**Source code:**

**DungeonOfDooom-master\Sourcecode\project\assets\maps**

* level1.json

{

"lev":

{

"name":"level1",

"coin\_num":30,

"coin\_win":20,

"Width" : 26,

"Height" : 18,

"tiles":

[

{"type":0,"visibility":true},

{"type":1,"visibility":true},

{"type":1,"visibility":true}

],

"map":

[

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":3},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":3}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}]

]

}

}

* level2.json

{

"lev":

{

"name":"level2",

"coin\_num":25,

"coin\_win":15,

"Width" : 26,

"Height" : 18,

"map":

[

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":3},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":3}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}]

]

}

}

* level3.json

{

"lev":

{

"name":"level3",

"coin\_num":25,

"coin\_win":20,

"Width" : 26,

"Height" : 18,

"map":

[

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":3},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":3}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":1},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0},{"type":1},{"type":0},{"type":1},{"type":0},{"type":1},{"type":1},{"type":1},{"type":0},{"type":0},{"type":0},{"type":0}],

[{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0},{"type":0}]

]

}

}

**DungeonOfDooom-master\Sourcecode\project\assets\test**

* test.json

{

"id": 1

}

* test.asset

testasset :)

**DungeonOfDooom-master\Sourcecode\project\src\bot\main\java\com\dod\bot\communicators**

* CommunicatorBase.java

package com.dod.bot.communicators;

import javax.ws.rs.client.\*;

import javax.ws.rs.core.MultivaluedHashMap;

import javax.ws.rs.core.MultivaluedMap;

import javax.ws.rs.core.Response;

import javax.ws.rs.ext.ContextResolver;

import org.glassfish.jersey.moxy.json.MoxyJsonConfig;

import org.glassfish.jersey.moxy.json.MoxyJsonFeature;

import java.util.HashMap;

import java.util.Map;

import java.util.UUID;

/\*\*

\* Handles communication to/from the server

\*/

public class CommunicatorBase {

private static WebTarget target;

private static String sessionId;

private static String username;

private static String password;

private static final String apiAddress = "http://localhost:8080";

protected static WebTarget getTarget() {

if(target == null)

init();

return target;

}

private static void init() {

Map<String, String> namespacePrefixMapper = new HashMap<String, String>();

namespacePrefixMapper.put("http://www.w3.org/2001/XMLSchema-instance", "xsi");

MoxyJsonConfig moxyJsonConfig = new MoxyJsonConfig()

.setNamespacePrefixMapper(namespacePrefixMapper)

.setNamespaceSeparator(':');

final ContextResolver<MoxyJsonConfig> jsonConfigResolver = moxyJsonConfig.resolver();

Client c = ClientBuilder.newBuilder()

.register(MoxyJsonFeature.class)

.register(jsonConfigResolver)

.build();

//Generate random user/pass

username = UUID.randomUUID().toString();

password = UUID.randomUUID().toString();

target = c.target(apiAddress);

sessionId = registerUserAndGetSessionId(username, password);

}

private static String registerUserAndGetSessionId(String username, String password) {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", username);

formData.add("password", password);

Response registerResponse = getTarget().path("player/register").request().post(Entity.form(formData));

//get the sessionId so we can send authorised session cookies with requests

return registerResponse.getCookies().get("JSESSIONID").getValue();

}

protected Invocation.Builder request(String path) {

Invocation.Builder request = getTarget().path(path).request();

request.cookie("JSESSIONID",sessionId);

return request;

}

protected Response post(String path, MultivaluedMap<String, String> params) {

return post(request(path), params);

}

protected Response post(Invocation.Builder request, MultivaluedMap<String, String> params) {

return request.post(Entity.form(params));

}

protected Response get(String path) {

return get(request(path));

}

protected Response get(Invocation.Builder request) {

return request.get();

}

}

* MatchCommunicator.java

package com.dod.bot.communicators;

import javax.ws.rs.core.MultivaluedHashMap;

import javax.ws.rs.core.MultivaluedMap;

import java.util.UUID;

/\*\*

\* Handles match requests to the server

\*/

public class MatchCommunicator extends CommunicatorBase {

public void joinMatch(UUID matchId) {

MultivaluedMap<String, String> params = new MultivaluedHashMap<String, String>();

params.add("matchId", matchId.toString());

post("match/join", params);

}

}

* MoveCommunicator.java

package com.dod.bot.communicators;

import javax.ws.rs.core.MultivaluedHashMap;

import javax.ws.rs.core.MultivaluedMap;

/\*\*

\* Communicates move requests to the server

\*/

public class MoveCommunicator extends CommunicatorBase {

public void moveDirection(String direction) {

MultivaluedMap<String, String> params = new MultivaluedHashMap<String, String>();

params.add("key", direction);

post("game/move", params);

}

}

* stateCommunicator.java

package com.dod.bot.communicators;

import com.dod.service.model.GameStateModel;

/\*\*

\* Communicates status requests to the server

\*/

public class stateCommunicator extends CommunicatorBase {

public GameStateModel getState() {

return get("game/status").readEntity(GameStateModel.class);

}

}

**DungeonOfDooom-master\Sourcecode\project\src\bot\main\java\com\dod\bot**

* Bot.java

package com.dod.bot;

import com.dod.service.model.GameStateModel;

import com.dod.bot.communicators.MatchCommunicator;

import com.dod.bot.communicators.MoveCommunicator;

import com.dod.bot.communicators.stateCommunicator;

import com.dod.service.model.TileModel;

import java.util.List;

import java.util.Random;

import java.util.UUID;

/\*\*

\* The bot

\*/

public class Bot {

private MatchCommunicator matchCommunicator;

private MoveCommunicator moveCommunicator;

private com.dod.bot.communicators.stateCommunicator stateCommunicator;

private double delta;

private double timestep = 200 \* 1000000;

private long previousTime;

private boolean isPlaying = false;

private GameStateModel state;

private Random random;

public Bot() {

this.matchCommunicator = new MatchCommunicator();

this.moveCommunicator = new MoveCommunicator();

this.stateCommunicator = new stateCommunicator();

random = new Random();

}

public void play(UUID matchId) {

isPlaying = true;

matchCommunicator.joinMatch(matchId);

state = stateCommunicator.getState();

delta = 0;

previousTime = System.nanoTime();

while(isPlaying) {

long currentTime = System.nanoTime();

delta += currentTime - previousTime;

previousTime = currentTime;

if(delta > timestep) {

delta -= timestep;

state = stateCommunicator.getState();

if (random.nextBoolean()) {

moveCommunicator.moveDirection(random.nextBoolean() ? "A" : "D");

} else {

moveCommunicator.moveDirection(random.nextBoolean() ? "W" : "S");

}

if (state.isHasEnded()) {

isPlaying = false;

}

}

}

}

}

* Map.java

package com.dod.bot;

import com.dod.models.Point;

import com.dod.service.model.TileModel;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Map for the bot modeled on the responses from the server

\*/

public class Map {

private ArrayList<List<TileModel>> map;

public Map() {

map = new ArrayList<List<TileModel>>();

}

public void addTile(TileModel[] tiles) {

int xMax = 0;

int yMax = 0;

for(TileModel tile : tiles) {

if(tile.getPosition().x > xMax) xMax = tile.getPosition().x;

if(tile.getPosition().y > yMax) yMax = tile.getPosition().y;

}

for(int x = 0; x < xMax; x++) {

List<TileModel> row = map.get(x);

if(row == null) {

row = new ArrayList<TileModel>();

map.add(row);

}

for(int y =0; y < yMax; y++) {

TileModel tile = null;

for(TileModel tileInput : tiles) {

if(tileInput.getPosition().equals(new Point(x,y))) {

tile = tileInput;

break;

}

}

//row.set(y, tile);

}

}

}

}

* Main.java

package com.dod.bot;

import java.util.UUID;

/\*\*

\* Gets command parameters and intitialises bot

\*/

public class Main {

/\*\*

\* Start the bot

\* @param args expects 1 argument of ID for match to join

\*/

public static void main(String args[]) {

UUID matchId = null;

try {

matchId = UUID.fromString(args[0]);

}

catch(Exception e) {

e.printStackTrace();

return;

}

Bot bot = new Bot();

bot.play(matchId);

}

}

**DungeonOfDooom-master\Sourcecode\project\src\bot**

* pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>dungeon-of-doom</groupId>

<artifactId>dungeon-of-doom-bot</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-moxy</artifactId>

<version>2.24.1</version>

</dependency>

<dependency>

<groupId>dungeon-of-doom</groupId>

<artifactId>dungeon-of-doom-service</artifactId>

<version>1.0</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<!-- Mark JAR as executable -->

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-jar-plugin</artifactId>

<version>3.0.2</version>

<configuration>

<archive>

<manifest>

<addClasspath>true</addClasspath>

<classpathPrefix>lib/</classpathPrefix>

<mainClass>com.dod.bot.Main</mainClass>

</manifest>

</archive>

</configuration>

</plugin>

</plugins>

</build>

</project>

**DungeonOfDooom-master\Sourcecode\project\src\Client\assets**

* style.css

/\* Dungon of Doom CSS stylesheet 2016 University of Bath \*/

ul {

list-style-type: none;

margin: 0;

padding: 0;

overflow: hidden;

background-color: #333;

}

ul#logged-in-header {

display: none;

}

ul#logged-in-header li {

cursor: pointer;

}

li {

float: left;

}

li a {

display: block;

color: #EEE;

text-align: center;

padding: 14px 16px;

text-decoration: none;

}

li a:hover:not(.active) {

background-color: #000000;

}

.join-link {

color:#11D;

cursor: pointer;

}

.active {

background-color: #af222a;

}

body {

background-image: url('header.jpg');

background-color: #cccccc;

background-size: cover;

font-family: 'VT323', monospace;

color: #EEE;

}

section {

position: fixed;

border: #333333;

background-color: rgba(52, 7, 5, 0.55);

margin-right: 7%;

margin-left: 7%;

margin-top: 3%;

width: 87%;

height: 80%;

text-align: center;

display:none; !important

}

.center {

position :relative;

margin-left: 30%;

width: 45%;

padding: 20px;

}

h1 {

text-decoration: underline;

color: #EEE;

font-size: 40px;

}

#score-table {

border-collapse: collapse;

width: 100%;

}

#score-table td, #score-table th {

border: 1px solid #ddd;

font-size: 20px;

text-align: center;

padding: 8px;

color: #EEE;

}

#score-table tr:hover {

background-color: #333;

}

#score-table th {

padding-top: 12px;

padding-bottom: 12px;

text-align: center;

font-size: 30px;

color: #EEE;

}

footer {

background-color: rgba(0, 0, 0, 0.77);

width: 100%;

bottom: 0;

position: fixed;

}

.container {

margin-left: 30%;

padding: 40px;

position: relative;

margin-top: 9%;

width: 72%;

font-size: large;

}

input[type=text], input[type=password] {

width: 40%;

padding: 12px 20px;

margin: 8px 0;

display: inline-block;

border: 1px solid #ccc;

box-sizing: border-box;

font-size: medium;

}

input {

color:#000;

}

button {

background-color: rgba(0, 0, 0, 0.69);

color: #EEE;

padding: 14px 20px;

margin: 10px;

border: none;

cursor: pointer;

width: 40%;

margin-left: 8%;

}

ul#guest-header {

list-style-type: none;

margin: 0;

padding: 0;

overflow: hidden;

background-color: #333;

}

ul#guest-header li {

text-align: center;

color:white;

font-size: 32px;

}

h3 {

text-decoration: underline;

}

.validation {

color:red;

font-weight:bold;

}

**DungeonOfDooom-master\Sourcecode\project\src\Client\scripts**

* main.js

/\*\*

\* 2016 Dungeon of Dooom University of Bath.

\* "Part of the graphic tiles used in this program is the Public domain roguelike tileset "RLTiles".

\* Some of the tiles have been modified by our Team. You can find the original tileset at: http://rltiles.sf.net

\* You can find Dungeon Crawl Stone Soup modified tilesets at: http://code.google.com/p/crawl-tiles/downloads/list"

\* Tileset was downloaded from opengameart.org/content/dungeon-crawl-32x32-tiles

\*/

game = [];

game.menu = [];

game.auth = [];

game.constants = [];

game.func = [];

game.match = [];

game.var = [];

game.match.var = [];

game.camera = {};

game.var.init = function() {

game.var.xSize = 900;

game.var.ySize = 600;

game.var.playerCharacter = {};

game.var.scale = 50;

game.var.tiles = [];

game.var.characters = [];

game.var.minCoins = {};

game.var.winText = [];

game.var.renderer = {};

game.var.stage = {};

game.var.graphics = {};

game.var.playerTitles = [];

game.var.isRunning = false;

game.var.delta = 0;

game.var.timeStep = 1000 / 20;

game.var.lastFrameTimestamp = 0;

game.var.opacityVis = 1.0;

game.var.opacityInvis = 0.3;

};

game.var.init();

game.var.colours = [];

game.var.colours.background = 0x000000;

game.var.colours.wall = 0x8c8c8c;

game.var.colours.floor = 0xbf8040;

game.var.colours.gold = 0xffff66;

game.var.colours.player = 0xff2222;

game.var.colours.exit = 0x2222ff;

game.var.colours.shaded = [];

game.var.colours.shaded.wall = 0x565656;

game.var.colours.shaded.floor = 0x8c5010;

game.var.colours.shaded.gold = 0xcccc33;

game.var.colours.shaded.player = 0xcc0000;

game.var.colours.shaded.exit = 0x0000cc;

game.match.var.isLobbying = false;

game.match.var.isWaitingTostart = false;

game.match.var.delta = 0;

game.match.var.timeStep = 1000 / 5;

game.match.var.lastFrameTimestamp = 0;

game.constants.api = "http://localhost:8080/";

game.constants.loginFailed = "Oops, that didn't work. Make sure your username/password are correct.";

game.constants.registrationFailed = "Oops, that didn't work. Fields cannot be empty or more than 255 characters.";

/\*\*

\* Generic method to make a GET request. Uses xhr fields to ensure cookies are sent across domain.

\* @param url {string}to send the request to

\* @param data {string}to send with the request

\* @param success function to execute on success

\* @param error function to execute on failure

\*/

game.func.get = function(url, data, success, error) {

$.ajax({

type: "GET",

url: url,

data: data,

success: success,

error: error,

xhrFields: {

withCredentials: true

}

});

};

/\*\*

\* Generic method to make a POST request. Uses xhr fields to ensure cookies are sent across domain.

\* @param url {string}to send the request to

\* @param data {string}to send with the request

\* @param success function to execute on success

\* @param error function to execute on failure

\*/

game.func.post = function(url, data, success, error) {

$.ajax({

type: "POST",

url: url,

data: data,

success: success,

error: error,

xhrFields: {

withCredentials: true

}

});

};

/\*\*

\* Constructs an url of an endpoint given the endpoint's controller and action names.

\* @param controller {string}the controller to contact

\* @param action {string}the action to contact

\* @returns {string} the constructed path

\*/

game.func.getApiPath = function(controller, action) {

return game.constants.api + controller + "/" + action;

};

game.func.error = function( data, reason, exception ) {

alert(' an error occurred :(');

console.log(reason);

console.log(exception);

};

game.auth.hook = function( data ) {

game.menu.clearValidation();

$('#guest-header').css('display', 'none');

$('#logged-in-header').css('display','block');

game.menu.openMatchLobby();

};

game.menu.loginFormValidation = function(message ) {

$('#login-validation').html(message);

};

game.menu.clearValidation = function() {

var validatorElements = $('.validation');

validatorElements.html('');

validatorElements.css('display', 'none');

};

game.auth.register = function() {

var endpoint = game.func.getApiPath("player","register");

var username = $("#username").val();

var password = $("#password").val();

game.func.post(endpoint,

{ "username" : username, "password" : password },

game.auth.hook,

function() { game.menu.loginFormValidation(game.constants.registrationFailed)});

};

game.auth.login = function() {

var endpoint = game.func.getApiPath("player","login");

var username = $("#username").val();

var password = $("#password").val();

game.func.post(endpoint,

{ "username" : username, "password" : password },

game.auth.hook,

function() { game.menu.loginFormValidation(game.constants.loginFailed)});

};

game.menu.openMatchLobby = function() {

game.menu.allSections.css('display','none');

game.menu.lobby.css('display','block');

game.match.var.isLobbying = true;

requestAnimationFrame(game.match.updateMatchList);

};

game.menu.openTutorial = function() {

game.match.var.isLobbying = false;

game.menu.allSections.css('display','none');

game.menu.tutorial.css('display','block');

};

game.menu.openScoreboard = function() {

game.match.var.isLobbying = false;

var endpoint = game.func.getApiPath("score","top");

game.func.get(endpoint, { }, game.menu.displayScoreboard, game.func.error);

};

/\*\*

\* Renders the scoreboard in the #score-table table.

\* @param scoreBoard the JSON object returned from a query to the score/top endpoint

\*/

game.menu.displayScoreboard = function( scoreBoard ) {

$('#score-table tbody td').remove();

$.each(scoreBoard.scores, function(i, score) {

if(score != null) {

$('#score-table tbody').append($(String.format("<tr><td>{0}</td><td>{1}</td></tr>", score.username, score.value)))

}

});

game.menu.allSections.css('display','none');

game.menu.scoreboard.css('display','block');

};

game.match.list = function() {

//todo what if the webservice thinks you're already in a match?

var endpoint = game.func.getApiPath("match","list");

game.func.get(endpoint, {}, game.menu.displayMatchList, game.menu.error);

};

game.menu.displayMatchList = function( data ) {

var matchList = $('#match-list');

matchList.empty();

$.each( data, function( i, match ) {

var entry = $( String.format("<p><a data-id='{2}' class='join-link'>Join</a> {0}'s game with {1} players</p>", match.playerNames[0], match.playerNames.length, match.id) );

matchList.append(entry);

});

$(".join-link").click(game.match.join);

};

game.match.join = function( data ) {

var id = $(data.currentTarget).data("id");

game.match.var.isLobbying = false;

game.match.var.isWaitingTostart = true;

var endpoint = game.func.getApiPath("match","join");

game.func.post(endpoint, { "matchId" : id }, game.menu.displayMatchMenu, game.menu.error);

requestAnimationFrame(game.match.updateStatus);

};

/\*\*

\* Starts a new Match by sending a request to the web service

\*/

game.match.new = function() {

var endpoint = game.func.getApiPath("match","new");

game.match.var.isLobbying = false;

game.match.var.isWaitingTostart = true;

var level = game.menu.levelChooser.val();

game.func.post(endpoint, { "level" : level }, game.menu.displayMatchMenu);

requestAnimationFrame(game.match.updateStatus);

};

/\*\*

\* Starts the player's current Match by sending a request to the web service

\*/

game.match.start = function() {

game.match.var.isWaitingTostart = false;

var endpoint = game.func.getApiPath("match","start");

requestAnimationFrame(function() {game.func.post(endpoint, null, game.menu.initGameScreen, game.func.error) });

};

/\*\*

\* Initialises the game screen. Resets game variables and creates a new HTML5 canvas. Begins the game loop.

