CIS 550 Project: MONOPOLY OLYMPICS

Group Member: Chen Chen, Yayang Tian, Cong Liao, Wenbin Zhao

Project Name: Monopoly Olympics

Platform: Google App Engine

Techniques: SQL, XML, Java, Servlet, JavaScript, jQuery, Ajax, HTML, CSS, Photoshop

Website: monopolyolymics.appspot.com

Blog: http://monopolyolympics.blogspot.com/

Source control: Bitbucket

Abstract:

This application is inspired by the classic computer game: Monopoly (originally developed by SoftStar Entertainment Inc.). Players will play in the map where they can buy their land, build their houses, and charge fees to other players who stop at the owner's houses. The goal of the game is to let other players be bankrupted and collect all the wealth, and it why it is called Monopoly.

Our goal is to make an interesting web application, and online game is absolutely a good choice. Besides, this is an excellent way to attract people to know more about Olympics and learn the knowledge while playing the game. So we decide to develop an Olympic version of Monopoly. The currency in the game is the gold medals. Each time the player wants to purchase a house, he/she must use an athlete. The cost of buying the house is the number of gold medals the athlete has won in Olympic games, and the owner can charge the same amount of gold medals every time another player stops at his/her house. Besides, the owner can charge double amount of gold medals if the house is located on the same continent of the athlete. For example, Michael Phelps has won 18 gold medals, so it cost 18 gold medals to purchase a house using Michael Phelps. If the house is located in North America, it can charge other players 18*2=36 gold medals; otherwise it can only charge 18 gold medals in other continent.

Besides, players can upgrade their houses when they arrive at their houses again. They will need to use another athlete and the "money" to upgrade, and after that they can charge more fees to other players.

A random dice controls the steps. Each player is given a certain amount of athlete cards and

gold medals at the beginning of the game.

A message box will inform the player the game information and also tell them whether it is their turn to play.

Modules & Architecture:

Our program consists of four parts: user interface, front-end codes, server-end codes and data management.

Data management will import XML data and other data into a relational database, select our desired data and export the data as XML files.

The server-end is responsible for starting the game, maintaining game data, and updating latest information to the front-end.

Front-end will send requests to the server and collect the responses and call the correct UI APIs for display. Ajax is used to change the web page content without reloading the page. Besides, the front-end also needs to prevent illegal user behaviors.

UI is responsible for webpage layout and content design, necessary image elements using Photoshop, and animations achieved by jQuery functions.

Data Instance Use:

We use XML intensively in this project. XML is used to import data on the server, and to exchange data between the front-end and server by Ajax. Relational database is used to make and filter the data. Because after filtering, there are only about 500 tuples left, we directly use XML to import them to the server, and relational database is not directly connected to the server.

The data we use are the athletes' names, numbers of gold medals an athlete has won in the olympics history totally, the sport in which they participated, and their nationalities. Besides the provided dataset, we also collect the information from the Internet about which continent a country belongs to.

The number of gold medals is used as the currency in the game. The continent information is used as a way to charge extra fees to the opponents. The sport and nationalities are currently displayed to help the players know more about Olympics. One extra feature that may be done in the future is to add Olympic knowledge competitions in the game as a way to earn gold medals.

Data Cleaning & Import Algorithm:

Since the game is primarily based on the olympics dataset that has been provided, all the relevant information has to be extracted from the XML files and stored in the database afterwards according to the schema design. However, the original dataset contains too much irrelevant information, e.g., age, city, rank, etc. that need to be filtered out. Therefore a data parser is applied to the dataset on the purpose of data cleaning.

The data parser is implemented using Java SAX Parser. It has provided three methods for parsing an element tag in the XML file, i.e., startElement(), endElement() and characters(). The startElement() and endElement() notify the beginning and ending of an element with a specified tag name and the characters() can extract the text inside the element. Only the elements with relevant tag names are parsed and the corresponding information inside the elements are extracted according to the schema design.

