Scripting Team C

Sotonopoly

Chris Bulmer - cb11g09@ecs.soton.ac.uk

Alex Lo - acsl1g09@ecs.soton.ac.uk

Tom Blount - tb12g09@ecs.soton.ac.uk

Andy Cook - ac22g08@ecs.soton.ac.uk

Nicholas Angeli - na8g09@ecs.soton.ac.uk

Rodrigo Escobar - raeb1g08@ecs.soton.ac.uk

January 13, 2012

1 Description of Prototype Functionality

The application is a take on the multiplayer game of Monopoly where the server is based on the Google App Engine (at sotonopoly.appspot.com). This version uses buildings at the University of Southampton as the properties and has a theme of life as a university student in Southampton.

If a player goes straight to the domain without a link passed in the URL, then a game is created for them and they are provided with a link they can use to invite their friends to their game. Once there is at least two players in the game (and at most four), the player who created the game can start it or it is automatically started once the game has four players.

The players can then move around the board using dice rolls and buy properties and then build offices and research labs on them; instead of houses and hotels in the original Monopoly version. Other changes from the original version are stations are now pubs based on or around the University of Southampton Highfield Campus, the taxes are now based on paying for venues in Portswood, the jail has been changed to the library and the chance and community chest cards have been changed.

The server contains a complete game state and contains all of the game logic. The client contains a smaller version of the game state and a small amount of game logic. Clients can connect to a game and communicate with the server using AJAX requests. The server then sends messages to the client through the use of JSON messages in the Google Channels provided by the App Engine. The client keeps its game state updated from the messages sent from the server.

The JavaScript provides an animated interface on the client, displaying only the options which are relevant at any given time. It allows a player to keep track of what other users are doing in the game, which properties they own and their current money levels. It also provides a console with a history of actions.

Once all other players are bankrupt or all other players have left the game, then the remaining player is declared the winner.

2 List of Tools and Techniques Used

2.1 Tools

Firstly, due to the project being required to run on the Google App Engine, we were required to use the Google App Engine and the Google App Engine SDK for Python. The SDK allowed for us to run the application locally throughout the development and allows for convenient viewing of logs and the datastore. Furthermore, it allows for a straightforward deployment of the application to the App Engine.

Secondly, since this was a group project we decided to use a concurrent versioning system. We elected to use the ECS Forge as it provides SVN management and it's hosted within ECS making it straightforward as we all already have access to it and accounts set up.

Finally, we used several browser tools for debugging the client-side JavaScript. Namely, these are Firebug which was used in Firefox, Google Chrome's in-built element inspector and Opera's Dragonfly.

2.2 Techniques

We used the Google Channels which are provided by the App Engine to push messages to the clients. We elected to do this rather than polling the server as it allows for messages to be sent immediately rather than publishing messages or the game state on the server.

3 Relevant Statistics

Only files that we have developed have been included in the statistics below.

Language	Files	Lines of code	Comments	Blank lines
Python	w	x	y	z
JavaScript	w	x	y	z
HTML/CSS	w	x	y	z
Total	w	x	y	z

We have used several JavaScript and Python libraries in our application. The use of jQuery made the JavaScript more efficient, makes AJAX queries simpler, provides straightforward access to the DOM, aids cross-browser compatibility and provides more functions that standard JavaScript. This allowed us to write high level JavaScript without concerning ourselves with how different browsers will behave.

jQuery UI was used to style custom prompts and alerts instead of the normal browser prompts and alerts; as implementing these in JavaScript allows for the JavaScript to remain running whilst they are on screen and easily links in with jQuery. Both jQuery and jQuery came from jquery.com, however it is delivered via Microsoft's servers.

On the server, we used the libraries that are native to the Google App Engine. These were namely the Channels library and the webapp2 framework. The former was used as a straightforward method of pushing messages to the client from the server. The latter was used to create RequestHandlers so the server can handle dynamic URLs and invoke the correct methods in the game.

4 Overview of Design and Implementation

Server-side, we used object-oriented Python to conform with the standards set out by both the Google App engine and the Model-View-Controller architecture. We seperated the various data containing "model" classes (such as the Game and Player states) from the game logic of the "controller" class (GameUpdater).

The server pushes messages to the clients through channels, making use of the framework and API provided by Google. Messages containing what changes have occurred (rather than the entire game state being sent) allow the JavaScript to keep its own game state on the client, to minimize network traffic. The messages are based on a hashmap of message-type-to-function, which allows for uniform handling of all message types.

AJAX calls from client (using post)

Board is an image, not CSS

5 Evaluation