\*/

game.menu.initGameScreen = function() {

game.var.init();

game.menu.gameContainer.empty();

game.var.renderer = PIXI.autoDetectRenderer(game.var.xSize, game.var.ySize);

game.var.renderer.backgroundColor = game.var.colours.background;

game.var.renderer.transparent = true;

game.menu.gameContainer.append(game.var.renderer.view);

game.var.stage = new PIXI.Container();

// game.var.graphics = new PIXI.Graphics();

// game.var.stage.addChild(game.var.graphics);

game.menu.match.css('display', 'none');

game.menu.game.css('display', 'block');

game.var.isRunning = true;

requestAnimationFrame(game.updateGame);

};

game.initTextWinCondition = function( character ) {

var style = {

fontFamily : 'Arial',

fontSize : '18px',

fontStyle : 'italic',

fontWeight : 'bold',

fill : '#F7EDCA',

stroke : '#4a1850',

strokeThickness : 5,

dropShadow : true,

dropShadowColor : '#000000',

dropShadowAngle : Math.PI / 6,

dropShadowDistance : 4

};

game.var.winText[character.playerName] = new PIXI.Text('Collect '+ game.var.minCoins +' coins minimum to win! You collected ' + game.var.playerCharacter.noCoins + ' coins!', style);

}

game.initPlayerTitle = function( character ) {

var style = {

fontFamily : 'Arial',

fontSize : '18px',

fontStyle : 'italic',

fontWeight : 'bold',

fill : '#F7EDCA',

stroke : '#4a1850',

strokeThickness : 5,

dropShadow : true,

dropShadowColor : '#000000',

dropShadowAngle : Math.PI / 6,

dropShadowDistance : 4

};

game.var.playerTitles[character.playerName] = new PIXI.Text(character.playerName, style);

};

/\*\*

\* Renders the current game state to the canvas.

\*/

game.render = function() {

//game.var.graphics.clear();

game.var.stage = new PIXI.Container();

for(x = 0; x < game.var.tiles.length; x++) {

var row = game.var.tiles[x];

if(typeof row !== 'undefined') {

for (y = 0; y < game.var.tiles[x].length; y++) {

var tile = game.var.tiles[x][y];

if(typeof tile !== 'undefined') {

var tilePositionX = (x \* game.var.scale) - game.camera.x;

var tilePositionY = (y \* game.var.scale) - game.camera.y;

if (tile.type == 0) {

var wall = PIXI.Sprite.fromImage('assets/wall.png');

wall.x = tilePositionX;

wall.y = tilePositionY;

wall.alpha = tile.visible ? game.var.opacityVis : game.var.opacityInvis;

game.var.stage.addChild(wall);

}

else if (tile.type == 1) {

var floor = PIXI.Sprite.fromImage('assets/floor.png');

floor.x = tilePositionX;

floor.y = tilePositionY;

floor.alpha = tile.visible ? game.var.opacityVis : game.var.opacityInvis;

game.var.stage.addChild(floor);

}

else if (tile.type == 2) {

var coin = PIXI.Sprite.fromImage('assets/coin.png');

coin.x = tilePositionX;

coin.y = tilePositionY;

coin.alpha = tile.visible ? game.var.opacityVis : game.var.opacityInvis;

game.var.stage.addChild(coin);

}

else if(tile.type == 3) {

var exit = PIXI.Sprite.fromImage('assets/exit.png');

exit.x = tilePositionX;

exit.y = tilePositionY;

exit.alpha = tile.visible ? game.var.opacityVis : game.var.opacityInvis;

game.var.stage.addChild(exit);

}

if (tile.visible && tile.character !== null) {

var positionX = tilePositionX + game.var.scale / 2;

var positionY = tilePositionY + game.var.scale / 2;

var char = PIXI.Sprite.fromImage('assets/char.png');

char.x = positionX - game.var.scale / 2;

char.y = positionY - game.var.scale / 2;

game.var.stage.addChild(char);

var character = game.var.tiles[x][y].character;

var playerTitle = game.var.playerTitles[character.playerName];

if(typeof playerTitle === 'undefined') {

game.initPlayerTitle(character);

}

else {

playerTitle.x = positionX - game.var.scale;

playerTitle.y = positionY - game.var.scale;

}

game.var.stage.addChild(game.var.playerTitles[character.playerName]);

game.initTextWinCondition(game.var.tiles[x][y].character);

game.var.stage.addChild(game.var.winText[character.playerName]);

}

}

}

}

}

game.var.isRunning = true;

game.var.renderer.render(game.var.stage);

};

game.setAllTilesNotVisible = function() {

$.each(game.var.tiles, function(x, row) {

if(typeof row != 'undefined') {

$.each(row, function(y, tile) {

if(typeof tile != 'undefined')

tile.visible = false;

});

}

});

};

/\*\*

\* Updates the current game status based on the result from a status request.

\* Will end the gam eif the status response indicates that the game is over.

\* @param status

\*/

game.updateStatus = function( status ) {

game.var.characters = status.characters;

game.var.playerCharacter = status.playerCharacter;

game.var.minCoins = status.minNumOfCoins;

game.camera.x = (game.var.playerCharacter.position.x \* game.var.scale) - (game.var.xSize / 2);

game.camera.y = (game.var.playerCharacter.position.y \* game.var.scale) - (game.var.ySize / 2);

game.setAllTilesNotVisible();

$.each( status.tiles, function ( i, tile ) {

tile.character = null;

game.var.addTile(tile);

});

$.each( status.characters, function( i, character) {

game.var.tiles[character.position.x][character.position.y].character = character;

});

if(status.hasEnded) {

game.var.isRunning = false;

game.end();

}

else {

game.render();

}

};

/\*\*

\* Adds a tile if it doesn't already exist in memory, or updates the tile if it does.

\* Will expand the size of game.var.tiles if it isn't large enough.

\* @param tile

\*/

game.var.addTile = function( tile ) {

var pos = tile.position;

if(typeof game.var.tiles[pos.x] === 'undefined') {

game.var.tiles[pos.x] = [];

}

tile.visible = true;

game.var.tiles[pos.x][pos.y] = tile;

};

game.fetchStatus = function() {

var endpoint = game.func.getApiPath("game","status");

game.func.get(endpoint, {}, game.updateStatus, game.func.error);

};

/\*\*

\* Makes a game status request if game.var.timestep has passed since the last request.

\* Loops while game.var.isRunning

\* @param timestamp

\*/

game.updateGame = function( timestamp ) {

if(game.var.lastFrameTimestamp == 0) {

game.var.lastFrameTimestamp = timestamp + game.var.timeStep;

}

game.var.delta += timestamp - game.var.lastFrameTimestamp;

game.var.lastFrameTimestamp = timestamp;

if(game.var.delta > game.var.timeStep) {

game.fetchStatus();

game.var.delta -= game.var.timeStep;

}

if(game.var.isRunning) {

requestAnimationFrame(game.updateGame);

}

};

/\*\*

\* Makes a match list request if game.match.var.timeStep has passed since the last request

\* Loops while game.match.var.isLobbying

\* @param timestamp

\*/

game.match.updateMatchList = function( timestamp ) {

if(game.match.var.lastFrameTimestamp == 0) {

game.match.var.lastFrameTimestamp = timestamp + game.match.var.timeStep;

}

game.match.var.delta += timestamp - game.match.var.lastFrameTimestamp;

game.match.var.lastFrameTimestamp = timestamp;

if(game.match.var.delta >= game.match.var.timeStep) {

game.match.list();

game.match.var.delta -= game.match.var.timeStep;

}

if(game.match.var.isLobbying) {

requestAnimationFrame(game.match.updateMatchList);

}

};

/\*\*

\* Makes a Match Status request if game.match.var.timeStep has passed since the last request.

\* Loops while game.match.var.isWaitingToStart

\* @param timestamp

\*/

game.match.updateStatus = function( timestamp ) {

if(game.match.var.lastFrameTimestamp == 0) {

game.match.var.lastFrameTimestamp = timestamp + game.match.var.timeStep;

}

game.match.var.delta += timestamp - game.match.var.lastFrameTimestamp;

game.match.var.lastFrameTimestamp = timestamp;

if(game.match.var.delta >= game.match.var.timeStep) {

game.match.fetchStatus();

game.match.var.delta -= game.match.var.timeStep;

}

if(game.match.var.isWaitingTostart) {

requestAnimationFrame(game.match.updateStatus);

}

};

game.match.fetchStatus = function() {

var endpoint = game.func.getApiPath("match","status");

game.func.get(endpoint, {}, game.menu.displayMatchMenu, game.func.error);

};

/\*\*

\* Generates the Match Status details on the Match screen.

\* @param data Match Status as a JSON object

\*/

game.menu.displayMatchMenu = function( data ) {

game.menu.lobby.css('display','none');

game.menu.match.css('display','block');

var matchDeatils = $("#match-details");

matchDeatils.empty();

matchDeatils.append($("<h2>Waiting to start.</h2>"));

matchDeatils.append($(String.format("<p>To add a bot use the following ID: {0}</p>", data.id)));

matchDeatils.append($("<h3>Players:</h3>"));

$.each( data.playerNames, function( i, name ) {

var entry = $( String.format("<p>{0}</p>", name) );

matchDeatils.append(entry);

});

if(data.state == 'Ingame') {

game.match.var.isWaitingTostart = false;

game.menu.initGameScreen();

}

};

game.match.leave = function() {

game.var.isRunning = false;

var endpoint = game.func.getApiPath("match","leave");

requestAnimationFrame(function() {game.func.post(endpoint, { }, game.menu.openMatchLobby, game.func.error)});

};

game.menu.move = function( key ) {

var endpoint = game.func.getApiPath("game","move");

game.var.status = game.func.post(endpoint, {"key" : key}, game.updateStatus, game.func.error);

};

/\*\*

\* Builds the end-game screen and switches to it

\* @param result a MatchResultModel object

\*/

game.menu.showEndGameScreen = function( result ) {

if(result.winner == game.var.playerCharacter.playerName) {

$('#end-game-title').html("YOU WIN!")

}

else {

$('#end-game-title').html("YOU LOOSE!")

}

$('#end-game-detail').html(String.format("{0} wins with {1} coins! Your score is {2} ", result.winner, result.winnerCoins, result.score));

game.menu.gameContainer.empty();

game.menu.game.css('display','none');

game.menu.end.css('display','block');

};

game.end = function() {

game.var.isRunning = false;

var endpoint = game.func.getApiPath("match","result");

game.func.get(endpoint, { }, game.menu.showEndGameScreen, game.func.error);

};

$( document ).ready(function() {

game.menu.login = $('#login');

game.menu.lobby = $('#lobby');

game.menu.levelChooser = $('#level');

game.menu.match = $('#match');

game.menu.tutorial = $('#tutorial');

game.menu.end = $('#end-game');

game.menu.scoreboard = $('#score');

game.menu.game = $('#game');

game.menu.gameContainer = $('#game-container');

game.menu.allSections = $('section');

game.menu.login.css('display', 'block');

$('#register-btn').click(game.auth.register);

$('#login-btn').click(game.auth.login);

$('#new-match-btn').click(game.match.new);

$('#start-match-btn').click(game.match.start);

$('#match-leave-btn').click(game.match.leave);

$('#return-btn').click(game.match.leave);

$('#lobby-link').click(game.menu.openMatchLobby);

$('#tutorial-link').click(game.menu.openTutorial);

$('#score-link').click(game.menu.openScoreboard);

window.addEventListener('keydown', function(event) {

if (game.var.isRunning) {

switch (event.keyCode) {

case 65:

case 37: // Left

game.menu.move('A');

break;

case 87:

case 38: // Up

game.menu.move('W');

break;

case 68:

case 39: // Right

game.menu.move('D');

break;

case 83:

case 40: // Down

game.menu.move('S');

break;

}

}

}, false);

});

**DungeonOfDooom-master\Sourcecode\project\src\Client**

* index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Dungeon of Dooom</title>

<link href="https://fonts.googleapis.com/css?family=Ubuntu" rel="stylesheet">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">

<link href="assets/style.css" rel="stylesheet">

</head>

<body>

<ul id="logged-in-header">

<li><a id="lobby-link">Lobby</a></li>

<li><a id="tutorial-link">How to play</a></li>

<li><a id="score-link">Score Table</a></li>

<li style="float:right"><a class="active">Dungeon of Doom</a></li>

</ul>

<ul id="guest-header">

<li >Dungeon of Doom</li>

</ul>

<section id="login">

<div class="col-md-5 col-md-offset-3">

<div class="row">

<label for="username"><b>Username</b></label>

<input type="text" placeholder="Enter Username" id="username" name="username" required>

</div>

<div class="row">

<label for="password"><b>Password</b></label>

<input type="password" placeholder="Enter Password" id="password" name="password" required>

</div>

<div class="row">

<p id="login-validation" class="validation"></p>

</div>

<div class="row">

<button id="login-btn" class="btn btn-danger btn-lg">Login</button>

</div>

<div class="row">

<button id="register-btn" class="btn btn-danger btn-lg">Register</button>

</div>

</div>

</section>

<section id="lobby">

<h1>Matches</h1>

<p id="match-list"></p>

<div class="row">

<button id="new-match-btn" class="btn btn-danger btn-lg">New Match</button>

<label for="level">Level</label>

<input type="number" id="level" name="level" min="1" max="3" value="1">

</div>

</section>

<section id="match">

<h1>Match</h1>

<p id="match-details"></p>

<button id="start-match-btn" class="btn btn-danger btn-lg">Start</button>

</section>

<section id="game">

<div id="game-container"></div>

<div clas="row">

<button id="match-leave-btn" class="btn btn-danger btn-lg">Leave</button>

</div>

</section>

<section id="end-game">

<h1 id="end-game-title"></h1>

<p id="end-game-detail"></p>

<button id="return-btn" class="btn btn-danger btn-lg">Return</button>

</section>

<section id="tutorial">

<aside class="left">

<h1> Instructions </h1><br>

The dungeon of doom is an online multiplayer game, which starts with registration of player.

The game icludes the functionality to choose either single player or multi-player as well as an option to view top score achivied by player.

<br><br>

The objective of the game is to collect the specified amount of gold in the dungeon and get to the exit before other player.

</aside>

<div class="right">

<img src="assets/layout.jpeg"/>

</div>

</section>

<section id="score">

<h1 id="homeHeading">Score Table</h1>

<table id= "score-table">

<thead>

<tr>

  <th>Username</th>

  <th>Score</th>

</tr>

</thead>

<tbody>

</tbody>

</table>

</section>

<footer>

<p>Dungeon of Dooom coursework entry for University of Bath Software Engineering unit</p>

</footer>

<script type="text/javascript" src="scripts/lib/stringformat.js"></script>

<script type="text/javascript" src="scripts/lib/jquery.min.js"></script>

<script type="text/javascript" src="scripts/lib/pixi.js"></script>

<script type="text/javascript" src="scripts/main.js"></script>

</body>

</html>

**DungeonOfDooom-master\Sourcecode\project\src\domain\com\dod\db\repositories**

* DatabaseRepository.java

package com.dod.db.repositories;

import com.dod.db.DatabaseConnection;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

/\*\*

\* <pre>

\* A base class of the Repository pattern

\* Introduces the generic getStatement() method to reuse that code across the different repositories

\* </pre>

\*/

public class DatabaseRepository<T> {

/\*\*

\* Make a SELECT query to fetch the unique object in question from the database

\* @param object an instance of the object in question with the unique field (but not necessarily others) filled out

\* @return An instance of the object

\* @throws SQLException if the statement fails or connection cannot be established

\*/

public T get(T object) throws SQLException { return null; }

/\*\*

\* Make an INSERT query to insert the object in question into the database

\* @param object the object in question

\* @return true if successful, false otherwise

\* @throws SQLException

\*/

public boolean insert(T object) throws SQLException { return false; }

/\*\*

\* Make a DELETE query to delete the object in question from the database

\* @param object the object in question with the unique field (but not necessarily others) filled out

\* @return true if successful, false otherwise

\* @throws SQLException when the statement fails

\*/

public boolean delete(T object) throws SQLException { return false; }

protected PreparedStatement ps;

/\*\*

\* Prepares a statement from a string using the database connection

\* @param text the text of the statement

\* @return a PreparedStatement instance

\* @throws SQLException when the statement fails

\*/

protected PreparedStatement getStatement(String text) throws SQLException

{

Connection con = DatabaseConnection.getConnection();

PreparedStatement ps = con.prepareStatement(text);

return ps;

}

}

* IPlayerRepository.java

package com.dod.db.repositories;

import com.dod.models.Player;

import java.sql.SQLException;

/\*\*

\* <pre>

\* Follows the Repository pattern.

\* Intended for selecting/inserting/deleting "Player" entries from the database.

