the cgi has just be run it is enough to look at the end of the log file using the linux command **tail** to look at the end of the long file, for example to read the last 100 lines of the log on the screen write:

```
tail -100 /vol/www2logs/error_log.1053129600
```

A perl cgi should contain the

```
use CGI::Carp;
```

statement so perl errors are formatted in the standard format.

Management

One of the first things to do is to organize the group. You are not assessed on how the group is organised but a well organised group will be able to do more and better. Keeping a log of what you do will provide information for your report.

Use **cvs** the "Concurrent Versions System" [http://www.cvshome.org/docs/manual/cvs.html], which helps mul-

tiple developers working on the same files. It also keeps a record of file changes, so an earlier version can be recovered if a change has caused a bug.

Working away from the department

1. If you don't have the software you need take care not to spend too much time on installing and debugging software such as a web server or database.
2. You are responsible for making regular backup of your work.
3. Make sure that any software you use is available in the department for the demonstration before you start using it.
4. If you use another database such as mysql avoid any non-standard sql so your program can be converted to using Postgresql.
5. Test that your program on the department's server

Assessment

The Milestone (10%)

Show that you have chosen an implementation language, and constructed a prototype that can select and update an item stored in the database, and displaying the result on the web page. The page must be demonstrated to one of the lab helpers by the beginning of the second week.

The report (30%)

The report should be between 5 and 10 pages and should be submitted to the student general office by 7th June. The report need not be produced using a text processor, but must be readable. A useful gui text formatter on linux is LyX (type lyx and read the online documentation using the Help menu item). Your report should cover:

- Your choice of the implementation language (CGI,php,jsp ...)
- Describe the design and structure of the program and the user interface. Provide an overview of the implementation.
- The conclusion should include what have you learned and what you might have done differently.

The combined demonstration and presentation (60%)

The presentation should be no more than 20 minutes in room 311 at the scheduled time, all members of your group should be present. There will be two dual-boot computers using the standard lab setup in 311, with projectors attached to both.

The presentation should cover the same areas as the report but demonstrate your game being played by two people, showing the user interface and features. Your project must work on at least one of the two supported browsers Mozilla or Internet Explorer .

The presentation need not be done using any tools as long as it can be read and followed by the audience. Unix users might want to try out MagicPoint for their presentation, type:

```
mgp /usr/share/doc/packages/magicpoint-1.09a/sample/sample.mgp
```

and then look at the sample file. Alternatively LyX can be used to generate pdf or postscript slides using the foils package. Windows users can use Power Point if they wish.

Preparation

Rehearse the presentation/demonstration to be check your timing and examples.

Come to the earlier presentations to see what your friends have done.

Documents

- An overview of JSP [http://www.apl.jhu.edu/~hall/java/Servlet-Tutorial/],
- An introduction to jdbc [http://java.sun.com/docs/books/tutorial/jdbc/basics/].
- `/usr/share/doc/packages/postgresql-7.3.4` contains local Postgresql online documents in html on the linux lab machines.
- Using CVS [http://www.cvshome.org/docs/manual/cvs.html].
- A member of the Elite ring [http://homepages.ihug.co.nz/~healer8/kelpie/] Elite is 'the' space game.
- The DBI manual [http://www.doc.ic.ac.uk/lab/labman/lookup-man.cgi?dbi] dbi is the perl database interface.
- The Php tutorial and manual [http://www.php.net/].
- The perl/CGI manual [http://www.doc.ic.ac.uk/lab/labman/lookup-man.cgi?cgi].
- Micorosoft's NET.ASP page [http://msdn.microsoft.com/asp.net]
- A tutorial on using flash [http://www.w3schools.com/flash/default.asp].
- Check the technical library [http://library.doc.ic.ac.uk/index.html] for more detailed books on relevant topics.
- an introduction to SVG [http://pike.port5.com/prog_svg_intro.html]
- Frequently asked questions and examples [http://www.doc.ic.ac.uk/~mwj99/traderfaq.html] related to the project, with examples of using a database from CGI, php and jsp. At the moment this is out of date.