The extracted information need to be imported into a database. Particularly, we choose MySQL for data storage and management. We use Java Servlet and integrate the queries in the Java program to communicate with the database. Once the connection has been established, we create the necessary tables and insert the corresponding data using CREATE and INSERT statements respectively according to our schema design.

Algorithms & Communication Protocols:

We use Ajax to communicate between front-end and server. Traditionally, if you want to change the content of a web page, you have to reload it. But obviously we cannot use such a way to build a game. And Ajax is a perfect choice for our usage.

Ajax is not a new language. It is more like a principles of writing javascript and server-end codes. Each time the player does some action and trigger some event, the front-end will send an Ajax request to the server. The server will analyze the request, perform the correct calculation and maintenance, and send back a response to the front-end. The front-end then catch this information, and use javascript the dynamically change the web content.

Periodic update is also used to inform the latest news to the front-end. Every one second, the front-end will send an Ajax request to the server to ask for update. For each player, the server will maintain a queue to store the events since last periodic update (like other players have moved, or purchased a house, etc.). Then the server will process this queue, combining the information belonging to the same category (ex: a player may move twice during the period, the the server should send back the final location of that player), and send this information back to the front-end. The front-end then update the web pages according to the updates to show the behaviors of other players.

The information sent by Ajax is stored as XML. So actually the server and front-end are changing XMLs all the time. It is easy to send and easy to parse.

Use Case:

This application is designed as a multi-user online game, so you can play it whenever you want! It is interesting, and is helpful if you want to know more about Olympic Games.

Optimization Techniques Employed:

During the process of data cleaning, we have filtered out large amount of irrelevant information and reduced the size of the dataset significantly based on the schema before importing the data into the database.

To generate an athlete card, we have to select the athlete name, sport, gold number, country and continent by joining the athlete and country table. The common attribute is country name. Therefore, we create an index on the country name and use the larger relation, athlete table as the inner join when performing the select-join operation.

Technical Specifications:

1. Server-end:

Server-end program consists of two parts: the game logic control and Ajax response.

The game logic control and status maintenance is essentially a big Java class that describes the game. Separate classes are created to describe game players, athletes, map, houses, etc. The control class is a singleton class, which means it only has one object. It provides APIs to the Ajax responses to change the game status and let the game process smoothly. And it also generates XML format response as a string and passes it to the Ajax response class to send to the front-end.

The game logic class should maintain the game status for all players. So it maintains a queue containing update events for each player. The front-end will periodically send update requests, and server will send back updates to inform the changes of the game (ex:other player's movement and purchase), and the queue will contain all the events that happen since last periodic update.

The Ajax response part will receive Ajax requests from the front-end, pass the requests to the game controller, get the response from the game controller and send it back to front-end. In other words, this class acts as an interface between the server and the front-end.

To avoid possible errors caused by multi-threads, there is only one function that communicates with the Ajax response. And this function, in Java, is declared as "synchronized", which means that in any time, only one thread can access this function. This function will call other functions to do the job, and will return only after the jobs are done, and then another thread will be able to call this function and process its request.

Some static functions are in charge of constructing and destructing the game. It can ensure singleton pattern and release the memory as needed.

2. Front-end Interface:

Front-end interface consists of 3 parts: art design, website layouts, and jQuery animation.

Art design. The map in the main playing interface, and all the html backgrounds are drawn manually using Photoshop, while most of the basic figures like boxes are drawn using Illustrator and rendered using Photoshop. The key part, five continents map, consists of 5 layers: a world map background, a multi-directional road, signs(including lands for purchase, bombs, stop signs), houses with 3 levels, and 4 figures. The smooth connections between layers were made possible via layer masks, the 4 different colors of houses were created using color channel, and the realistics roads are derived using renderings.