\* </pre>

\*/

public interface IPlayerRepository {

/\*\*

\* Make a SELECT query to fetch the unique Player in question from the database

\* @param object an instance of the Player in question with the unique field (but not necessarily others) filled out

\* @return Player object fetched from the database

\* @throws SQLException if the statement fails or connection cannot be established

\*/

Player get(Player object) throws SQLException;

/\*\*

\* Make an INSERT query to insert the Player in question into the database

\* @param object the Player in question

\* @return true if successful, false otherwise

\* @throws SQLException when the statement fails

\*/

boolean delete(Player object) throws SQLException;

/\*\*

\* Make a DELETE query to delete the Player in question from the database

\* @param object the Player in question with the unique field (but not necessarily others) filled out

\* @return true if successful, false otherwise

\* @throws SQLException when the statement fails

\*/

boolean insert(Player object) throws SQLException;

}

* IScoreRepository.java

package com.dod.db.repositories;

import com.dod.models.Player;

import com.dod.models.Score;

import java.sql.SQLException;

/\*\*

\* <pre>

\* Follows the Repository pattern.

\* Intended for selecting/inserting/deleting "Score" entries from the database.

\* </pre>

\*/

public interface IScoreRepository {

/\*\*

\* Make a SELECT query to fetch the unique Score in question from the database

\* @param object an instance of the Score in question with the unique field (but not necessarily others) filled out

\* @return Score fetched from the database

\* @throws SQLException if the statement fails or connection cannot be established

\*/

Score get(Score object) throws SQLException;

/\*\*

\* Make an INSERT query to insert the Score in question into the database

\* @param object the Score in question

\* @return true if successful, false otherwise

\* @throws SQLException

\*/

boolean insert(Score object) throws SQLException;

/\*\*

\* Make a DELETE query to delete the Score in question from the database

\* @param object the Score in question with the unique field (but not necessarily others) filled out

\* @return true if successful, false otherwise

\* @throws SQLException

\*/

boolean delete(Score object) throws SQLException;

/\*\*

\* Get the 10 highest scores from database

\* @return Score[] array of 10 Score objects

\* @throws SQLException when the statement fails

\*/

Score[] getHighestScores() throws SQLException;

/\*\*

\* Get the 10 highest scores of the player

\* @param object Player object

\* @return Score[] array of 10 Score objects

\* @throws SQLException when the statement fails

\*/

Score[] getPlayerScores(Player object) throws SQLException;

}

* PlayerRepository.java

package com.dod.db.repositories;

import com.dod.models.Player;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

/\*\*

\* <pre>

\* Implements IPlayerRepository.

\* Follows the Repository pattern.

\* Intended for selecting/inserting/deleting "Player" entries from the database.

\* </pre>

\*/

public class PlayerRepository extends DatabaseRepository<Player> implements IPlayerRepository {

private final String deleteQuery = "DELETE FROM player WHERE username = ?";

private final String getQuery = "SELECT username, password, salt FROM player WHERE username = ?";

private final String insertQuery = "INSERT INTO player (username, password, level, salt) VALUES (?, ?, 0, ?)";

/\*\*

\* Make a SELECT query to fetch the unique Player in question from the database

\* @param object an instance of the Player in question with the unique field (but not necessarily others) filled out

\* @return Player object fetched from the database

\* @throws SQLException if the statement fails or connection cannot be established

\*/

@Override

public Player get(Player object) throws SQLException {

PreparedStatement statement = getStatement(getQuery);

statement.setString(1, object.getUsername());

ResultSet rs = statement.executeQuery();

if (rs.next())

return new Player(rs.getString("username"), rs.getString("password"), rs.getBytes("salt"));

else

return null;

}

/\*\*

\* Make an INSERT query to insert the Player in question into the database

\* @param object the Player in question

\* @return true if successful, false otherwise

\* @throws SQLException when the statement fails

\*/

@Override

public boolean delete(Player object) throws SQLException {

PreparedStatement statement = getStatement(deleteQuery);

statement.setString(1, object.getUsername());

if (statement.executeUpdate() == 0) {

return false;

} else {

return true;

}

}

/\*\*

\* Make a DELETE query to delete the Player in question from the database

\* @param object the Player in question with the unique field (but not necessarily others) filled out

\* @return true if successful, false otherwise

\* @throws SQLException when the statement fails

\*/

@Override

public boolean insert(Player object) throws SQLException{

PreparedStatement statement = getStatement(insertQuery);

statement.setString(1, object.getUsername());

statement.setString(2, object.getHashedPassword());

statement.setBytes(3, object.getSalt());

try {

statement.executeUpdate();

}

catch (SQLException e) {

return false;

}

statement.close();

return true;

}

}

* ScoreRepository.java

package com.dod.db.repositories;

import com.dod.models.Player;

import com.dod.models.Score;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

/\*\*

\* <pre>

\* Implements IPlayerRepository.

\* Follows the Repository pattern.

\* Intended for selecting/inserting/deleting "Score" entries from the database.

\* </pre>

\*/

public class ScoreRepository extends DatabaseRepository<Score> implements IScoreRepository {

private final String getPlayerQuery = "SELECT \* FROM score WHERE username='?' ORDER BY value DESC LIMIT 10";

private final String deleteQuery = "DELETE FROM score WHERE id = ?";

private final String getScoreQuery = "SELECT \* FROM score WHERE id = ?";

private final String getQuery = "SELECT \* FROM score ORDER BY value DESC LIMIT 10";

private final String insertQuery = "INSERT INTO score (username, value) VALUES (?, ?)";

/\*\*

\* Inserts a score value to score table of database based on player's

\* username.

\* @param scoreObject current score that we need to score

\* @return true if insertion was successful else false

\* @throws SQLException when the statement fails

\*/

@Override

public boolean insert(Score scoreObject) throws SQLException {

PreparedStatement statement = getStatement(insertQuery);

statement.setString(1, scoreObject.getUsername());

statement.setInt(2, scoreObject.getValue());

try {

statement.executeUpdate();

} catch (SQLException e) {

return false;

}

statement.close();

return true;

}

/\*\*

\* Delete a score row from database

\* !! We should not use that.

\* @param object score object to delete

\* @return true if the deletion was successful else false

\* @throws SQLException when the statement fails

\*/

@Override

public boolean delete(Score object) throws SQLException {

PreparedStatement statement = getStatement(deleteQuery);

statement.setInt(1, object.getId());

if (statement.executeUpdate() == 0) {

return false;

} else {

return true;

}

}

/\*\*

\* Get the 10 highest scores from database

\* @return Score[] array of 10 Score objects

\* @throws SQLException when the statement fails

\*/

public Score[] getHighestScores() throws SQLException {

PreparedStatement statement = getStatement(getQuery);

//statement.setString(1, object.getUsername());

ResultSet rs = statement.executeQuery();

Score[] result = new Score[10];

int i = 0;

while (rs.next()) {

Score temp = new Score(rs.getInt("id"), rs.getString("username"), rs.getInt("value"));

result[i] = temp;

i++;

}

return result;

}

/\*\*

\* Get the 10 highest scores of the player

\* @param object Player object

\* @return Score[] array of 10 Score objects

\* @throws SQLException when the statement fails

\*/

public Score[] getPlayerScores(Player object) throws SQLException {

PreparedStatement statement = getStatement(getPlayerQuery);

statement.setString(1, object.getUsername());

ResultSet rs = statement.executeQuery();

Score[] result = new Score[10];

int i = 0;

while (rs.next()) {

Score temp = new Score(rs.getInt("id"), rs.getString("username"), rs.getInt("value"));

result[i] = temp;

i++;

}

return result;

}

/\*\*

\* returns a Score based on id from the database

\* @param Score to be fetched must have unique identifier populated

\* @return Score object

\* @throws SQLException when the statement fails

\*/

@Override

public Score get(Score object) throws SQLException {

PreparedStatement statement = getStatement(getScoreQuery);

statement.setInt(1, object.getId());

ResultSet rs = statement.executeQuery();

if (rs.next()) {

return new Score(rs.getInt(1), rs.getString(2), rs.getInt(3));

} else {

return null;

}

}

}

**DungeonOfDooom-master\Sourcecode\project\src\domain\com\dod\db**

* DatabaseConnection.java

package com.dod.db;

import com.mysql.jdbc.jdbc2.optional.MysqlDataSource;

import java.sql.Connection;

import java.sql.SQLException;

/\*\*

\* Stores a connection to the database using the singleton pattern

\*/

public class DatabaseConnection {

private static Connection connection;

/\*\*

\* A static connection to ensure that all sessions use the same MySql connection

\* Could be done more intelligently with connection pooling

\* @return Connection instance

\* @throws SQLException when the database connection cannot be established

\*/

public static Connection getConnection() throws SQLException {

if(connection != null) {

return connection;

}

else {

MysqlDataSource dataSource = new MysqlDataSource();

dataSource.setUser("dungeonofdoom");

dataSource.setPassword("Delicate.Sunshine.Twist.Myth32");

dataSource.setServerName("localhost");

dataSource.setDatabaseName("dungeonofdoom");

connection = dataSource.getConnection();

return connection;

}

}

/\*\*

\* Closes the connection

\*/

public static void Close() {

try {

connection.close();

}

catch(SQLException e) {

System.console().printf(e.getMessage());

}

}

}

**DungeonOfDooom-master\Sourcecode\project\src\domain\com\dod\game**

* IMatchList.java

package com.dod.game;

import com.dod.models.Match;

import java.util.List;

import java.util.UUID;

/\*\*

\* Stores ongoing matches in memory and provides functions to access these matches.

\*/

public interface IMatchList {

/\*\*

\* Returns a singleton instance of MatchList, creating it if it hasn't been initialised yet.

\* @return MatchList

\*/

void addMatch(Match match);

/\*\*

\* Gets all matches that are in the Lobbying state

\* @return List of Match objects

\*/

List<Match> getLobbyingMatches();

/\*\*

\* Gets a Match by a particular ID. Returns null if the match is missing.

\* @param id the UUID that corresponds to the match to be fetched

\* @return Match

\*/

Match getMatch(UUID id);

/\*\*

\* Gets a match by player name. Each player should only have one match. Returns null if player has no match.

\* @param username the username of the player

\* @return Match

\*/

Match getMatchForPlayer(String username);

/\*\*

\* Returns true if the player has a match in the list

\* @param username the player's username

\* @return true if the player has a match in the list otherwise false

\*/

boolean playerHasMatch(String username);

/\*\*

\* Removes the match fitting the specified ID from the list

\* @param id the UUID that corresponds to the particular Match to be removed

\*/

void removeMatch(UUID id);

}

* MatchList.java

package com.dod.game;

import com.dod.models.Match;

import com.dod.models.MatchState;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import java.util.UUID;

/\*\*

\* <pre>

\* Implementation of IMatchList

\* Stores ongoing matches in memory and provides functions to access these matches.

\* Uses a singleton so that we can fetch the same object between requests

\* (And because this is much easier to test than making all methods static)

\* </pre>

\*/

public class MatchList implements IMatchList {

private static IMatchList instance;

/\*\*

\* Returns a singleton instance of MatchList, creating it if it hasn't been initialised yet.

\* @return MatchList

\*/

public static IMatchList instance() {

if(instance == null) {

instance = new MatchList();

}

return instance;

}

private List<Match> ongoingMatches = new ArrayList();

/\*\*

\* Adds a match to the list

\* @param match the match to add

\*/

public void addMatch(Match match) {

ongoingMatches.add(match);

}

/\*\*

\* Gets all matches that are in the Lobbying state

\* @return List of Match objects

\*/

public List<Match> getLobbyingMatches() {

List<Match> result = new ArrayList();

for(Match match : ongoingMatches) {

if(match.getState() == MatchState.Lobbying) {

result.add(match);

}

}

return result;

}

/\*\*

\* Gets a Match by a particular ID. Returns null if the match is missing.

\* @param id the UUID that corresponds to the match to be fetched

\* @return Match

\*/

public Match getMatch(UUID id) {

Match result = null;

for(Match match : ongoingMatches) {

if(match.getId().equals(id)) {

result = match;

break;

}

}

return result;

}

/\*\*

\* Gets a match by player name. Each player should only have one match. Returns null if player has no match.

\* @param username the username of the player

\* @return Match

\*/

public Match getMatchForPlayer(String username) {

Match result = null;

for(Match match: ongoingMatches) {

if(match.hasCharacter(username)) {

result = match;

break;

}

}

return result;

}

/\*\*

\* Returns true if the player has a match in the list

\* @param username the player's username

\* @return true if the player has a match in the list otherwise false

\*/

public boolean playerHasMatch(String username) {

return getMatchForPlayer(username) != null;

}

/\*\*

\* Removes the match fitting the specified ID from the list

\* @param id the UUID that corresponds to the particular Match to be removed

\*/

public void removeMatch(UUID id) {

for(Match match: ongoingMatches) {

if(match.getId().equals(id)) {

ongoingMatches.remove(match);

break;

}

}

}

}

**DungeonOfDooom-master\Sourcecode\project\src\domain\com\dod\models**

* Character.java

package com.dod.models;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* <pre>

\* A Character is a fictional entity that moves around the game world.

\* A Character belongs to a Player.

\* A Character has a position and can interact with coins and the exit.

\* </pre>

\*/

public class Character {

private Point position;

private Player player;

private int collectedCoins;

private List<Point> collectedCoinsPos;

public Character(Point position, Player player) {

this.position = position;

this.player = player;

this.collectedCoinsPos = new ArrayList<>();

collectedCoins = 0;

}

/\*\*

\* The player's position in the game world

\* @return Point

\*/

public Point getPosition() {

return position;

}

/\*\*

\* The player's position in the game world

\* @param position Point

\*/

public void setPosition(Point position) {

this.position = position;

}

/\*\*

\* The Player that this Character belongs to

\* @return Player

\*/

public Player getPlayer() {

return player;

}

/\*\*

\* The Player that this Character belongs to

\* @param player Player

\*/

public void setPlayer(Player player) {

this.player = player;

}

public int getCollectedCoins() {

return collectedCoins;

}

public void setCollectedCoins(int collectedCoins) {

this.collectedCoins = collectedCoins;

}

/\*\*

\* Keeps track of which coins on the map this Character has collected.

\* This enables us to leave the coin on the Map once it has been picked up, thereby allowing other players

\* to pick it up, and yet not send the same coin to the same player's client again.

\* @return a list of Point objects that represent the points on the map where the Character has collected a coin

\*/

public List<Point> getCollectedCoinsPos() {

return collectedCoinsPos;

}

/\*\*

\* Keeps track of which coins on the map this Character has collected.

\* This enables us to leave the coin on the Map once it has been picked up, thereby allowing other players

\* to pick it up, and yet not send the same coin to the same player's client again.

\* @param newPoint the Point to add to the collection

\*/

public void addCollectedCoinsPos(Point newPoint) {

this.collectedCoinsPos.add(newPoint);

}

}

* Map.java

package com.dod.models;

import java.io.Serializable;

import java.util.Random;

/\*\*

\* <pre>

A Map stores a 2-dimensional grid of Tiles.

A Map has a name, width, height and numbe rof coins total and required to win.

\* </pre>

\*/

public class Map {

protected int width;

protected int height;

protected String name;

protected int totalNumberOfCoins;

protected int numberOfCoinsNeededToWin;

protected Tile[][] tiles;

public Map(int width, int height) {

tiles = new Tile[width][height];

}

public Map(String name, int totalNumberOfCoins, int numberOfCoinsNeededToWin, int width, int height, Point mapSize) {

this.name = name;

this.totalNumberOfCoins = totalNumberOfCoins;

this.numberOfCoinsNeededToWin = numberOfCoinsNeededToWin;

this.width = width;

this.height = height;

tiles = new Tile[mapSize.x][mapSize.y];

}

public void setTile(Point position, Tile tile) {

tiles[position.x][position.y] = tile;

}

public String getName(){

return name;

}

public void setName(String name){

this.name = name;

}

/\*\*

\* The total number of coins that should be created in the map.

\* @return int

\*/

public int getCoinNo(){

return totalNumberOfCoins;

}

/\*\*

\* The total number of coins that should be created in the map.

\* @param coin\_no int

\*/

public void setCoinNo(int coin\_no){

this.totalNumberOfCoins = coin\_no;

}

/\*\*

\* The total number of coins needed to win on this map

\* @return int

\*/

public int getCoinWin(){

return numberOfCoinsNeededToWin;

}

/\*\*

\* The total number of coins needed to win on this map

\* @param coin\_win int

\*/

public void setCoinWin(int coin\_win){

this.numberOfCoinsNeededToWin = coin\_win;

}

public Tile getTile(Point point) {

return tiles[point.x][point.y];

}

public int getWidth() {

return width;

}

public int getHeight() {

return height;

}

/\*\*

\* Gets a random position of a tile that is not a wall, coin or exit.

\* @return Point

\*/

public Point getRandomFreeTilePoint() {

Random random = new Random();

Point point = null;

while(point == null) {

int x = random.nextInt(width-1);

int y = random.nextInt(height-1);

if(tiles[x][y].getType() == TileType.Empty.getValue()) {

point = new Point(x,y);

}

}

return point;

}

}

* Match.java

package com.dod.models;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

import java.util.UUID;

/\*\*

\* Represents a match

\*/

public class Match {

private UUID id;

private Map map;

private List<Character> characters;

private MatchState state;

private long timer;

private int score;

public Match(Map map) {

this.id = UUID.randomUUID();

this.map = map;

this.characters = new ArrayList();

state = MatchState.Lobbying;

timer = 0;

score = 0;

}

public Map getMap() {

return map;

}

public void addCharacter(Player player, Point position) {

characters.add(new Character(position, player));

}

public void removeCharacter(Player player) {

for(Character character : characters) {

if(character.getPlayer().getUsername().equals(player.getUsername())) {

characters.remove(character);

break;

}

}

}

public Character getCharacter(String username) {

Character result = null;

for(Character character : characters) {

if(character.getPlayer().getUsername().equals(username)) {

result = character;

break;

}

}

return result;

}

public String[] getPlayerNames() {

String[] names = new String[characters.size()];

for(int i = 0; i < characters.size(); i++) {

names[i] = characters.get(i).getPlayer().getUsername();

}

return names;

}

public boolean hasCharacter(String userName) {

return getCharacter(userName) != null;

}

public UUID getId() {

return id;

}

public void startGame() {

state = MatchState.Ingame;

}

public MatchState getState() {

return state;

}

public void setState(MatchState state) {

this.state = state;

}

public List<Character> getCharactersOnTile(Point point) {

List<Character> charactersOnTile = new ArrayList();

for(Character character :characters) {

if(character.getPosition().equals(point)) {

charactersOnTile.add(character);

}

}

return charactersOnTile;

}

public Character getCharacterWithHighestCoins() {

Character character = null;

for(Character c : characters) {

if(character == null || c.getCollectedCoins() > character.getCollectedCoins()) {

character = c;

}

}

return character;

}

public long getTimer() {

return timer;

}

public void setTimer(long timer) {

this.timer = timer;

}

public int getScore() { return this.score; }

public void setScore(int score) { this.score = score; }

}

* MatchState.java

package com.dod.models;

/\*\*

\* The state of a Match.

