Jelly Pirate: Networking Infrastructure

**Node.js**

We chose node.js because of its concurrency solution (non-blocking io) fit the real-time gaming scenario. Server is now host in the server room at Entertainment Technology Center under CMU network.

**Socket.io**

Game logic between phone and Unity is heavily rely on socket.io’s API. The core server code is basically passing messages to the proper client.

On Unity side we use a socket.io plugin to communicate to the server:

<https://github.com/fpanettieri/unity-socket.io>

**Connection**

The structure is basically 1 server and 5 client (4 phone and 1 Unity). Most in-game data (cannon rotation, shooting, button press…) are packed in two types of event: “u2p event” (unity to phone) and “p2u event”. The event pass with a JSON object containing the data of the event.

Ex**:** When we press a button on the phone, it send a p2u event to server with a custom name (for the functionality of this message) and up to 6 parameters. Server then pass the event to Unity. If Unity has set a callback for this event, players will see the action on the TV screen.

**Latency:**

Most traffic now is the rotation input from the phone. We haven’t do any optimization now and it send 60 message per second.

There’s a known problem that Unity occasionally fails to receive package for about 1 second. To mitigate the problem we interpolate the input value in Unity’s update loop. Later we may reduce transmission rate to 20/sec.

**Team Efficiency**

The working pipeline of communication between Phone and Unity is structured as a “Cross-Server Event Center”. We developed our own API that wrap the socket.io functions and JSON parser, so all messages sent/receive between clients can be done in one line of code.

A live document for all custom events is also very important since tasks for web front-end and Unity are usually assigned to different people. This is our current event list:

<https://docs.google.com/spreadsheets/d/1HrePI_9C8ubpN_qK3VaY-oKyB8jB7XkilfsPH5I5NB0/edit#gid=0>

**Event Center Detail**

The event center contains a hashtable that stores function pointers of callbacks. When event center receive package of certain type, it search the hashtable by eventName and invoke all of its callbacks.

For example, if a cannon in Unity want to listen to the firing button at phone, it should call this function:

**RegisterP2UEvent(“Shoot”, *OnShoot*);**

Where *OnShoot* is the callback in which Unity developers implement the actual function of the cannon. All this function do is saving the callback *OnShoot* into the hashtable in EventCenter.

If phone want to send message to Unity, all he need to do is call this function:

**SendP2UEvent(“Shoot”, 20,35);**

Which send this JSON object to Unity:

{

eventName:”Shoot”,

p1:”20”,

p2:”35”

}

Once Unity receive the event, it search the hashtable with the key “Shoot” and invoke callbacks with parameters (“20”,”35”).

----------------------------------------------------

Feel free to ask me if there’s anything unclear.

**Chih-Wei Chen**

*Backend Engineer*

[madturtle84@gmail.com](mailto:madturtle84@gmail.com) | <https://www.linkedin.com/in/jerryccw/>