Website layouts. There are three webpages, login.html, waiting.html, and game.html. An icon will be assigned to each player while they are waiting for others players. In the main playing interface, there are several containers: name container, deposit container, cards container and hint container. Each of them is assigned with a unique id and a corresponding class, so that the elements within containers can align orderly. In CSS, by adjusting z-index, each layers can appear by their priority. In addition, the stable positionings of the elements can be achieved by assigning "position: absolute" and shadow and 3D effects are made possible by CSS3 box-shadow, -moz-transition and etc.

jQuery animation. All the animations can be summarized as: "(selector).animate({styles}, callback)". Among these variables, the fatal part of jQuery animation is the "callback function". It ensures that all the scenarios can execute sequentially, otherwise following lines of code will execute before full animations end. In other words, we should ask to build a house after the movements to that location. Furthermore, "{style}" is assigned "+=left" and "+=top" of CSS properties for the difference of the next and the current movement. This is implemented by storing three arrays, delta X movements, delta Y movements, and the positionings of houses. In this way, jQuery animation can be triggered sequentially upon a click of the "roll" button for a random number from 1 to 6. What's more, other animations such as "dice rolling", "going to heaven", and "information sliding effects" are all based on the same technique.

3. Interaction Between Front-end and Server-end
The interaction between front-end and back-end consists of four parts: jQuery event, Ajax request, callback function and workflow control.

The jQuery event is a bunch of jQuery functions which capture user's behaviors and show results. In the register phase, the username will be recorded and passed to the following webpages. In the waiting phase, user will be assigned an unique id to represent himself. If all set, game begins. In the game phase, user behavior can be identified as five types: dice, move, place, money and bankrupt. When the dice is clicked, dice rolling event is triggered and the user will received a random number between 1 and 6. Then the moving event is triggered and the user will move forward into a new grid according to his dice. The grid may be three types: place, money and stop. When entering a place grid, it means the user can buy this place if it is empty, upgrade if it is already his and not at highest level, or pay if some other player has already bought it. When entering a money grid, his money will be changed by +10 or -10. All these events may makes the player bankrupt when his money is no larger than 0. He is kicked off the game and all his places will be reset.

The Ajax request will parse the user's behaviors into XML format in jQuery events and send the Ajax request to the server. It act as an interface between the server and the front-end coupling with Ajax response. Asynchronization of Ajax ensures the high interoperation of the game.

Callback functions is the mechanism for updating the other players' data which guarantees the synchronization and consistency of different users. It is opposite to the jQuery event part - it captures the results from server and show the corresponding other players' operations to the user. It receives the update message from the server which have recorded all players' operations. In the waiting phase, all current players' data are passed to every player which shows how many users are available and their usernames. In the game phase, when the user receives the update message of other users, the status of other users will be changed.

Workflow control is the most important and most sophisticated part in the system. It arranges the sequence of operations of each behavior. It should guarantee all messages from jQuery events (front-end) and callback function (back-end) are acted and displayed in a reasonable order which makes no side-effect. As Ajax is asynchronized, we should buffered all incoming messages and make it into a “synchronized sequence”.

We originally use cookie (jQuery Cookie Plugin) to prevent one person from opening multiple web pages. But it is not easy for debugging and testing, so we commented the corresponding codes.

Highlights:

- Interesting online game. You can play at home with your friends.

- Use Ajax and XML to exchange information and change web page without reloading it.
- Apply animation on the web page.
- Synchronization through periodic updates.
- Develop dynamic web pages by JavaScript and jQuery.
- Multi-thread control at server-end.
- Non-modal message-box to inform the player
- Clickable athlete card to show detailed information

Work Division:

Cong Liao: Database related work, including transferring data from XML to relational database, collecting other necessary data, setting up database and writing SQL queries. Also front-end UI APIs.

Chen Chen: Server-end program. One part is the logic control and data maintenance of the game. The other part is the Ajax response from the server to the front-end.

Yayang Tian: User Interface, including HTML design and implementation, and JavaScript that controls the movement.

Wenbin Zhao: Front-end program. It acts as a connection between server and UI.

Open Source Code Used:

- AjaxRequest
- jQuery Cookie Plugin
- Apprise Message Box