\*/

public enum MatchState {

Lobbying,

Ingame,

Over

}

* Player.java

package com.dod.models;

/\*\*

\* <pre>

\* A Player represents the user that is in control of the game client

\* A Player can sign in with a username or password

\* A Player has a level and a password salt

\* A Player's password is always hashed

\* </pre>

\*/

public class Player {

private String username;

private String hashedPassword;

private int level;

private byte[] salt;

public Player(String name) {

this.username = name;

}

public Player(String name, String hashedPassword, byte[] salt) {

this.username = name;

this.hashedPassword = hashedPassword;

this.salt = salt;

}

public String getUsername() {

return username;

}

public void setUsername(String value) {

username = value;

}

public String getHashedPassword() {

return hashedPassword;

}

public void setHashedPassword(String hashedPassword) {

this.hashedPassword = hashedPassword;

}

public int getLevel() {

return level;

}

public void setLevel(int level) {

this.level = level;

}

public byte[] getSalt() {

return salt;

}

public void setSalt(byte[] salt) {

this.salt = salt;

}

}

* Point.java

package com.dod.models;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* Bean class for storing a point (or vertex) in the map.

\*/

@XmlRootElement

public class Point {

public int x;

public int y;

public Point() {}

public Point(int x, int y) {

this.x = x;

this.y = y;

}

@Override

public boolean equals(Object obj) {

boolean result = false;

if (obj instanceof Point) {

Point point = (Point) obj;

if (point.x == x && point.y == y) {

result = true;

}

}

return result;

}

}

* Score.java

package com.dod.models;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* <pre>

\* A Score stores the points a Player achieved when they completed a Match.

\* A Score as an ID in order to store the Score as a unique databaes record

\* A Score also has a value and the username of the player that the score is related to.

\* </pre>

\*/

@XmlRootElement

public class Score {

private int id;

private String username;

private int value;

public Score(int id, String username, int value) {

this.id = id;

this.username = username;

this.value = value;

}

public Score(String username, int value) {

this.id = -1;

this.username = username;

this.value = value;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getValue() {

return value;

}

public void setValue(int value) {

this.value = value;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

}

* Tile.java

package com.dod.models;

import java.io.Serializable;

/\*\*

\* <pre>

\* A Tile represents single tile on the grid that is the Map

\* A Tile has a Type that indicates whether it is eg a wall, floor, coin or exit tile.

\* A Tile may or may not be visible

\* </pre>

\*/

public class Tile {

protected int type;

protected boolean visibility;

public Tile(int type, boolean visibility){

this.setType(type);

this.setVisibility(visibility);

}

public Tile(int type) {

this.type = type;

}

public int getType() {

return type;

}

public void setType(int type) {

this.type = type;

}

public boolean isVisible() {

return visibility;

}

public void setVisibility(boolean visibility) {

this.visibility = visibility;

}

public String toString(){

return "Type: "+this.type+"\nVisibility: "+this.visibility;

}

}

* TileType.java

package com.dod.models;

/\*\*

\* The type of a tile, i.e is this tile a wall, floor or something else.

\*/

public enum TileType {

Wall(0),

Empty(1),

Coin(2),

Exit(3);

private final int value;

TileType(int value) {

this.value = value;

}

public int getValue() {

return value;

}

}

**DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\constant**

* Assets.java

package com.dod.service.constant;

/\*\*

\* A set of static constant strings that define the paths to our assets.

\* Must always start with a slash.

\*/

public class Assets {

public static final String MapLevelOne = "/maps/level1.json";

public static final String MapLevelFormat = "/maps/level%s.json";

}

**DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\controller**

* GameController.java

package com.dod.service.controller;

import com.dod.game.MatchList;

import com.dod.models.Map;

import com.dod.models.Player;

import com.dod.service.model.GameStateModel;

import com.dod.service.service.MovementService;

import com.dod.service.service.StateService;

import com.dod.service.service.VisibilityService;

import org.glassfish.grizzly.http.server.Request;

import javax.validation.constraints.NotNull;

import javax.ws.rs.\*;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import java.sql.SQLException;

/\*\*

\* A controller to manage in-game game-related functionality ie getting the current state of the world or moving.

\*/

@Path("game")

public class GameController {

@Context

private Request request;

StateService stateService;

MovementService movementService;

public GameController() {

stateService = new StateService(new VisibilityService(), MatchList.instance());

movementService = new MovementService();

}

/\*\*

\* Responds with the current gamestate from the Player's Character's perspective, i.e. only returning visible tiles

\* If Player has no current ongoing Match returns 500 error.

\* @return Response 200 OK with GameStateModel as a JSON object

\*/

@GET

@Produces(MediaType.APPLICATION\_JSON)

@Path("status")

public Response status() {

String username = (String)request.getSession().getAttribute("player");

GameStateModel state = stateService.GetState(new Player(username));

return Response

.ok()

.entity(state)

.build();

}

/\*\*

\* An endpoint to request the Player's Character move once in a particular direction.

\* Responds with game status after move.

\* If Player has no current ongoing Match returns 500 error.

\* @param direction a char from {W,S,A,D} pertaining to a particular direction in the WASD layout, must not be null

\* @return Response 200 OK with GameStateModel as a JSON object

\*/

@POST

@Produces(MediaType.APPLICATION\_JSON)

@Path("move")

public Response move(@NotNull @FormParam("key") String direction) {

String username = (String)request.getSession().getAttribute("player");

try {

movementService.Move(direction, new Player(username));

}

catch(SQLException e) {

e.printStackTrace();

return Response.serverError().build();

}

GameStateModel state = stateService.GetState(new Player(username));

return Response

.ok()

.entity(state)

.build();

}

}

* MatchController.java

package com.dod.service.controller;

import com.dod.db.repositories.PlayerRepository;

import com.dod.game.MatchList;

import com.dod.models.Match;

import com.dod.models.Player;

import com.dod.service.model.MatchStatus;

import com.dod.service.service.IOService;

import com.dod.service.service.MatchService;

import com.dod.service.service.ParseService;

import org.glassfish.grizzly.http.server.Request;

import javax.validation.constraints.NotNull;

import javax.ws.rs.\*;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import java.sql.SQLException;

import java.util.UUID;

/\*\*

\* A controller to manage Matches- joining, listing, starting a new one etc.

\*/

@Path("match")

public class MatchController {

@Context

private Request request;

private MatchService matchService;

public MatchController() {

this.matchService = new MatchService(

new IOService(),

new ParseService(),

new PlayerRepository(),

MatchList.instance());

}

/\*\*

\* Responds with the status of the player's current Match.

\* If Player has no current Match returns a 500 error.

\* @return Response 200 OK with MatchStatus encoded in JSON

\*/

@GET

@Produces(MediaType.APPLICATION\_JSON)

@Path("status")

public Response status() {

String username = (String)request.getSession().getAttribute("player");

return Response

.ok()

.entity(matchService.getStatus(new Player(username)))

.build();

}

/\*\*

\* Starts a new Match in a particular level and responds with that Match's status

\* @param level int the level to load for this Match, must not be null

\* @return Response 200 OK with MatchStatus encoded in JSON or null if a Match cannot be crated

\*/

@POST

@Produces(MediaType.APPLICATION\_JSON)

@Path("new")

public Response newMatch(

@NotNull @FormParam("level") int level

) {

String userName = (String)request.getSession().getAttribute("player");

MatchStatus newMatch = matchService.createMatch(userName, level);

if(newMatch != null) {

return Response

.ok()

.entity(newMatch)

.build();

}

else {

return Response.serverError().build();

}

}

/\*\*

\* Changes a Match's status to Ingame (marking the start of the Match for all players)

\* @return MatchStatus encoded in JSON

\*/

@POST

@Produces(MediaType.TEXT\_PLAIN)

@Path("start")

public Response start() {

String username = (String)request.getSession().getAttribute("player");

matchService.startMatch(new Player(username));

return Response

.ok()

.build();

}

/\*\*

\* Lists all currently lobbying matches in a JSON array

\* @return Response 200 OK JSON array with encoded MatchStatus for each lobbying Match

\*/

@GET

@Produces(MediaType.APPLICATION\_JSON)

@Path("list")

public Response list() {

MatchStatus[] matches = matchService.getLobbyingMatches();

return Response

.ok()

.entity(matches)

.build();

}

/\*\*

\* Joins the Player in an ongoing Match

\* @param matchId the UUID ID of the Match, must not be null

\* @return Response 200 OK with the latest MatchStatus encoded in JSON

\*/

@POST

@Produces(MediaType.APPLICATION\_JSON)

@Path("join")

public Response join(

@NotNull @FormParam("matchId") UUID matchId

) {

String username = (String)request.getSession(false).getAttribute("player");

try {

matchService.joinMatch(new Player(username), matchId);

return Response

.ok()

.entity(matchService.getStatus(new Player(username)))

.build();

}

catch(SQLException e) {

e.printStackTrace();

return Response

.serverError()

.build();

}

}

/\*\*

\* Removes the Player from their current Match

\* @return Response 200 OK with a blank body

\*/

@POST

@Produces(MediaType.TEXT\_PLAIN)

@Path("leave")

public Response leave() {

String username = (String)request.getSession(false).getAttribute("player");

matchService.leaveMatch(new Player(username));

return Response

.ok()

.build();

}

/\*\*

\* Fetches the result of a Match from memory

\* @return Resepons 200 OK with JSON encoded MatchResultModel

\*/

@GET

@Produces(MediaType.APPLICATION\_JSON)

@Path("result")

public Response result() {

String username = (String)request.getSession(false).getAttribute("player");

return Response

.ok()

.entity(matchService.getMatchResult(new Player(username)))

.build();

}

}

* PlayerController.java

package com.dod.service.controller;

import com.dod.db.repositories.PlayerRepository;

import com.dod.service.model.LoginModel;

import com.dod.service.service.AuthenticationService;

import com.dod.service.service.IAuthenticationService;

import javax.servlet.http.HttpServletRequest;

import javax.validation.constraints.Max;

import javax.validation.constraints.Min;

import javax.validation.constraints.NotNull;

import javax.ws.rs.FormParam;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import java.sql.SQLException;

import org.glassfish.grizzly.http.server.Request;

import org.hibernate.validator.constraints.Length;

/\*\*

\* <pre>

\* Manages registering and logging in a player

\* Creates the session that other controllers can use to fetch user details

\* </pre>

\*/

@Path("player")

public class PlayerController {

@Context

private Request request;

IAuthenticationService service;

public PlayerController() {

service = new AuthenticationService(new PlayerRepository());

}

/\*\*

\* Authorises a user and starts a session with them

\* @param username must be unique, not empty and less than 256 characters

\* @param password must not be empty and less than 256 characters

\* @return Response with blank body, 200 if successful otherwise 400 or 500

\*/

@POST

@Produces(MediaType.TEXT\_PLAIN)

@Path("login")

public Response login(

@NotNull @Length(min = 1, max =255) @FormParam("username") String username,

@NotNull @Length(min = 1, max =255) @FormParam("password") String password

) {

boolean isAuthorised = service.Login(new LoginModel(username, password));

if(isAuthorised) {

request.getSession(true);

request.getSession().setAttribute("player",username);

return Response.ok().build();

}

else {

return Response

.status(403)

.build();

}

}

/\*\*

\* Registers a user for the service. Username must be unique.

\* @param username must be unique, not empty and less than 256 characters

\* @param password must not be empty and less than 256 characters

\* @return Response with blank body, 200 if successful otherwise 400 or 500

\*/

@POST

@Produces(MediaType.TEXT\_PLAIN)

@Path("register")

public Response register(

@NotNull @Length(min = 1, max =255) @FormParam("username") String username,

@NotNull @Length(min = 1, max = 255) @FormParam("password") String password

) {

boolean success = service.Register(new LoginModel(username,password));

if(success) {

request.getSession(true);

request.getSession().setAttribute("player", username);

return Response.ok().build();

}

else {

return Response.status(400).build();

}

}

}

* ScoreController.java

package com.dod.service.controller;

import com.dod.db.repositories.IScoreRepository;

import com.dod.db.repositories.ScoreRepository;

import com.dod.service.model.ScoreboardModel;

import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import java.sql.SQLException;

/\*\*

\* Fetches and returns the top scores

\*/

@Path("score")

public class ScoreController {

private IScoreRepository repository;

public ScoreController() {

this.repository = new ScoreRepository();

}

/\*\*

\* Fetches the top 10 scores across all players.

\* @return Response 200 OK with a JSON encoded ScoreboardModel or 500 if an error occurred

\*/

@GET

@Produces(MediaType.APPLICATION\_JSON)

@Path("top")

public Response top() {

ScoreboardModel scoreBoard = null;

try {

scoreBoard = new ScoreboardModel(repository.getHighestScores());

}

catch(SQLException e) {

e.printStackTrace();

return Response.serverError().build();

}

return

Response.ok()

.entity(scoreBoard)

.build();

}

}

**DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\filters**

* corsFilter.java

package com.dod.service.filters;

/\*\*

\* Adds CORS filter to header, enabling cross-origin AJAX requests

\* Based on: https://stackoverflow.com/questions/28065963/how-to-handle-cors-using-jax-rs-with-jersey

\*/

import java.io.IOException;

import javax.ws.rs.container.ContainerRequestContext;

import javax.ws.rs.container.ContainerResponseContext;

import javax.ws.rs.container.ContainerResponseFilter;

import javax.ws.rs.ext.Provider;

@Provider

public class corsFilter implements ContainerResponseFilter {

/\*\*

\* Adds CORS headers to the Response before sending it

\* @param request ContainerRequestContext

\* @param response ContainerResponseContext

\*/

@Override

public void filter(ContainerRequestContext request,

ContainerResponseContext response) {

response.getHeaders().add("Access-Control-Allow-Origin", "http://localhost:63342");

response.getHeaders().add("Access-Control-Allow-Headers",

"origin, content-type, accept, authorization");

response.getHeaders().add("Access-Control-Allow-Credentials", "true");

response.getHeaders().add("Access-Control-Allow-Methods",

"GET, POST, PUT, DELETE, OPTIONS, HEAD");

}

}

**DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\model**

* CharacterModel.java

package com.dod.service.model;

import com.dod.models.Point;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* A simpler model of Character for JSON encoding

\*/

@XmlRootElement

public class CharacterModel {

private String playerName;

private int noCoins;

private Point position;

public CharacterModel() { }

public CharacterModel(String playerName, int noCoins, Point position) {

this.playerName = playerName;

this.noCoins = noCoins;

this.position = position;

}

public String getPlayerName() {

return playerName;

}

public void setPlayerName(String playerName) {

this.playerName = playerName;

}

public int getNoCoins() {

return noCoins;

}

public void setNoCoins(int noCoins) {

this.noCoins = noCoins;

}

public Point getPosition() {

return position;

}

public void setPosition(Point position) {

this.position = position;

}

}

* GameStateModel.java

package com.dod.service.model;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* Represents the current GameState. Intended to be communicated to the client via JSON encoding.

\*/

@XmlRootElement

public class GameStateModel {

private TileModel[] tiles;

private CharacterModel[] characters;

private CharacterModel playerCharacter;

private int minNumOfCoins;

private boolean hasEnded;

public GameStateModel() { }

public GameStateModel(TileModel[] tiles, CharacterModel[] characters, CharacterModel playerCharacter, boolean hasEnded, int minNumOfCoins) {

this.tiles = tiles;

this.characters = characters;

this.playerCharacter = playerCharacter;

this.hasEnded = hasEnded;

this.minNumOfCoins = minNumOfCoins;

}

public TileModel[] getTiles() {

return tiles;

}

public void setTiles(TileModel[] tiles) {

this.tiles = tiles;

}

public CharacterModel[] getCharacters() {

return characters;

}

public void setCharacters(CharacterModel[] characters) {

this.characters = characters;

}

/\*\*

\* The Character belonging to the Player that made the request

\* @return Character

\*/

public CharacterModel getPlayerCharacter() {

return playerCharacter;

}

/\*\*

\* The Character belonging to the Player that made the request

\* @param playerCharacter Character

\*/

public void setPlayerCharacter(CharacterModel playerCharacter) {

this.playerCharacter = playerCharacter;

}

/\*\*

\* Whether the match is ongoing- triggers the client's endgame if true

\* @return boolean

\*/

public boolean isHasEnded() {

return hasEnded;

}

/\*\*

\* Whether the match is ongoing- triggers the client's endgame if true

\* @param hasEnded boolean

\*/

public void setHasEnded(boolean hasEnded) {

this.hasEnded = hasEnded;

}

/\*\*

\* The minimum number of coins needed to win the Match

\* @return int

\*/

public int getMinNumOfCoins() { return minNumOfCoins; }

/\*\*

\* \* The minimum number of coins needed to win the Match

\* @param minNumOfCoins int

\*/

public void setMinNumOfCoins(int minNumOfCoins) { this.minNumOfCoins = minNumOfCoins; }

}

* LoginModel.java

package com.dod.service.model;

import com.dod.models.Player;

/\*\*

\* Simple model/bean used to pass information to/from the AuthorisationService

\*/

public class LoginModel {

private String userName;

private String password;

public LoginModel(String userName, String password) {

this.userName = userName;

this.password = password;

}

public String getUserName() {

return userName;

}

public void setUserName(String userName) {

this.userName = userName;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

/\*\*

\* Convenience method to return the LoginModel's username in the Player model

\* @return Player

\*/

public Player asPlayer() {

return new Player(userName);

}

}

* MatchResultModel.java

package com.dod.service.model;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* Models the information the client needs to display the end-game screen when the game ends.

