Overview

This document describes the technical implementation of Utopolis, the city building game developed as part of the course Building Serious Games at TU Delft university in 2013/2014. In this document you will also find detailed installation instructions. The current source code, wiki and issue tracker is maintained at github, https://github.com/BSG-TUDelft/Utopolis. The game is open source and anyone is welcome to explore and contribute to the code.

Server

The server is developed in the Java programming language, thus a Java runtime environment (version 1.6+) and development kit is required. It communicates with the client through Jersey, an implementation of JAX-RS: Java API for RESTful Web Services. The server exposes all necessary resources and operations in a RESTful manner. We chose a RESTful (Representational state transfer) architecture because of performance, scalability, simplicity and reliability.

Objects are automatically serialized through the use of EclipseLink MOXy, an implementation of Java Architecture for XML Binding (JAXB). The reason we used MOxy is because we could change the database objects without having to make any adjustments in our 'business tier'. Also, this framework doesn't require annotation that plain JAXB does require. Thus we decreased error margin, code size and development time.

For persistence we used Hibernate ORM (with JPA annotations) to map Java classes to database tables. It also provides us with query and retrieval functions. This way the server is database implementation independent and we would not have to write manual queries and parse the results. The underlying database is MySQL, known for it's excellent integration with Hibernate. However, if the need arises, switching to any other relational database is a matter of changing the hibernate configuration file (hibernate.cfg.xml).

The combination of Jersey and Hibernate makes the current server setup very extensible. Adding new resources, procedures and functionalities should be quite trivial. All the frameworks and components we have utilized are platform independent.

Therefore, the server can be deployed on any system with sufficient hardware specifications. The exact hardware requirements are very much dependent on the number of players participating in the game and the duration of the game.

Finally, as a build and deployment platform, Apache Maven was chosen to pull in and resolve all dependencies. Maven makes sure that the project is properly built. Using this automation helps us, and other developers, with a hassle-free build and deployment process on multiple platforms.

Client

For the game client we chose the web browser. This because our users are likely to be familiar with those and already use them on a daily basis. It also means schools will not have to worry about deployment and installation, because there is no need for any installation whatsoever. The only requirement is a somewhat modern browser which supports WebGL. In the case of updates or patches, deployment is a non-issue. Because of the recent development in browser capabilities, with regards to WebGL in specific we were able to create a visually attractive, dynamic game world. This also means users are able to run the game on smartphones, tablets and any device with support for HTML5 and WebGL.

We've used the THREE.js Javascript game engine to help create the 3D world. This was the most mature 3D engine (and still is, at the time of writing). For the graphical user interface we used jQuery as well as a number of useful plugins. The client periodically polls the server to get updates.
We chose to stay close to the 'iron age' theme of the game by creating a visual style of the GUI that features a lot of wood and metal textures. The layout of the GUIs elements is rather similar to other games, so should feel intuitively to work with (we confirmed this during our user test).

Teacher Client

Utopolis also offers an interface dedicated to the actions that a teacher can perform during the gameplay period. Some of these are: assign or remove citizens from players, as well as to set up a new game, among others.
To keep the consistency with the technologies used in the game client, this interface was implemented using JQuery, JavaScript, HTML5 and CSS3.

The creation of this last module of the project was implemented close to the end, since we needed to know in advance what features and functionalities were needed and also, those that our commissioner requested to facilitate. Unfortunately, we didn't have much time and a slightly bad design decision was made, namely: dynamic content on pages is being injected directly into the html page using string concatenation, which is not a good choice. We were forced by time limitations to take this route. To solve this, the usage of a templating engine like Apache Velocity or Java Server Pages should be considered.

Future development

To further develop the game we propose to create more (dynamic) game content, such as quests that teachers can create and some of the features mentioned in the Game Design Document appendix, such as Crafting, Trading and Population Dynamics.

