|  |
| --- |
| University of southampton |
| Contagion – Developer Guide |
|  |
| **Lewis Hill** |
| **10/22/2019** |

Table of Contents

[Setup (Server) 3](#_Toc22643401)

[Setup (Client - Local) 4](#_Toc22643402)

[Setup (Client – Remote) 4](#_Toc22643403)

[Updating Server-Client Communications 5](#_Toc22643404)

[Other Useful Areas 6](#_Toc22643405)

[Keywords 7](#_Toc22643406)

# Introduction

This guide is designed to provide a starting point for other people to tackle any bugs or implement new features for the Contagion game in the Influence Maximisation project. Please contact me at [lh9g14@soton.ac.uk](mailto:lh9g14@soton.ac.uk) for details not covered in this document.

In the ‘Setup’ sections, the first section details how to perform a first-time setup of the relevant area. This will be followed in green by a shorter section detailing how to access them in subsequent occasions, where required.

The credentials are:

Gmail: contagiongamesoton@gmail.com / southampt0N

Heroku: [email] / southampt0N!

# Setup (Server - Local)

Prerequisites: Node.js\*\*

To run the server, simply change *Server.LocalMode* and *Simulations.LocalMode*  to true and run *node server.js* in the command prompt in the ContagionServer directory. If you’re anticipating having to restart the server frequently, you may wish to call *servercode.bat*, which calls the above command after the previous instance is killed many times.

However, if you’re planning on doing any non-trivial debugging, I strongly suggest using Visual Studio Code, and following the guide here: <https://itnext.io/the-absolute-easiest-way-to-debug-node-js-with-vscode-2e02ef5b1bad>

Running the server locally allows you to test out any changes without having to go through the lengthier process of uploading your data remotely, and almost always behaves in the same way as the remote server. Any differences are controlled in the code by Server.LocalMode and Simulations.LocalMode – make sure these are set to true when running locally!

# Setup (Server- Remote)

1. Install the Heroku CLI (works well with Git Bash if you have it) at <https://devcenter.heroku.com/articles/heroku-cli>
2. Navigate to the ContagionServer folder in the command line
3. Enter heroku login and login with the credentials I gave you and Seb earlier
4. Enter: git remote add herokugit@heroku.com:stark-atoll-77422.git
5. Enter: git logs --tail

Subsequent Times:

1. Navigate to the ContagionServer folder
2. Enter: git logs --tail

# Setup (Client - Local)

# Setup (Client – Remote)

# Setup (Database – Remote Only)

Accessing the database:

(NOTE: This is a more convenient command line method that involves less setup, but not one that I have personally tested. If there are significant problems in getting this to work, I can write a guide for the more heavy-handed but visual method I used)

First Time:

1. Login to Heroku using the credentials I gave you and Seb earlier
2. Go here: <https://dashboard.heroku.com/apps/stark-atoll-77422> and click "Heroku Postgress"
3. Click Settings -> View Credentials and note the details
4. Setup postgres on your machine by following <https://devcenter.heroku.com/articles/heroku-postgresql#set-up-postgres-on-mac>
5. In the ContagionServer folder (in command line), enter heroku pg:psql
6. The command line should prompt you for some details that match what you saw in step 3.
7. For overall game data, enter the query "SELECT \* FROM master\_games\_table;" or individual step data, "SELECT \* FROM player\_actions\_table;" (without quotes)
8. For any more complex queries (like get a game and associated moves), I can write a guide at a later date.

Subsequent Times: Follow steps 1-3, 5 & 7 (possibly just 5 & 7 if your command line interface remembers your details)

# Updating Server-Client Communications

Overview

As this is a somewhat involved task, spread across the client and server files, a brief demonstration is included here.

Server -> Client communication is handled by *Server.sendClientMessage* in *server.js* in ContagionServer (outgoing) and *parseEvent* in S*imulations.js* in ContagionClient/js/sim (incoming)

Client -> Server communication is handled by the *sendServerMessage* function in S*imulations.js* (outgoing) and *Server.ParseMessage* function in *server.js* (incoming)

Both of these use the Message.js class, a very simple wrapper whose wrapper takes a *payload* (data to be sent) and *status* (identifier for the type of message). For instance:



Sends a NEW\_GAME\_TOKEN type message with a payload containing the client’s username.

The types of message that the clients and servers can send and receive are mutually exclusive. E.g. clients can only send (not receive) NEW\_GAME\_TOKEN messages, and servers can only receive (not send) these messages. This is not strictly enforced, but keeps things simple.

Implementing New Messages

For both client -> server and server -> client messages, implementing new types is a simple three step process.

1. Once you have the data you wish to send, put it in a list by enclosing the comma-separated data within square brackets (if there is more than one element to send), e.g.:

*var payload = [user.Username, user.Score, user.Fun];*

1. Then choose a descriptive name for your message type, e.g. USER\_RESULTS\_TOKEN, and create a new message & pass it to the dedicated communications function as shown above.
2. If you did this on the server, change to the client (and vice versa). Then, in the *Server.ParseMessage* or *parseEvent* function, change the switch function to include a new case for your chosen message type (e.g. USER\_RESULTS\_TOKEN), and implement the logic for how you want to handle the transmitted data from there. The communication function detects the message type and allows you to run whatever you like from there with the payload of the message.

# Other Useful Areas

Implementing new AIs

Server/Simulations.LocalMode, etc.

# Keywords

N.B. I strongly recommend using Agent Ransack (<https://www.mythicsoft.com/agentransack/>) for searching for these keywords – it is a powerful tool for searching the inside of files & has excellent filtering functionality (usually you’ll want to use \*.js for the file name).