\*/

@XmlRootElement

public class MatchResultModel {

private String winner;

private int winnerCoins;

private int score;

public MatchResultModel(String winner, int winnerCoins, int score) {

this.winner = winner;

this.winnerCoins = winnerCoins;

this.score = score;

}

public MatchResultModel() { }

public String getWinner() {

return winner;

}

public void setWinner(String winner) {

this.winner = winner;

}

public int getWinnerCoins() {

return winnerCoins;

}

public void setWinnerCoins(int winnerCoins) {

this.winnerCoins = winnerCoins;

}

public int getScore() { return score; }

public void setScore(int score) { this.score = score; }

}

* MatchStatus.java

package com.dod.service.model;

import com.dod.models.Match;

import javax.xml.bind.annotation.XmlID;

import javax.xml.bind.annotation.XmlRootElement;

import java.util.UUID;

/\*\*

\* Models the current state of a lobbying match.

\*/

@XmlRootElement

public class MatchStatus

{

private String[] playerNames;

@XmlID

private UUID id;

private String state;

public MatchStatus() {}

public MatchStatus(Match match) {

this.playerNames = match.getPlayerNames();

this.id = match.getId();

this.state = match.getState().toString();

}

public String[] getPlayerNames() {

return playerNames;

}

public UUID getId() {

return id;

}

public void setPlayerNames(String[] playerNames) {

this.playerNames = playerNames;

}

public void setId(UUID id) {

this.id = id;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

}

* ScoreboardModel.java

package com.dod.service.model;

import com.dod.models.Score;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* Models a collection of scores to be displayed on a score table

\*/

@XmlRootElement

public class ScoreboardModel {

Score[] scores;

public ScoreboardModel(Score[] scores) {

this.scores = scores;

}

public ScoreboardModel() {

}

public Score[] getScores() {

return scores;

}

public void setScores(Score[] scores) {

this.scores = scores;

}

}

* TileModel.java

package com.dod.service.model;

import com.dod.models.Point;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* A simpler Tile model just for JSON encoding

\*/

@XmlRootElement

public class TileModel {

private int type;

private Point position;

public TileModel() { }

public TileModel(int type, Point position) {

this.type = type;

this.position = position;

}

public int getType() {

return type;

}

public void setType(int type) {

this.type = type;

}

public Point getPosition() {

return position;

}

public void setPosition(Point position) {

this.position = position;

}

}

**DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service\service**

* AuthenticationService.java

package com.dod.service.service;

import com.dod.db.repositories.IPlayerRepository;

import com.dod.models.Player;

import com.dod.service.model.LoginModel;

import org.apache.commons.codec.binary.Base64;

import javax.crypto.SecretKey;

import javax.crypto.SecretKeyFactory;

import javax.crypto.spec.PBEKeySpec;

import java.security.NoSuchAlgorithmException;

import java.security.SecureRandom;

import java.security.spec.InvalidKeySpecException;

import java.sql.SQLException;

/\*\*

\* <pre>

\* Handles authenticating a user against their user/pass combo

\* Uses a salt, generated using a secure RNG

\* Uses PlayerRepository to fetch Player database details

\* </pre>

\*/

public class AuthenticationService implements IAuthenticationService {

IPlayerRepository repository;

public AuthenticationService(IPlayerRepository repository) {

this.repository = repository;

}

/\*\*

\* Registers a new user

\* @param model LoginModel containing the user/pass to be registered

\* @return boolean true if successful otherwise false

\*/

@Override

public boolean Register(LoginModel model) {

boolean result = false;

Player player = player = model.asPlayer();

Player repositoryPlayer = null;

try {

repositoryPlayer = repository.get(player);

}

catch(SQLException e) {

e.printStackTrace();

}

if(repositoryPlayer == null) {

try {

generateSalt(player);

player.setHashedPassword(hashAndSalt(model.getPassword(), player.getSalt()));

repository.insert(player);

result = true;

} catch (Exception e) {

result = false;

}

}

return result;

}

/\*\*

\* Registers a new user

\* @param model LoginModel containing the user/pass to be authorised

\* @return boolean true if the user is authorised, otherwise false

\*/

@Override

public boolean Login(LoginModel model) {

boolean result = false;

try {

Player player = repository.get(model.asPlayer());

if (hashAndSalt(model.getPassword(), player.getSalt()).equals(player.getHashedPassword())) {

result = true;

}

}

catch(Exception e) {

e.printStackTrace();

}

return result;

}

/\*\*

\* Generates a random secure salt

\* @param player Player to set the salt for- gets inserted into the database later

\* @throws NoSuchAlgorithmException could be thrown due to a dependency problem

\*/

private void generateSalt(Player player) throws NoSuchAlgorithmException {

byte[] salt = SecureRandom.getInstance("SHA1PRNG").generateSeed(32);

player.setSalt(salt);

}

/\*\*

\* Hashes and salts a password

\* @param password the password to be hashed/salted

\* @param salt the salt to salt the password with

\* @return String the hashed/salted password

\* @throws NoSuchAlgorithmException could be thrown due to a dependency problem

\* @throws InvalidKeySpecException could be thrown due to a dependency problem

\*/

private String hashAndSalt(String password, byte[] salt) throws NoSuchAlgorithmException, InvalidKeySpecException {

String hashedPassword = hash(password, salt);

return Base64.encodeBase64String(salt) + hashedPassword;

}

/\*\*

\* Hashes a password

\* @param password the password to be hashed

\* @param salt the salt to secure the password with

\* @return String the hashed password

\* @throws NoSuchAlgorithmException could be thrown due to a dependency problem

\* @throws InvalidKeySpecException could be thrown due to a dependency problem

\*/

private String hash(String password, byte[] salt) throws NoSuchAlgorithmException, InvalidKeySpecException {

SecretKeyFactory f = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");

SecretKey key = f.generateSecret(new PBEKeySpec(

password.toCharArray(), salt, 20\*1000, 256)

);

return Base64.encodeBase64String(key.getEncoded());

}

}

* IAuthenticationService.java

package com.dod.service.service;

import com.dod.service.model.LoginModel;

import java.sql.SQLException;

/\*\*

\* <pre>

\* Handles authenticating a user against their user/pass combo

\* </pre>

\*/

public interface IAuthenticationService {

/\*\*

\* Registers a new user

\* @param model LoginModel containing the user/pass to be registered

\* @return boolean true if successful otherwise false

\*/

boolean Register(LoginModel model);

/\*\*

\* Registers a new user

\* @param model LoginModel containing the user/pass to be authorised

\* @return boolean true if the user is authorised, otherwise false

\*/

boolean Login(LoginModel model);

}

* IIOService.java

package com.dod.service.service;

import org.json.simple.JSONObject;

import org.json.simple.parser.ParseException;

import java.io.IOException;

/\*\*

\* Handles IO within the Service

\*/

public interface IIOService {

/\*\*

\* Fetches an asset as a String

\* @param path String the path to the asset we are to fetch

\* @return String the contents of the asset

\* @throws IOException if the file is missing

\*/

String getString(String path) throws IOException;

/\*\*

\* Fetches an asset as parsed JSON

\* @param path String the path to the asset we are to fetch

\* @return JSONObject the parsed content of the asset

\* @throws IOException if the file is missing

\* @throws ParseException if the file isn't encoded in valid JSON

\*/

JSONObject getJsonObject(String path) throws IOException, ParseException;

}

* IMatchService.java

package com.dod.service.service;

import com.dod.models.Player;

import com.dod.service.model.MatchResultModel;

import com.dod.service.model.MatchStatus;

import java.sql.SQLException;

import java.util.UUID;

/\*\*

\* Manages joining/starting/ending matches.

\*/

public interface IMatchService {

/\*\*

\* Creates a new Match

\* @param userName String username of the Player who is starting the Match

\* @param level int the number of the level to load for this Match

\* @return MatchStatus of the newly created Match

\*/

MatchStatus createMatch(String userName, int level);

/\*\*

\* Changes a Match's state to InGame

\* @param player Player whose ongoing Match will be modified

\*/

void startMatch(Player player);

/\*\*

\* Returns the MatchStatus for a particular Player's Match

\* @param player Player whose ongoing Match will be fetched

\* @return

\*/

MatchStatus getStatus(Player player);

/\*\*

\* Removes a Player from their current ongoing Match

\* @param player Player the Player whom will be removed from their ongoing Match

\*/

void leaveMatch(Player player);

/\*\*

\* Changes a Match's state to Over

\* @param player Player whose ongoing Match will be modified

\*/

void endMatch(Player player);

/\*\*

\* Adds the Player to a particular Match

\* @param player Player whom will be added

\* @param matchID UUID of the Match that player will be addd to

\* @throws SQLException thrown if Player doesn't exist or a SQL connectivity issue occurs

\*/

void joinMatch(Player player, UUID matchID) throws SQLException;

/\*\*

\* Get all Matches currently in the Lobbying state

\* @return MatchStatus[] array of all Matches in the Lobbying state

\*/

MatchStatus[] getLobbyingMatches();

/\*\*

\* Gets the MatchResultModel for a finished Match

\* todo why not remove the Player from the Match at this point rather than send another request?

\* @param player Player the Player that has a finished Match

\* @return MatchResultModel pertaining to the player's Match

\*/

MatchResultModel getMatchResult(Player player);

}

* IMovementService.java

package com.dod.service.service;

import com.dod.models.Character;

import com.dod.models.Map;

import com.dod.models.Player;

import com.dod.models.Point;

import java.sql.SQLException;

/\*\*

\* Interface for MovementService.

\* Handles game logic to move a character from one point to another.

\*/

public interface IMovementService {

/\*\*

\* Moves the Player in a particular direction. Will increment player's gold if interacting with gold coins, can

\* trigger end of the Match when player interacts with Exit.

\* @param direction String a char from {W,S,A,D} pertaining to a particular direction in the WASD layout

\* @param player Player whom's Character will be moved

\* @return Point that the Player has moved to

\* @throws SQLException if the database cannot be reached or statement fails while inserting new score

\*/

Point Move(String direction, Player player) throws SQLException;

}

* IOService.java

package com.dod.service.service;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

import java.io.IOException;

import java.nio.charset.StandardCharsets;

import java.nio.file.Files;

import java.nio.file.Paths;

/\*\*

\* Handles IO within the Service

\*/

public class IOService implements IIOService {

private String pathToAssets = "..//..//assets";

private JSONParser parser;

public IOService(String pathToAssets) {

this.pathToAssets = pathToAssets;

parser = new JSONParser();

}

public IOService() {

parser= new JSONParser();

}

/\*\*

\* Fetches an asset as a String

\* @param path String the path to the asset we are to fetch

\* @return String the contents of the asset

\* @throws IOException if the file is missing

\*/

@Override

public String getString(String path) throws IOException {

byte[] encoded = Files.readAllBytes(Paths.get(pathToAssets + path));

return new String(encoded, StandardCharsets.UTF\_8);

}

/\*\*

\* Fetches an asset as parsed JSON

\* @param path String the path to the asset we are to fetch

\* @return JSONObject the parsed content of the asset

\* @throws IOException if the file is missing

\* @throws ParseException if the file isn't encoded in valid JSON

\*/

@Override

public JSONObject getJsonObject(String path) throws IOException, ParseException {

String input = getString(path);

return (JSONObject) parser.parse(input);

}

}

* IParseService.java

package com.dod.service.service;

import com.dod.models.Map;

import org.json.simple.JSONObject;

/\*\*

\* Parses JSON objects- namely the Map

\*/

public interface IParseService {

/\*\*

\* Parses a Map object from it's JSON encoding

\* @param input JSONObject a JSON encoding of the Map

\* @return Map an initialised Map parsed from JSON

\* @throws NullPointerException may be thrown by SimpleJson while parsing

\*/

Map parseMap(JSONObject input) throws NullPointerException;

}

* IStateService.java

package com.dod.service.service;

import com.dod.models.Player;

import com.dod.service.model.GameStateModel;

/\*\*

\* Generates a representation of the current game state form the perspective of a particular character

\*/

public interface IStateService {

/\*\*

\* Generates and returns a representation of the current game state form the perspective of a particular character

\* @param player Player the Player a GameStateModel will be generated for

\* @return GameStateModel a model of the current game state

\*/

GameStateModel GetState(Player player);

}

* IVisibilityService.java

package com.dod.service.service;

import com.dod.models.Map;

import com.dod.models.Character;

/\*\*

\* Calculates the visible tiles from the perspective of a particular Character

\*/

public interface IVisibilityService {

/\*\*

\* Generates a copy of a Map with the correct isVisible flags set for the perspective of a particular Character

\* @param deungeonMap the Map pchar resides in

\* @param pchar the Character the perspective of which we're generating visibility with

\* @return a copy of dungeonMap with correct isVisible flags set for the perspective of pchar

\*/

Map createVisibleMap(Map deungeonMap, Character pchar);

}

* MatchService.java

package com.dod.service.service;

import com.dod.db.repositories.IPlayerRepository;

import com.dod.game.IMatchList;

import com.dod.models.\*;

import com.dod.models.Character;

import com.dod.service.constant.Assets;

import com.dod.service.model.MatchResultModel;

import com.dod.service.model.MatchStatus;

import java.sql.SQLException;

import java.util.Date;

import java.util.List;

import java.util.UUID;

/\*\*

\* <pre>

\* Manages joining/starting/ending matches.

\* Makes heavy use of MatchList to store matches in memory.

\* Uses PlayerRepository to fetch Player data.

\* Uses IOService and ParseService to load levels when starting a new Match.

\* </pre>

\*/

public class MatchService implements IMatchService {

private IIOService ioService;

private IParseService parseService;

private IPlayerRepository playerRepository;

private IMatchList matchList;

public MatchService(IIOService ioService, IParseService parseService, IPlayerRepository playerRepository, IMatchList matchList) {

this.ioService = ioService;

this.parseService = parseService;

this.playerRepository = playerRepository;

this.matchList = matchList;

}

/\*\*

\* Creates a new Match

\* @param userName String username of the Player who is starting the Match

\* @param level int the number of the level to load for this Match

\* @return MatchStatus of the newly created Match

\*/

@Override

public MatchStatus createMatch(String userName, int level) {

Map map = null;

Player player;

try {

String path = String.format(Assets.MapLevelFormat, Integer.toString(level));

map = parseService.parseMap(ioService.getJsonObject(path));

player = playerRepository.get(new Player(userName));

}

catch(Exception e) {

e.printStackTrace();

return null;

}

Match match = new Match(map);

match.addCharacter(player, map.getRandomFreeTilePoint());

for(int i = 0; i < map.getCoinNo(); i++) {

map.getTile(map.getRandomFreeTilePoint()).setType(TileType.Coin.getValue());

}

matchList.addMatch(match);

return new MatchStatus(match);

}

/\*\*

\* Changes a Match's state to InGame

\* @param player Player whose ongoing Match will be modified

\*/

@Override

public void startMatch(Player player) {

Match match = matchList.getMatchForPlayer(player.getUsername());

Date temp = new Date();

match.setTimer(temp.getTime());

match.setState(MatchState.Ingame);

}

/\*\*

\* Returns the MatchStatus for a particular Player's Match

\* @param player Player whose ongoing Match will be fetched

\* @return

\*/

@Override

public MatchStatus getStatus(Player player) {

if(!matchList.playerHasMatch(player.getUsername())) {

return null;

} else {

Match match = matchList.getMatchForPlayer(player.getUsername());

return new MatchStatus(match);

}

}

/\*\*

\* Removes a Player from their current ongoing Match

\* @param player Player the Player whom will be removed from their ongoing Match

\*/

@Override

public void leaveMatch(Player player) {

Match match = matchList.getMatchForPlayer(player.getUsername());

match.removeCharacter(player);

}

/\*\*

\* Changes a Match's state to Over

\* @param player Player whose ongoing Match will be modified

\*/

@Override

public void endMatch(Player player) {

Match match = matchList.getMatchForPlayer(player.getUsername());

matchList.removeMatch(match.getId());

}

/\*\*

\* Adds the Player to a particular Match

\* @param player Player whom will be added

\* @param matchID UUID of the Match that player will be addd to

\* @throws SQLException thrown if Player doesn't exist or a SQL connectivity issue occurs

\*/

@Override

public void joinMatch(Player player, UUID matchId) throws SQLException {

Match match = matchList.getMatch(matchId);

player = playerRepository.get(player);

match.addCharacter(player, match.getMap().getRandomFreeTilePoint());

}

/\*\*

\* Get all Matches currently in the Lobbying state

\* @return MatchStatus[] array of all Matches in the Lobbying state

\*/

@Override

public MatchStatus[] getLobbyingMatches() {

List<Match> matches = matchList.getLobbyingMatches();

MatchStatus[] matchStatuses = new MatchStatus[matches.size()];

for(int i = 0; i < matches.size(); i++) {

matchStatuses[i] = new MatchStatus(matches.get(i));

}

return matchStatuses;

}

/\*\*

\* Gets the MatchResultModel for a finished Match

\* todo why not remove the Player from the Match at this point rather than send another request?

\* @param player Player the Player that has a finished Match

\* @return MatchResultModel pertaining to the player's Match

\*/

@Override

public MatchResultModel getMatchResult(Player player) {

Match match = matchList.getMatchForPlayer(player.getUsername());

Character winner = match.getCharacterWithHighestCoins();

return new MatchResultModel(winner.getPlayer().getUsername(), winner.getCollectedCoins(), match.getScore());

}

}

* MovementService.java

package com.dod.service.service;

import com.dod.db.repositories.IScoreRepository;

import com.dod.db.repositories.ScoreRepository;

import com.dod.game.IMatchList;

import com.dod.game.MatchList;

import com.dod.models.\*;

import com.dod.models.Character;

import java.sql.SQLException;

import java.util.Date;

/\*\*

\* Implementation of IMovementService

\*/

public class MovementService implements IMovementService {

IMatchList matchList;

IScoreRepository scoreRepository;

public MovementService() {

this.matchList = MatchList.instance();

this.scoreRepository = (IScoreRepository)new ScoreRepository();

}

/\*\*

\* Moves the Player in a particular direction. Will increment player's gold if interacting with gold coins, can

\* trigger end of the Match when player interacts with Exit.

