Starting a game using the emulator

In this chapter we will learn how to:

1) Use the emulator.

2) Make the first connection between our game and the server.

3) Get the stageX and stageY.

4) Read the server log.

How do you use the emulator?

1. Go to the directory named Emulator and open index.html
2. The window that opens is the emulator definitions page.
3. “The file name” is the game’s SWF full or relative path.
4. “CONTAINER\_gameWidth” is the game’s width.
5. “CONTAINER\_gameHeight” is the game’s height
6. We set the “Number of Players” to 2.
7. We set the “Number of Viewers” to 0.
8. Press launch game

\*for more information on the Emulator visit: http://code.google.com/p/multiplayer-api/wiki/Emulator

Registering our game on the server

To tell the server our game is ready we call doRegisterOnServer() from the TickTacToeTuturialMain class, the TickTacToeTuturialMain class is responsible for all our game server communication.

If you tried loading both this tutorial’s SWF and the previous tutorial SWF in the emulator, you will notice that in this SWF the log will not be empty, and that in the first SWF the mouse clicks have a strange effect, we will fix later on in this tutorial.

\*for more information about doRegisterOnServer visit: <http://code.google.com/p/multiplayer-api/wiki/doRegisterOnServer>

Getting our stage position

1. After calling doRegisterOnServer(), the server will send our game a few custom values, in this tutorial we will talk about 2 of them: CUSTOM\_INFO\_KEY\_gameStageX, and CUSTOM\_INFO\_KEY\_gameStageY.
2. To get the custom values we need to override gotCustomInfo, we can do this by writing:  
   **override** **public** **function** gotCustomInfo(infoEntries:Array):**void** {  
   }
3. Now remember that in the first SWF the mouse clicks had a strange effect on the Ticktacktoe board . That is because when we tested the game as SWF, our game’s x and y positions were 0, but when the game is loaded into a container, in this case our emulator’s container, the game is not placed in the axis origin, which makes the absolute x and y positions of the mouse different.
4. To solve this problem we need to get the x and y positions at which our game is placed and decrease them from the absolute x and y positions we get from our mouse event.
5. To get our game’s x position we call T.custom (CUSTOM\_INFO\_KEY\_gameStageX, 0) during or after the gotCustomInfo callback.
6. To get our game’s y position we call T.custom (CUSTOM\_INFO\_KEY\_gameStageY, 0) during or after the gotCustomInfo callback.

\* For more information about gotCustomInfo visit:

<http://code.google.com/p/multiplayer-api/wiki/gotCustomInfo>

http://code.google.com/p/multiplayer-api/wiki/custom

Reading the server log

Now that the server log is not empty, we should learn how to make sense out of it.

Let’s start with the log attributes:

1. Num – the serial number of the log entry
2. User – the user name of the calling/getting game
3. FunctionName – the name of the function that was called
4. Arguments – a JSON string representing the arguments in the call

The calls we can see in this game so far are:

1. gotCustomInfo - represents a gotCustomInfo callback being sent to a specific user
2. gotUserInfo – represents a gotUserInfo callback being sent to a specific user
3. gotMatchStarted – represents a gotMatchStarted callback being sent to a specific user
4. doFinishedCallback – called after a callback was finished on a specific user game client.

For more information about the emulator visit: <http://code.google.com/p/multiplayer-api/wiki/Emulator>