The Github issues page features an extensive list of nice-to-haves and future work, such as:

- Animated construction phases of the buildings
- Loading screen to provide the user with feedback during loading of the client data
- Improved security to combat cracking
  Move around in the game world using mouse only
- Water and more variation in the terrain
- More animals and perhaps even NPCs
- Player colour using custom shader
- Day and night cycle
- Basic authentication to keep a user logged in during a session
- Normal and environment mapping on buildings
- 'Wonder' buildings that signify the height of a civilization
- Viewing other cities in a province
- Gold as a resource, obtainable through taxes

Recommendations

We are up coming designers and developers that like to contribute to this project to come up with cool ideas, test them and implement them. In this case it is important to keep the user group, their skill set and their interests in mind. Also one should take into consideration the possibilities and limitations of this project running in an educational context.

As for technical recommendations, it is important to test all client code on multiple browsers. Also keep in mind that we have a stateless, session-less server (based on REST principles) and there is no persistent connection between the client and the server. That means that the client can only send requests to the server, if something changes on the server between requests, there is no way for the client to know about it besides polling the server state periodically. Also the server has no way of knowing whether a user closed his or her browser, nevertheless this is not an issue in a game like Utopolis in which no fast-paced real time action is performed.

Another topic, which relates to recommendations and further future work, is that all three modules of this project have no security whatsoever. This was a choice we made at the start of the project with agreement from our Commissioner. It must be mentioned that the architecture and setup of the server is such that adding security, authentication and authorization is facilitated and should be not too difficult to accomplish.

Installation instructions

These instructions describe how to install the server. The client does not require installation, just point the browser to the right url.

The server should run on any platform that runs Java and MySQL, so these instructions are kept as platform-independent as possible. We assume the game's source code (in the original directory structure) is on the hard drive and you are familiar with using a terminal.

1. Install Apache HTTP server + MySQL
To run the server we need a HTTP server. Apache HTTP server is preferred and tested. Install Apache2 through a package management tool (e.g *apt-get*) or download it here:
> http://httpd.apache.org/download.cgi
Start apache and copy the full source code of the game to a web directory (or create a virtual directory).

The database runs on MySQL. Install MySQL server and client through a package management tool (e.g *apt-get*) or download it here:
> http://dev.mysql.com/downloads/
Run MySQL. Note that you have to run MySQL without root password.

Alternatively, you can make life a bit easier on yourself and download a web server solution stack package XAMPP which has both.
> http://www.apachefriends.org/en/xampp.html

2. Install Java JDK
The server runs on Java, so you need to download a Java JDK, at least version 1.5. Download it through a package management tool (e.g. *apt-get*) or download it here:
>
http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html

3. Install Apache Maven
Then Apache Maven has to be installed. This will ensure all dependencies and Java libraries are installed. Install maven through a package management tool (e.g *apt-get*) or download it here:
> http://maven.apache.org/download.cgi

4. Create database schema
The server requires a database schema to be in place before the database can be generated. Create a new database schema using mysql client or phpmyadmin. The name of this schema has to be *"utopolis"*.

5. Compile and run!
Drop to a terminal and go to the directory where "pom.xml" resides (currently, this is nl.tudelft.bsg.utopolis.server). Type

```
$ mvn clean install
```

All required libraries and dependencies will be installed. After that type

```
$ mvn exec:java
```

To compile and start the server.

6. Set correct server path
The last thing that remains is to tell the client where it can find the server. In a text editor, open the file */client/main.js*. On the third line you will see:

```
var host = "http://localhost:8080/api/";
```

Change 'localhost' to the URL of the server that is publicly accessible.

Also, do not forget to change the same variable (host) but on the teacher client side. Go to */teacher-client/js/scriptWrapper.js*, and you can find it on the first line of that file.

7. Client access
Point your browser to *http://[server-address]/[source-directory]/client/index.html* for the game client.
To access the teacher interface, go to
http://[server-address]/[source-directory]/teacher-client/index.html

Game Controls

w - move camera forward
s - move camera backward
a - move camera left
d - move camera right

r - move camera up
f - move camera down

q - rotate camera left
e - rotate camera right

, - zoom in
. - zoom out

[- rotate building CCW (placement mode)
] - rotate building CW (placement mode)

x - remove selected building

l - toggle leader board

m - toggle music