\* @param direction String a char from {W,S,A,D} pertaining to a particular direction in the WASD layout

\* @param player Player whom's Character will be moved

\* @return Point that the Player has moved to

\* @throws SQLException if the database cannot be reached or statement fails while inserting new score

\*/

@Override

public Point Move(String direction, Player player) throws SQLException {

Match match = matchList.getMatchForPlayer(player.getUsername());

Character pChar = match.getCharacter(player.getUsername());

Map dungeonMap = match.getMap();

Point newPoint;

switch (direction) {

case "W":

// check if movement valid

newPoint = new Point(pChar.getPosition().x, pChar.getPosition().y - 1);

return updatePosition(newPoint, dungeonMap, pChar);

case "D":

newPoint = new Point(pChar.getPosition().x + 1, pChar.getPosition().y);

return updatePosition(newPoint, dungeonMap, pChar);

case "S":

newPoint = new Point(pChar.getPosition().x, pChar.getPosition().y + 1);

return updatePosition(newPoint, dungeonMap, pChar);

case "A":

newPoint = new Point(pChar.getPosition().x - 1, pChar.getPosition().y);

return updatePosition(newPoint, dungeonMap, pChar);

default:

return pChar.getPosition();

}

}

/\*\*

\* Decides whether or not to update the Player's Position and interacts with special Tiles.

\* @param newPoint Point the Point the Character wishes to move to

\* @param dungeonMap Map that the Character is moving in

\* @param pChar Character that is moving

\* @return Point the Point that the Character is now in

\* @throws SQLException if the database cannot be reached or statement fails while inserting new score

\*/

private Point updatePosition(Point newPoint, Map dungeonMap, Character pChar) throws SQLException {

if (dungeonMap.getTile(newPoint).getType() == TileType.Empty.getValue()) {

pChar.setPosition(newPoint);

} else if (dungeonMap.getTile(newPoint).getType() == TileType.Coin.getValue()){

pChar.setPosition(newPoint);

if (!pChar.getCollectedCoinsPos().contains(newPoint)) {

pChar.setCollectedCoins(pChar.getCollectedCoins() + 1);

pChar.addCollectedCoinsPos(newPoint);

}

}

else if(dungeonMap.getTile(newPoint).getType() == TileType.Exit.getValue()) {

if(pChar.getCollectedCoins() > dungeonMap.getCoinWin()) {

pChar.setPosition(newPoint);

Match match = matchList.getMatchForPlayer(pChar.getPlayer().getUsername());

match.setState(MatchState.Over);

Date date = new Date();

match.setTimer(date.getTime() - match.getTimer());

int score = ((int) ((double)pChar.getCollectedCoins() / (double)match.getTimer() \* 10000000));

match.setScore(score);

scoreRepository.insert(new Score(pChar.getPlayer().getUsername(), score));

}

}

return pChar.getPosition();

}

}

* ParseService.java

package com.dod.service.service;

import com.dod.models.Map;

import com.dod.models.Point;

import com.dod.models.Tile;

import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import java.util.Iterator;

/\*\*

\* Implementation of IParseService.

\*/

public class ParseService implements IParseService {

/\*\*

\* Parses a Map object from it's JSON encoding

\* @param input JSONObject a JSON encoding of the Map

\* @return Map an initialised Map parsed from JSON

\* @throws NullPointerException may be thrown by SimpleJson while parsing

\*/

@Override

public Map parseMap(JSONObject input) throws NullPointerException {

JSONObject level = getLevel(input);

JSONArray rowsOfTiles = (JSONArray) level.get("map");

int xSize = ((JSONArray) rowsOfTiles.get(0)).size();

int ySize = rowsOfTiles.size();

Map map = new Map(

(String) level.get("name"),

((Long) (level.get("coin\_num"))).intValue(),

((Long) (level.get("coin\_win"))).intValue(),

((Long) (level.get("Width"))).intValue(),

((Long) (level.get("Height"))).intValue(),

new Point(xSize, ySize));

for (int y = 0; y < ySize; y++) {

JSONArray row = (JSONArray) rowsOfTiles.get(y);

for (int x = 0; x < xSize; x++) {

JSONObject tile = (JSONObject) row.get(x);

map.setTile(new Point(x, y), new Tile(((Long) tile.get("type")).intValue()));

}

}

return map;

}

/\*\*

\* Figures out the level name based on the number of the level and returns the initial element

\* @param input the level numer

\* @return JSONObject of the Map object

\*/

private JSONObject getLevel(JSONObject input) {

Iterator<String> keys = input.keySet().iterator();

String levelKey = keys.hasNext() ? keys.next() : "";

return (JSONObject)input.get(levelKey);

}

}

* StateService.java

package com.dod.service.service;

import com.dod.game.IMatchList;

import com.dod.models.Character;

import com.dod.models.\*;

import com.dod.service.model.CharacterModel;

import com.dod.service.model.GameStateModel;

import com.dod.service.model.TileModel;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Generates a representation of the current game state form the perspective of a particular character

\*/

public class StateService implements IStateService {

IVisibilityService visibilityService;

IMatchList matchList;

public StateService(IVisibilityService visibilityService, IMatchList matchList) {

this.visibilityService = visibilityService;

this.matchList = matchList;

}

/\*\*

\* Generates and returns a representation of the current game state form the perspective of a particular character

\* @param player Player the Player a GameStateModel will be generated for

\* @return GameStateModel a model of the current game state

\*/

@Override

public GameStateModel GetState(Player player) {

Match match = matchList.getMatchForPlayer(player.getUsername());

Map map = visibilityService.createVisibleMap(match.getMap(), match.getCharacter(player.getUsername()));

List<TileModel> tiles = new ArrayList();

List<CharacterModel> characters = new ArrayList();

for(int x = 0; x < map.getWidth(); x++) {

for(int y = 0; y < map.getHeight(); y++) {

Point point = new Point(x,y);

Tile tile = map.getTile(point);

if(tile.isVisible()) {

tiles.add(new TileModel(tile.getType(), point));

List<Character> charactersOnTile = match.getCharactersOnTile(point);

for(Character character : charactersOnTile) {

characters.add(new CharacterModel(

character.getPlayer().getUsername(),

character.getCollectedCoins(),

character.getPosition()));

}

}

}

}

Character character = match.getCharacter(player.getUsername());

return new GameStateModel(tiles.toArray(

new TileModel[tiles.size()]),

characters.toArray(new CharacterModel[characters.size()]),

new CharacterModel(

character.getPlayer().getUsername(),

character.getCollectedCoins(),

character.getPosition()),

match.getState() == MatchState.Over,

match.getMap().getCoinWin());

}

}

* VisibilityService.java

package com.dod.service.service;

import com.dod.models.Map;

import com.dod.models.Character;

import com.dod.models.Point;

import com.dod.models.Tile;

/\*\*

\* Calculates the visible tiles from the perspective of a particular Character

\*/

public class VisibilityService implements IVisibilityService {

/\*\*

\* Generates a copy of a Map with the correct isVisible flags set for the perspective of a particular Character

\* @param deungeonMap the Map pchar resides in

\* @param pchar the Character the perspective of which we're generating visibility with

\* @return a copy of dungeonMap with correct isVisible flags set for the perspective of pchar

\*/

@Override

public Map createVisibleMap(Map dungeonMap, Character pchar) {

Map returnValue = new Map(dungeonMap.getName(), dungeonMap.getCoinNo(), dungeonMap.getCoinWin(), dungeonMap.getWidth(), dungeonMap.getHeight(), new Point(dungeonMap.getWidth(), dungeonMap.getHeight()));

for (int i = 0; i < dungeonMap.getWidth(); i++) {

for (int j = 0; j < dungeonMap.getHeight(); j++) {

if (pchar.getCollectedCoinsPos().contains(new Point(i, j)))

returnValue.setTile(new Point(i, j), new Tile(1, true));

else

returnValue.setTile(new Point(i,j), dungeonMap.getTile(new Point(i, j)));

if (pchar.getPosition().x - 2 > i || pchar.getPosition().x + 2 < i||pchar.getPosition().y -2 > j||pchar.getPosition().y+2 < j)

returnValue.getTile(new Point(i, j)).setVisibility(false);

else

returnValue.getTile(new Point(i, j)).setVisibility(true);

}

}

return returnValue;

}

}

**DungeonOfDooom-master\Sourcecode\project\src\service\src\main\java\com\dod\service**

* Main.java

package com.dod.service;

import com.dod.service.filters.corsFilter;

import org.glassfish.grizzly.http.server.HttpServer;

import org.glassfish.jersey.grizzly2.httpserver.GrizzlyHttpServerFactory;

import org.glassfish.jersey.server.ResourceConfig;

import java.io.IOException;

import java.net.URI;

/\*\*

\* Main class.

\*

\*/

public class Main {

// Base URI the Grizzly HTTP server will listen on

public static final String BASE\_URI = "http://localhost:8080/";

/\*\*

\* Starts Grizzly HTTP server exposing JAX-RS resources defined in this application.

\* @return Grizzly HTTP server.

\*/

public static HttpServer startServer() {

// create a resource config that scans for JAX-RS resources and providers

// in com.dod.service package

final ResourceConfig rc = new ResourceConfig().packages("com.dod.service");

rc.register(new corsFilter());

// create and start a new instance of grizzly http server

// exposing the Jersey application at BASE\_URI

return GrizzlyHttpServerFactory.createHttpServer(URI.create(BASE\_URI), rc);

}

/\*\*

\* Main method.

\* @param args

\* @throws IOException

\*/

public static void main(String[] args) throws IOException {

final HttpServer server = startServer();

System.out.println(String.format("Jersey app started with WADL available at "

+ "%sapplication.wadl\nHit enter to stop it...", BASE\_URI));

System.in.read();

server.stop();

}

}

**DungeonOfDooom-master\Sourcecode\project\src\service**

* pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>dungeon-of-doom</groupId>

<artifactId>dungeon-of-doom-service</artifactId>

<packaging>jar</packaging>

<version>1.0</version>

<name>dungeon-of-doom-service</name>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey</groupId>

<artifactId>jersey-bom</artifactId>

<version>${jersey.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-grizzly2-http</artifactId>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-moxy</artifactId>

</dependency>

<dependency>

<groupId>com.googlecode.json-simple</groupId>

<artifactId>json-simple</artifactId>

<version>1.1.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/commons-codec/commons-codec -->

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.10</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.40</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.ext</groupId>

<artifactId>jersey-bean-validation</artifactId>

<version>2.24.1</version>

</dependency>

<dependency>

<groupId>com.owlike</groupId>

<artifactId>genson</artifactId>

<version>1.4</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>2.5.1</version>

<inherited>true</inherited>

<configuration>

<source>1.7</source>

<target>1.7</target>

</configuration>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>1.2.1</version>

<executions>

<execution>

<goals>

<goal>java</goal>

</goals>

</execution>

</executions>

<configuration>

<mainClass>com.dod.service.Main</mainClass>

</configuration>

</plugin>

</plugins>

</build>

<properties>

<jersey.version>2.24.1</jersey.version>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

</project>

**DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\integration\db**

* DatabaseConnectionTests.java

package dod.test.integration.db;

import com.dod.db.DatabaseConnection;

import org.junit.Assert;

import org.junit.Test;

import java.sql.Connection;

import java.sql.SQLException;

/\*\*

\* Tests database integration

\*/

public class DatabaseConnectionTests {

@Test

public void ShouldConnectToDatabase() {

Connection connection = null;

try {

connection = DatabaseConnection.getConnection();

Assert.assertFalse(connection.isClosed());

}

catch(SQLException e) {

Assert.fail(e.getMessage());

}

DatabaseConnection.Close();

}

@Test

public void ShouldCloseDatabase() {

Connection connection = null;

try {

connection = DatabaseConnection.getConnection();

}

catch(SQLException e) {

Assert.fail(e.getMessage());

}

DatabaseConnection.Close();

try {

Assert.assertTrue(connection.isClosed());

}

catch(SQLException e) {

Assert.fail(e.getMessage());

}

}

}

* DatabaseQueryTests.java

package dod.test.integration.db;

import com.dod.db.repositories.PlayerRepository;

import com.dod.db.repositories.ScoreRepository;

import com.dod.models.Player;

import com.dod.models.Score;

import org.apache.commons.codec.digest.DigestUtils;

import org.junit.Assert;

import org.junit.Test;

import java.sql.SQLException;

/\*\*

\* Unit tests for Database

\*/

public class DatabaseQueryTests {

@Test

public void shouldReturnTrueIfNewPlayerValueIsAddedInDatabase() {

PlayerRepository pr = new PlayerRepository();

String pass = DigestUtils.sha1Hex("1234");

Player pl = new Player("test", pass, new byte[0]);

try {

Assert.assertTrue(pr.insert(pl));

} catch (SQLException e) {

Assert.fail(e.toString());

e.printStackTrace();

}

}

@Test

public void shouldReturnTrueIfPlayerValueExistsInDatabase() {

PlayerRepository pr = new PlayerRepository();

String pass = DigestUtils.sha1Hex("1234");

Player pl = new Player("test", pass, new byte[0]);

try {

Assert.assertTrue(pl.getUsername().equals(pr.get(pl).getUsername()) && pl.getHashedPassword().equals(pr.get(pl).getHashedPassword()));

} catch (SQLException e) {

Assert.fail(e.toString());

e.printStackTrace();

}

}

@Test

public void shouldReturnTrueIfPlayerValueIsDeleted() {

PlayerRepository pr = new PlayerRepository();

String pass = DigestUtils.sha1Hex("1234");

Player pl = new Player("test", pass, new byte[0]);

try {

Assert.assertTrue(pr.delete(pl));

} catch (SQLException e) {

Assert.fail(e.toString());

e.printStackTrace();

}

}

@Test

public void shouldReturnTrueIfNewScoreValueIsAdded() {

ScoreRepository pr = new ScoreRepository();

Player nPlayer = new Player("test", "1234", new byte[0]);

Score temp = new Score(nPlayer.getUsername(), 20);

try {

Assert.assertTrue(pr.insert(temp));

} catch (SQLException e) {

Assert.fail(e.toString());

e.printStackTrace();

}

}

@Test

public void shouldReturnTrueIfScoreValueExistsInDatabase() {

ScoreRepository pr = new ScoreRepository();

Score temp = new Score(1, "test", 20);

try {

Assert.assertTrue(temp.getId() == pr.get(temp).getId() && temp.getValue() == pr.get(temp).getValue() && temp.getUsername().equals(pr.get(temp).getUsername()));

} catch (SQLException e) {

Assert.fail(e.toString());

e.printStackTrace();

}

}

@Test

public void shouldReturnTrueIfScoreValueIsDeleted() {

ScoreRepository pr = new ScoreRepository();

Score temp = new Score(1, "test", 20);

try {

Assert.assertTrue(pr.delete(temp));

} catch (SQLException e) {

Assert.fail(e.toString());

e.printStackTrace();

}

}

}

**DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\integration\service**

* AuthenticatedClientTestBase.java

package dod.test.integration.service;

import com.dod.game.MatchList;

import com.dod.service.Main;

import com.dod.service.model.MatchStatus;

import org.glassfish.grizzly.http.server.HttpServer;

import org.glassfish.jersey.moxy.json.MoxyJsonConfig;

import org.glassfish.jersey.moxy.json.MoxyJsonFeature;

import org.junit.After;

import org.junit.Before;

import javax.ws.rs.client.Client;

import javax.ws.rs.client.ClientBuilder;

import javax.ws.rs.client.Entity;

import javax.ws.rs.client.WebTarget;

import javax.ws.rs.core.MultivaluedHashMap;

import javax.ws.rs.core.MultivaluedMap;

import javax.ws.rs.core.Response;

import javax.ws.rs.ext.ContextResolver;

import java.util.\*;

import static org.junit.Assert.assertEquals;

/\*\*

\* A base class for testing endpoints with sessions

\*/

public class AuthenticatedClientTestBase {

protected WebTarget target;

protected String testUsername;

protected String sessionId;

protected List<UUID> matchesToRemove;

private HttpServer server;

@Before

public void setUp() {

server = Main.startServer();

//Setup JSON client

Map<String, String> namespacePrefixMapper = new HashMap<String, String>();

namespacePrefixMapper.put("http://www.w3.org/2001/XMLSchema-instance", "xsi");

MoxyJsonConfig moxyJsonConfig = new MoxyJsonConfig()

.setNamespacePrefixMapper(namespacePrefixMapper)

.setNamespaceSeparator(':');

final ContextResolver<MoxyJsonConfig> jsonConfigResolver = moxyJsonConfig.resolver();

Client c = ClientBuilder.newBuilder()

.register(MoxyJsonFeature.class)

.register(jsonConfigResolver)

.build();

target = c.target(Main.BASE\_URI);

//Generate random user/pass for testing

testUsername = UUID.randomUUID().toString();

//Register user/pass so we have a guarunteed user that exists

sessionId = registerUserAndGetSessionId(testUsername);

//For cleaning up the static MatchList

matchesToRemove = new ArrayList();

}

@After

public void tearDown() throws Exception {

server.stop();

//Cleanup static data before next turn

for(UUID id : matchesToRemove) {

MatchList.instance().removeMatch(id);

}

}

protected String registerUserAndGetSessionId(String identifier) {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", identifier);

formData.add("password", identifier);

Response registerResponse = target.path("player/register").request().post(Entity.form(formData));

//get the sessionId so we can send authorised session cookies with requests

return registerResponse.getCookies().get("JSESSIONID").getValue();

}

protected MatchStatus startNewMatch() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("level", "1");

javax.ws.rs.client.Invocation.Builder request = target.path("match/new").request();

request.cookie("JSESSIONID",sessionId);

Response result = request.post(Entity.form(formData));

assertEquals(200, result.getStatus());

return result.readEntity(MatchStatus.class);

}

}

* GameControllerTests.java

package dod.test.integration.service;

import com.dod.game.MatchList;

import com.dod.models.Player;

import com.dod.models.Point;

import com.dod.service.model.GameStateModel;

import com.dod.service.model.MatchStatus;

import org.junit.Assert;

import org.junit.Test;

import javax.ws.rs.client.Invocation;

import javax.ws.rs.core.Response;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertNotNull;

/\*\*

\* Tests the GameController

\*/

public class GameControllerTests extends AuthenticatedClientTestBase {

@Test

public void shouldRespondToStatus() {

MatchStatus matchStatus = startNewMatch();

matchesToRemove.add(matchStatus.getId());

Invocation.Builder request = target.path("game/status").request();

request.cookie("JSESSIONID",sessionId);

Response response = request.buildGet().invoke();

Assert.assertEquals(200, response.getStatus());

GameStateModel result = response.readEntity(GameStateModel.class);

assertNotNull(result);

assertEquals(1, result.getCharacters().length);

assertEquals(468, result.getTiles().length);

assertNotNull(result.getTiles()[0].getPosition());

}

@Test

public void shouldRespondToMove() {

String responseMsg = target.path("game/move").request().post(null).readEntity(String.class);

assertEquals("unimplemented", responseMsg);

}

}

* MatchControllerTests.java

package dod.test.integration.service;

import com.dod.game.MatchList;

import com.dod.models.Match;

import com.dod.models.MatchState;

import com.dod.models.Player;

import com.dod.models.Point;

import com.dod.service.model.MatchStatus;

import org.junit.Assert;

import org.junit.Test;

import javax.ws.rs.client.\*;

import javax.ws.rs.core.MultivaluedHashMap;

import javax.ws.rs.core.MultivaluedMap;

import javax.ws.rs.core.Response;

import java.util.\*;

import static junit.framework.Assert.assertNotNull;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertNull;

/\*\*

\* Tests for MatchController

\* !NOTE! : As of right now these tests will ONLY work if you add a Symboolic Link directory (mklink /d in Win)

\* to the git root pointing to the Assets folder

\* This is becaues of project config issues... It's a crap solution I know.

\* Potential future solutions:

\* Add a run parameter that can override the assets folder path

\* Place a static variable somewhere that can be overridden by the Tests project, to hold the assets folder path.

\* Find a way to pass a variable into the HttpServer object that can be fed to the IOService

\*/

public class MatchControllerTests extends AuthenticatedClientTestBase {

@Test

public void shouldGiveCurrentMatchStatus() {

MatchStatus matchStatus = startNewMatch();

Invocation.Builder request = target.path("match/status").request();

request.cookie("JSESSIONID",sessionId);

MatchStatus response = request.get(MatchStatus.class);

assertNotNull(response);

matchesToRemove.add(matchStatus.getId());

assertEquals(matchStatus.getId(), response.getId());

assertEquals(testUsername, response.getPlayerNames()[0]);

}

@Test

public void whenPlayerHasNoOngoingMatchStatusShouldReturnNull() {

Invocation.Builder request = target.path("match/status").request();

request.cookie("JSESSIONID",sessionId);

MatchStatus response = request.get(MatchStatus.class);

assertNull(response);

}

@Test

public void shouldCreateNewMatch() {

MatchStatus matchStatus = startNewMatch();

matchesToRemove.add(matchStatus.getId());

assertNotNull(matchStatus.getId());

matchesToRemove.add(matchStatus.getId());

assertEquals(testUsername, matchStatus.getPlayerNames()[0]);

assertNotNull(MatchList.instance().getMatch(matchStatus.getId()));

}

@Test

public void shouldStartMatch() {

MatchStatus matchStatus = startNewMatch();

matchesToRemove.add(matchStatus.getId());

Invocation.Builder request = target.path("match/start").request();

request.cookie("JSESSIONID",sessionId);

Response result = request.post(null);

assertEquals(200, result.getStatus());

assertEquals(MatchState.Ingame, MatchList.instance().getMatch(matchStatus.getId()).getState());

}

@Test

public void joinShouldAddUserToMatch() {

//Add a match with the original test user

MatchStatus matchStatus = startNewMatch();

matchesToRemove.add(matchStatus.getId());

//Register another user that isn't already a member of the new match

String newTestUsername = UUID.randomUUID().toString();

String newUserSession = registerUserAndGetSessionId(newTestUsername);

Invocation.Builder request = target.path("match/join").request();

request.cookie("JSESSIONID",newUserSession);

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("matchId", matchStatus.getId().toString());

Response response = request.post(Entity.form(formData));

assertEquals(200, response.getStatus());

Assert.assertTrue(MatchList.instance().getMatch(matchStatus.getId()).hasCharacter(newTestUsername));

MatchStatus result = response.readEntity(MatchStatus.class);

assertNotNull(result);

assertEquals(matchStatus.getId(), result.getId());

}

//todo improve this test so that it doesn't break other tests if it fails

@Test

public void listShouldListAllLobbyingMatches() {

MatchList.instance().addMatch(new Match(null));

MatchList.instance().addMatch(new Match(null));

MatchList.instance().addMatch(new Match(null));

Invocation.Builder request = target.path("match/list").request();

request.cookie("JSESSIONID",sessionId);

Response result = request.get();

MatchStatus[] response = result.readEntity(MatchStatus[].class);

assertEquals(200, result.getStatus());

assertEquals(3, response.length);

matchesToRemove.add(response[0].getId());

matchesToRemove.add(response[1].getId());

matchesToRemove.add(response[2].getId());

}

@Test

public void leaveShouldRemovePlayerFromMatch() {

MatchStatus matchStatus = startNewMatch();

matchesToRemove.add(matchStatus.getId());

Invocation.Builder request = target.path("match/leave").request();

request.cookie("JSESSIONID",sessionId);

Response result = request.post(null);

assertEquals(200, result.getStatus());

assertEquals(false, MatchList.instance().getMatch(matchStatus.getId()).hasCharacter(testUsername));

}

}

* MyResourceTest.java

package dod.test.integration.service;

import javax.ws.rs.client.Client;

import javax.ws.rs.client.ClientBuilder;

import javax.ws.rs.client.WebTarget;

import com.dod.service.Main;

import org.glassfish.grizzly.http.server.HttpServer;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class MyResourceTest {

private HttpServer server;

private WebTarget target;

@Before

public void setUp() throws Exception {

// start the server

server = Main.startServer();

// create the client

Client c = ClientBuilder.newClient();

// uncomment the following line if you want to enable

// support for JSON in the client (you also have to uncomment

// dependency on jersey-media-json module in pom.xml and Main.startServer())

// --

// c.configuration().enable(new org.glassfish.jersey.media.json.JsonJaxbFeature());

target = c.target(Main.BASE\_URI);

}

@After

public void tearDown() throws Exception {

server.stop();

}

/\*\*

\* Test to see that the message "Got it!" is sent in the response.

\*/

@Test

public void testGetIt() {

String responseMsg = target.path("myresource").request().get(String.class);

assertEquals("Got it!", responseMsg);

}

}

* PlayerControllerTests.java

package dod.test.integration.service;

import com.dod.db.repositories.PlayerRepository;

import com.dod.models.Player;

import com.dod.service.Main;

import org.glassfish.grizzly.http.server.HttpServer;

import org.junit.After;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import javax.ws.rs.client.Client;

import javax.ws.rs.client.ClientBuilder;

import javax.ws.rs.client.Entity;

import javax.ws.rs.client.WebTarget;

import javax.ws.rs.core.MultivaluedHashMap;

import javax.ws.rs.core.MultivaluedMap;

import javax.ws.rs.core.Response;

import static org.junit.Assert.assertEquals;

import static org.junit.Assert.assertNotNull;

/\*\*

\* Tests the PlayerController

\*/

public class PlayerControllerTests {

private HttpServer server;

private WebTarget target;

private PlayerRepository repository;

private final String testUsername = "testUsername";

private final String testNonExistantusername = "testNonexistantUsername";

private final String testPassword = "testPassword";

@Before

public void setUp() {

server = Main.startServer();

Client c = ClientBuilder.newClient();

repository = new PlayerRepository();

target = c.target(Main.BASE\_URI);

}

@After

public void tearDown() throws Exception {

server.stop();

try {

repository.delete(new Player(testUsername));

}

catch(Exception e) {

e.printStackTrace();

}

}

@Test

public void whenDetailsAreValidShouldRegisterPlayer() throws Exception {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", testUsername);

formData.add("password", testPassword);

Response response = target.path("player/register")

.request()

.post(Entity.form(formData));

assertEquals("", response.readEntity(String.class));

assertEquals(200, response.getStatus());

assertNotNull(repository.get(new Player(testUsername)));

}

@Test

public void whenUsernameEmptyRegisterShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", "");

formData.add("password", testPassword);

Response response = target.path("player/register")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

@Test

public void whenPasswordEmptyRegisterShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", testUsername);

formData.add("password", "");

Response response = target.path("player/register")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

@Test

public void whenPasswordTooLongRegisterShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", testUsername);

formData.add("password", generateStringOfSize(257));

Response response = target.path("player/register")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

@Test

public void whenUsernameTooLongRegisterShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", generateStringOfSize(256));

formData.add("password", testPassword);

Response response = target.path("player/register")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

@Test

public void whenUsernameAlreadyTakenRegisterShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", testUsername);

formData.add("password", testPassword);

Response response = target.path("player/register")

.request()

.post(Entity.form(formData));

assertEquals(200, response.getStatus());

response = target.path("player/register")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

@Test

public void whenDetailsValidLoginShouldReturnBlankOkStatus() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", testUsername);

formData.add("password", testPassword);

//Create player before trying to login

Response response = target.path("player/register")

.request()

.post(Entity.form(formData));

assertEquals(200, response.getStatus());

response = target.path("player/login")

.request()

.post(Entity.form(formData));

Assert.assertEquals(200, response.getStatus());

assertEquals("", response.readEntity(String.class));

}

@Test

public void whenUsernameEmptyLoginShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", "");

formData.add("password", testPassword);

Response response = target.path("player/login")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

@Test

public void whenPasswordEmptyLoginShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", testUsername);

formData.add("password", "");

Response response = target.path("player/login")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

@Test

public void whenPasswordTooLongLoginShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", testUsername);

formData.add("password", generateStringOfSize(256));

Response response = target.path("player/login")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

@Test

public void whenUsernameTooLongLoginShouldReturnValidationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", generateStringOfSize(256));

formData.add("password", testPassword);

Response response = target.path("player/login")

.request()

.post(Entity.form(formData));

assertEquals(400, response.getStatus());

}

/\*\*

\* We don't want to return validation here- we don't want to inform a malicious user

\* when they do or don't randomly guess a correct username

\*/

@Test

public void whenUsernameDoesNotExistLoginShouldReturnBlankAuthorisationError() {

MultivaluedMap<String, String> formData = new MultivaluedHashMap<String, String>();

formData.add("username", testNonExistantusername);

formData.add("password", testPassword);

Response response = target.path("player/login")

.request()

.post(Entity.form(formData));

assertEquals(403, response.getStatus());

assertEquals("", response.readEntity(String.class));

}

private String generateStringOfSize(int size) {

String result = "";

for(int i = 0; i < size; i++) {

result += "z";

}

return result;

}

}

**DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\unit\domain\game**

* MatchListTests.java

package dod.test.unit.domain.game;

import com.dod.game.MatchList;

import com.dod.models.Match;

import com.dod.models.MatchState;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import java.util.List;

import java.util.UUID;

import static org.mockito.Mockito.\*;

/\*\*

\* Tests for MatchList

\*/

public class MatchListTests {

private final String testUsername = "testUsername";

private MatchList matchList;

@Before

public void Setup() {

matchList = new MatchList();

}

@Test

public void shouldGetLobbyingMatches() {

Match lobbyingMatch = mock(Match.class);

Match ingameMatch = mock(Match.class);

Match anotherIngameMatch = mock(Match.class);

when(lobbyingMatch.getState()).thenReturn(MatchState.Lobbying);

when(ingameMatch.getState()).thenReturn(MatchState.Ingame);

when(anotherIngameMatch.getState()).thenReturn(MatchState.Ingame);

matchList.addMatch(lobbyingMatch);

matchList.addMatch(ingameMatch);

matchList.addMatch(anotherIngameMatch);

List<Match> result = matchList.getLobbyingMatches();

Assert.assertEquals(1, result.size());

Assert.assertEquals(lobbyingMatch, result.get(0));

}

@Test

public void shouldGetMatchById() {

Match matchOne = mock(Match.class);

Match matchTwo = mock(Match.class);

UUID idOne = UUID.randomUUID();

UUID idTwo = UUID.randomUUID();

when(matchOne.getId()).thenReturn(idOne);

when(matchTwo.getId()).thenReturn(idTwo);

matchList.addMatch(matchOne);

matchList.addMatch(matchTwo);

Match result = matchList.getMatch(idOne);

Assert.assertEquals(matchOne, result);

}

@Test

public void shouldGetMatchForPlayer() {

Match match = mock(Match.class);

when(match.hasCharacter(testUsername)).thenReturn(true);

matchList.addMatch(match);

Match result = matchList.getMatchForPlayer(testUsername);

Assert.assertEquals(match,result);

}

}

**DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\unit\domain\model**

* MatchTests.java

package dod.test.unit.domain.model;

import com.dod.models.\*;

import com.dod.models.Character;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import static org.mockito.Mockito.\*;

/\*\*

\* Tests some of the non-trivial Match functions

\*/

public class MatchTests {

Map map;

Match match;

Player player;

private final String testUsername = "testUsername";

@Before

public void Setup() {

map = mock(Map.class);

match = new Match(map);

player = mock(Player.class);

when(player.getUsername()).thenReturn(testUsername);

}

@Test

public void shouldAddCharacter() {

match.addCharacter(player, new Point(0,0));

Assert.assertTrue(match.hasCharacter(testUsername));

}

@Test

public void whenThereAreMultipleCharactersShouldGetCorrectCharacter() {

Player anotherPlayer = mock(Player.class);

Player anotherAnotherPlayer = mock(Player.class);

when(anotherPlayer.getUsername()).thenReturn("anotherTestUsername");

when(anotherAnotherPlayer.getUsername()).thenReturn("anotherAnotherTestUsername");

match.addCharacter(player, new Point(0,0));

match.addCharacter(anotherPlayer, new Point(0,0));

match.addCharacter(anotherAnotherPlayer, new Point(0,0));

Assert.assertEquals(player, match.getCharacter(testUsername).getPlayer());

}

}

**DungeonOfDooom-master\Sourcecode\project\src\tests\com\dod\test\unit\service**

* AuthenticationServiceTests.java

package dod.test.unit.service;

import com.dod.db.repositories.IPlayerRepository;

import com.dod.models.Player;

import com.dod.service.model.LoginModel;

import com.dod.service.service.AuthenticationService;

import com.dod.service.service.IAuthenticationService;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import static org.mockito.Mockito.\*;

/\*\*

\* Tests for the AuthenticationService

\*/

public class AuthenticationServiceTests {

private IAuthenticationService service;

private IPlayerRepository repository;

private final String testPlayername = "test";

private final String testPassword = "testPassword";

private final String incorrectTestPassword = "incorrectTestPassword";

//These two are calculated by the hashing algorithm from testPassword so should always work

private final byte[] testSalt = new byte[] {-77,14,44,-103,-37,0,60,-41,54,60,-24,-69,-10,-14,101,-17,101,

95,16,50,60,81,34,-90,-85,123,88,88,-18,71,80,93};

private final String testHashedPassword =

"sw4smdsAPNc2POi79vJl72VfEDI8USKmq3tYWO5HUF0=vNgKzsYRou5lhm4l8i7pFYsYeqeicv/5O5KeplB2rLY=";

@Before

public void Setup() throws Exception {

repository = mock(IPlayerRepository.class);

service = new AuthenticationService(repository);

}

@Test

public void whenUsernameDoesNotExistRegisterShouldCreatePlayerAndReturnTrue() throws Exception {

when(repository.get(any(Player.class))).thenReturn(null);

boolean result = service.Register(new LoginModel(testPlayername, testPassword));

verify(repository).insert(any(Player.class));

Assert.assertEquals(true, result);

}

@Test

public void whenUsernameDoesExistRegisterShouldReturnFalse() throws Exception {

when(repository.get(any(Player.class))).thenReturn(new Player(testPlayername, testPassword, new byte[0]));

boolean result = service.Register(new LoginModel(testPlayername, testPassword));

Assert.assertEquals(false, result);

}

@Test

public void whenDetailsAreValidLoginShouldReturnTrue() throws Exception {

when(repository.get(any(Player.class))).thenReturn(new Player(testPlayername, testHashedPassword, testSalt));

boolean result = service.Login(new LoginModel(testPlayername, testPassword));

Assert.assertEquals(true, result);

}

@Test

public void whenPlayerDoesNotExistLoginShouldReturnFalse() throws Exception {

when(repository.get(any(Player.class))).thenReturn(null);

boolean result = service.Login(new LoginModel(testPlayername, testPassword));

Assert.assertEquals(false, result);

}

@Test

public void whenPasswordIsWrongLoginShouldReturnFalse() throws Exception {

when(repository.get(any(Player.class))).thenReturn(new Player(testPlayername, testHashedPassword, testSalt));

boolean result = service.Login(new LoginModel(testPlayername, incorrectTestPassword));

Assert.assertEquals(false, result);

}

}

* IOServiceTests.java

package dod.test.unit.service;

import com.dod.service.constant.Assets;

import com.dod.service.service.IIOService;

import com.dod.service.service.IOService;

import org.json.simple.JSONObject;

import org.json.simple.parser.ParseException;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import java.io.IOException;

/\*\*

\* Unit tests for the IOService

\*/

public class IOServiceTests {

IIOService service;

private String testAssetPath = "\\test\\test.asset";

private String expectedTestAssetResult = "testasset :)";

private String nonExistantTestAssetPath = "nonexistant.asset";

private String testJsonPath = "\\test\\test.json";

@Before

public void Setup() {

service = new IOService(".\\assets");

}

@Test

public void shouldGetAssetAtPath() {

try {

String result = service.getString(testAssetPath);

Assert.assertEquals(expectedTestAssetResult, result);

}

catch(IOException e) {

Assert.fail("Unexepected exception thrown by service:" + e.toString());

e.printStackTrace();

}

}

@Test

public void whenPathIsInvalidShouldThrowException() {

try {

String result = service.getString(nonExistantTestAssetPath);

Assert.fail("Service did not throw exception when expected.");

}

catch(IOException e) {

//Pass!

}

}

@Test

public void shouldParseJsonFile() {

try {

JSONObject result = service.getJsonObject(testJsonPath);

Assert.assertTrue(result.containsKey("id"));

}

catch(Exception e) {

Assert.fail("Unexepected exception thrown by service:" + e.toString());

e.printStackTrace();

}

}

@Test

public void whenJsonIsInvalidShouldThrownParseException() {

try {

JSONObject result = service.getJsonObject(testAssetPath);

Assert.fail("Service did not throw exception when expected.");

}

catch(ParseException e) {

//Pass!

}

catch(Exception e) {

Assert.fail("Unexepected exception thrown by service:" + e.toString());

e.printStackTrace();

}

}

}

* MatchServiceTests.java

package dod.test.unit.service;

import com.dod.db.repositories.IPlayerRepository;

import com.dod.game.IMatchList;

import com.dod.game.MatchList;

import com.dod.models.\*;

import com.dod.service.model.MatchStatus;

import com.dod.service.service.IIOService;

import com.dod.service.service.IOService;

import com.dod.service.service.IParseService;

import com.dod.service.service.MatchService;

import org.json.simple.JSONObject;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import org.mockito.Mock;

import org.mockito.Mockito;

import java.sql.SQLException;

import java.util.UUID;

import static org.junit.Assert.fail;

import static org.mockito.Mockito.\*;

/\*\*

\* Tests for MatchService

\*/

public class MatchServiceTests {

MatchService service;

IIOService ioServiceMock;

IParseService parseServiceMock;

IPlayerRepository playerRepositoryMock;

IMatchList matchListSpy;

Map mapMock;

Player playerMock;

private final int testLevelNo = 0;

private final String testLevelPath = "/maps/level0.json";

private final String testUsername = "testUsername";

private final Point testPoint = new Point(0,0);

private final int testNumberOfCoins = 10;

@Before

public void setup() {

ioServiceMock = mock(IOService.class);

parseServiceMock = mock(IParseService.class);

playerRepositoryMock = mock(IPlayerRepository.class);

matchListSpy = spy(new MatchList());

mapMock = mock(Map.class);

playerMock = mock(Player.class);

when(playerMock.getUsername()).thenReturn(testUsername);

service = new MatchService(ioServiceMock, parseServiceMock, playerRepositoryMock, matchListSpy);

}

@Test

public void shouldCreateMatch() throws Exception {

when(ioServiceMock.getJsonObject(any(String.class))).thenReturn(new JSONObject());

when(parseServiceMock.parseMap(any(JSONObject.class))).thenReturn(mapMock);

when(playerRepositoryMock.get(any(Player.class))).thenReturn(new Player(testUsername));

when(mapMock.getRandomFreeTilePoint()).thenReturn(testPoint);

when(mapMock.getCoinNo()).thenReturn(testNumberOfCoins);

when(mapMock.getTile(any(Point.class))).thenReturn(new Tile(0));

MatchStatus result = service.createMatch(testUsername,testLevelNo);

verify(matchListSpy).addMatch(any(Match.class));

Assert.assertTrue(matchListSpy.playerHasMatch(testUsername));

Assert.assertEquals(result.getId(), matchListSpy.getMatchForPlayer(testUsername).getId());

Assert.assertEquals(testPoint,

matchListSpy.getMatchForPlayer(testUsername).getCharacter(testUsername).getPosition());

}

@Test

public void WhenCreatingMatchShouldAssignRandomCharacterAndCoinPositions() throws Exception {

when(ioServiceMock.getJsonObject(any(String.class))).thenReturn(new JSONObject());

when(parseServiceMock.parseMap(any(JSONObject.class))).thenReturn(mapMock);

when(playerRepositoryMock.get(any(Player.class))).thenReturn(new Player(testUsername));

when(mapMock.getRandomFreeTilePoint()).thenReturn(testPoint);

when(mapMock.getCoinNo()).thenReturn(testNumberOfCoins);

when(mapMock.getTile(any(Point.class))).thenReturn(new Tile(0));

MatchStatus result = service.createMatch(testUsername,testLevelNo);

verify(mapMock, times(testNumberOfCoins + 1)).getRandomFreeTilePoint();

Assert.assertEquals(testPoint,

matchListSpy.getMatchForPlayer(testUsername).getCharacter(testUsername).getPosition());

}

@Test

public void shouldStartMatch() {

Match matchSpy = spy(new Match(null));

matchListSpy.addMatch(matchSpy);

matchSpy.addCharacter(playerMock, testPoint);

service.startMatch(playerMock);

verify(matchListSpy, times(1)).getMatchForPlayer(testUsername);

verify(matchSpy, times(1)).setState(MatchState.Ingame);

Assert.assertEquals(MatchState.Ingame, matchSpy.getState());

}

@Test

public void shouldGetMatchStatus() {

Match matchSpy = spy(new Match(null));

matchSpy.addCharacter(playerMock, testPoint);

matchListSpy.addMatch(matchSpy);

MatchStatus result = service.getStatus(playerMock);

verify(playerMock, atLeastOnce()).getUsername();

verify(matchListSpy, times(1)).playerHasMatch(testUsername);

Assert.assertEquals(matchSpy.getId(), result.getId());

}

@Test

public void whenPlayerHasNoMatchGetStatusShouldReturnNull() {

MatchStatus result = service.getStatus(playerMock);

Assert.assertNull(result);

}

@Test

public void leaveMatchShouldRemoveCharacterFromMatch() {

Match matchSpy = spy(new Match(null));

matchSpy.addCharacter(playerMock, testPoint);

matchListSpy.addMatch(matchSpy);

service.leaveMatch(playerMock);

Assert.assertNull(matchSpy.getCharacter(testUsername));

}

@Test

public void endMatchShouldRemoveMatchFromMatchList() {

Match matchSpy = spy(new Match(null));

matchSpy.addCharacter(playerMock, testPoint);

matchListSpy.addMatch(matchSpy);

service.endMatch(playerMock);

verify(matchListSpy, times(1)).removeMatch(matchSpy.getId());

Assert.assertNull(matchListSpy.getMatchForPlayer(testUsername));

}

@Test

public void joinMatchShoulAddPlayerToMatch() throws Exception {

when(mapMock.getRandomFreeTilePoint()).thenReturn(testPoint);

when(playerRepositoryMock.get(any(Player.class))).thenReturn(playerMock);

Match matchSpy = spy(new Match(mapMock));

matchListSpy.addMatch(matchSpy);

service.joinMatch(playerMock, matchSpy.getId());

Assert.assertTrue(matchSpy.hasCharacter(testUsername));

}

@Test

public void whenSqlExceptionoccursJoinMatchShouldThrowException() throws Exception {

when(mapMock.getRandomFreeTilePoint()).thenReturn(testPoint);

when(playerRepositoryMock.get(any(Player.class))).thenThrow(new SQLException());

Match matchSpy = spy(new Match(mapMock));

matchListSpy.addMatch(matchSpy);

try {

service.joinMatch(playerMock, matchSpy.getId());

fail();

}

catch(SQLException e) {

//success!

}

catch(Exception e) {

fail();

}

}

@Test

public void getLobbyingMatchesShouldOnlyReturnMatchesInLobbyState() {

Match lobbyingMatchMock = mock(Match.class);

when(lobbyingMatchMock.getState()).thenReturn(MatchState.Lobbying);

Match inGameMatchMock = mock(Match.class);

when(inGameMatchMock.getState()).thenReturn(MatchState.Ingame);

UUID testId = UUID.randomUUID();

when(lobbyingMatchMock.getId()).thenReturn(testId);

matchListSpy.addMatch(lobbyingMatchMock);

matchListSpy.addMatch(inGameMatchMock);

MatchStatus[] result = service.getLobbyingMatches();

Assert.assertEquals(1, result.length);

Assert.assertEquals(testId, result[0].getId());

}

@Test

public void whenNoMatchesInLobbyStateGetLobbyingMatchesShouldReturnEmptyArray() {

MatchStatus[] result = service.getLobbyingMatches();

Assert.assertNotNull(result);

Assert.assertEquals(0, result.length);

}

}

* MovementTests.java

package dod.test.unit.service;

import com.dod.models.Character;

import com.dod.models.Map;

import com.dod.models.Player;

import com.dod.models.Point;

import com.dod.service.service.IOService;

import com.dod.service.service.MovementService;

import com.dod.service.service.ParseService;

import org.json.simple.JSONObject;

import org.json.simple.parser.ParseException;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import java.io.IOException;

/\*\*

\* Created by tasos on 11/12/2016.

\*/

public class MovementTests {

private IOService ioService;

private ParseService parService;

private Map dungeonMap;

private Character pChar, pChar2;

private JSONObject jobject;

@Before

public void Setup() {

ioService = new IOService(".\\assets");

try {

jobject = ioService.getJsonObject("\\maps\\Level1.json");

} catch (IOException e) {

e.printStackTrace();

} catch (ParseException e) {

e.printStackTrace();

}

parService = new ParseService();

dungeonMap = parService.parseMap(jobject);

pChar = new Character(new Point(4, 4), new Player("test"));

pChar2 = new Character(new Point(3, 1), new Player("dadasda"));

}

@Test

public void shouldReturnTrueIfPlayerMovedToRightTile() throws Exception {

MovementService moveService = new MovementService();

Assert.assertTrue(moveService.Move("D", new Player("test")).equals(new Point(5,4)));

}

@Test

public void shouldReturnFalseIfPlayerMovedToRightTile() throws Exception {

MovementService moveService = new MovementService();

Assert.assertFalse(moveService.Move("D", new Player("test")).equals(new Point(4,4)));

}

@Test

public void shouldReturnFalseIfPlayerMovesToWall() throws Exception {

MovementService moveService = new MovementService();

Assert.assertFalse(moveService.Move("D", new Player("test")).equals(new Point(3,0)));

}

@Test

public void shouldReturnTrueIfPlayerCantMoveToWall() throws Exception {

MovementService moveService = new MovementService();

Assert.assertFalse(moveService.Move("D", new Player("test")).equals(new Point(3,1)));

}

}

* ParseServiceTests.java

package dod.test.unit.service;

import com.dod.models.Map;

import com.dod.models.Point;

import com.dod.service.service.IParseService;

import com.dod.service.service.ParseService;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;

import org.json.simple.parser.ParseException;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

/\*\*

\* Tests for the ParseService

\*/

public class ParseServiceTests {

private IParseService service;

private JSONParser parser;

private String validJson =

"{ \"testLev\": { \"id\":\"test\", \"name\":\"test\",\"coin\_num\":6, \"coin\_win\":5, \"Width\" : 26, \"Height\" : 18, \"tiles\": [" +

"{\"id\": \"tile\_wall\",\"name\":\"wall\",\"type\":0,\"visibility\":true,\"touchable\":false}," +

"{\"id\": \"tile\_path\",\"name\":\"path\",\"type\":1,\"visibility\":true,\"touchable\":true}," +

"{\"id\": \"tile\_path2\",\"name\":\"path2\",\"type\":2,\"visibility\":true,\"touchable\":true}" +

"], \"map\":[ [{\"type\":0},{\"type\":0},{\"type\":0},{\"type\":0}], " +

"[{\"type\":0},{\"type\":0},{\"type\":0},{\"type\":0}]] } }";

private String invalidJson =

"{ \"testLev\": { \"id\":\"test\", \"tiles\": [" +

"{\"id\": \"tile\_wall\",\"name\":\"wall\",\"type\":0,\"visibility\":true,\"touchable\":false}," +

"{\"id\": \"tile\_path\",\"name\":\"path\",\"type\":1,\"visibility\":true,\"touchable\":true}," +

"{\"id\": \"tile\_path2\",\"name\":\"path2\",\"type\":2,\"visibility\":true,\"touchable\":true}" +

"], \"map\":[ [{\"type\":0},{\"type\":0},{\"type\":0},{\"type\":0}], " +

"[{\"type\":0},{\"type\":0},{\"type\":0},{\"type\":0}]] } }";

@Before

public void Setup() {

service = new ParseService();

parser = new JSONParser();

}

@Test

public void shouldGenerateMapFromJson() {

try {

JSONObject input = (JSONObject) parser.parse(validJson);

Map result = service.parseMap(input);

Assert.assertEquals(5, result.getCoinWin());

Assert.assertEquals(6, result.getCoinNo());

Assert.assertEquals("test", result.getName());

for(int x = 0; x < 4; x++) {

for(int y = 0; y < 2; y++) {

Assert.assertEquals(0, result.getTile(new Point(x,y)).getType());

}

}

}

catch(Exception e) {

e.printStackTrace();

Assert.fail("Unexepected exception thrown by service:" + e.toString());

}

}

@Test

public void whenJsonIsInvalidShouldThrowException() {

try {

JSONObject input = (JSONObject) parser.parse(invalidJson);

Map result = service.parseMap(input);

Assert.fail("Test did not throw expected exception.");

}

catch(NullPointerException e) {

//Passed!

}

catch(Exception e) {

Assert.fail("Unexepected exception thrown by service:" + e.toString());

e.printStackTrace();

}

}

}

* StateServiceTests.java

package dod.test.unit.service;

import com.dod.game.MatchList;

import com.dod.models.\*;

import com.dod.service.model.GameStateModel;

import com.dod.service.service.IOService;

import com.dod.service.service.IVisibilityService;

import com.dod.service.service.ParseService;

import com.dod.service.service.StateService;

import org.json.simple.JSONObject;

import org.json.simple.parser.ParseException;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import java.io.IOException;

import static org.junit.Assert.fail;

import static org.mockito.Mockito.\*;

/\*\*

\* Tests for the StateService.

\*/

public class StateServiceTests {

private IVisibilityService visibilityServiceMock;

private MatchList matchListMock;

private StateService stateService;

private Map map;

private Point testPoint;

private Match match;

private String testUsername = "testUsername";

@Before

public void Setup() {

visibilityServiceMock = mock(IVisibilityService.class);

matchListMock = mock(MatchList.class);

stateService = new StateService(visibilityServiceMock, matchListMock);

IOService ioService = new IOService();

JSONObject jobject = null;

try {

jobject = ioService.getJsonObject("\\maps\\level1.json");

} catch (IOException e) {

e.printStackTrace();

} catch (ParseException e) {

e.printStackTrace();

}

ParseService parService = new ParseService();

map = parService.parseMap(jobject);

match = new Match(map);

testPoint = map.getRandomFreeTilePoint();

match.addCharacter(new Player(testUsername), testPoint);

when(matchListMock.getMatchForPlayer(testUsername)).thenReturn(match);

}

// @Test

// public void shouldGetCurrentStateOfGame() {

// GameStateModel result = stateService.GetState(new Player(testUsername));

// when(visibilityServiceMock.getVisibleTilesForCharacter(map, match.getCharacter(testUsername))).thenReturn(map);

//

// Assert.assertEquals(testPoint, result.getCharacters()[0].getPosition());

// Assert.assertEquals(map.getWidth() \* map.getHeight(), result.getTiles().length);

// Assert.assertEquals(1, result.getCharacters().length);

// }

//

// @Test

// public void shouldOnlyReturnVisibleTiles() {

// when(visibilityServiceMock.getVisibleTilesForCharacter(map, match.getCharacter(testUsername))).thenReturn(null);

// //todo

// fail();

// }

}

* VisibilityServiceTest.java

package dod.test.unit.service;

import com.dod.models.Map;

import com.dod.models.Player;

import com.dod.models.Point;

import com.dod.models.Character;

import com.dod.service.service.IOService;

import com.dod.service.service.ParseService;

import com.dod.service.service.VisibilityService;

import org.json.simple.JSONObject;

import org.json.simple.parser.ParseException;

import org.junit.Assert;

import org.junit.Before;

import org.junit.Test;

import java.io.IOException;

/\*\*

\* Created by tasos on 7/12/2016.

\*/

public class VisibilityServiceTest {

private IOService ioService;

private ParseService parService;

private Map dungeonMap;

private JSONObject jobject;

private Character pChar;

private Map visibleMap;

@Before

public void Setup() {

ioService = new IOService(".\\assets");

visibleMap = new Map("level1", 30, 20, 26, 18, new Point(26, 18));

try {

jobject = ioService.getJsonObject("\\maps\\Level1.json");

} catch (IOException e) {

e.printStackTrace();

} catch (ParseException e) {

e.printStackTrace();

}

parService = new ParseService();

dungeonMap = parService.parseMap(jobject);

pChar = new Character(new Point(4, 4), new Player("test"));

}

@Test

public void shouldReturnTrueIfTheTile34IsVisible() {

VisibilityService vService = new VisibilityService();

visibleMap = vService.createVisibleMap(dungeonMap, pChar);

Assert.assertTrue(visibleMap.getTile(new Point(3, 4)).isVisible());

}

@Test

public void shouldReturnFalseIfTheTile77IsNotVisible() {

VisibilityService vService = new VisibilityService();

visibleMap = vService.createVisibleMap(dungeonMap, pChar);

Assert.assertFalse(visibleMap.getTile(new Point(7,7)).isVisible());

}

}

**DungeonOfDooom-master\Sourcecode\project\src\tests**

* pom.xml

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>dungeon-of-doom</groupId>

<artifactId>dungoen-of-doom-tests</artifactId>

<version>1.0</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.mockito/mockito-all -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-all</artifactId>

<version>1.9.5</version>

</dependency>

<dependency>

<groupId>dungeon-of-doom</groupId>

<artifactId>dungeon-of-doom-service</artifactId>

<version>1.0</version>

</dependency>

</dependencies>

</project>