Battle RoyAle Howto

Version: 0.2.0

Date: 06/02/13

Author: Evan Kluin

ICEMOBILE

Mensinge 2
1083 HA Amsterdam
Netherlands
www.icemobile.com
T: +31 (0) 20 368 06 45
F: +31 (0) 20 695 24 99

Table of contents

Introduction

This document describes the technical details of the ‘Battle RoyAIe’ IceMobile dev challenge. This includes a section about how to setup a connection with the Battle server and the protocol definition.

Please note that the server app and its documentation are being actively developed and therefore are subjected to change. The current version of the protocol is identified by a major and minor release number. Major versions break backwards compatibility and minor updates will add functionality without breaking compatibility. The version of the document is linked to the version of the protocol. Major version updates breaking backwards compatibility are kept to a minimum.

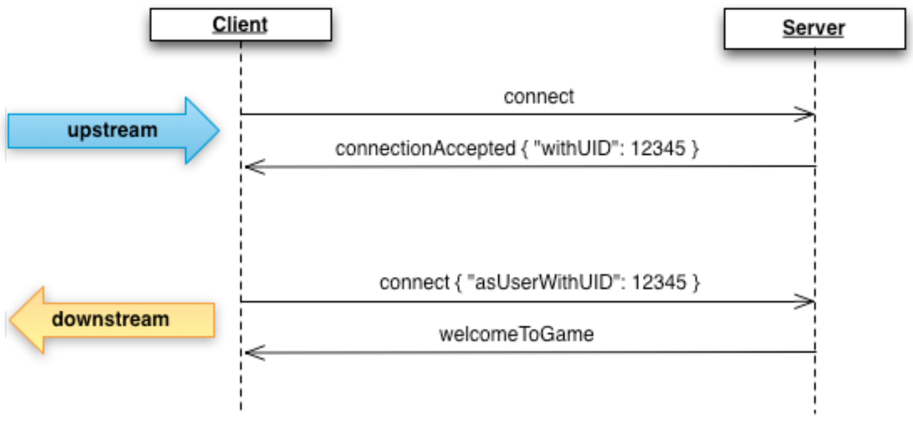
Current version of the Battle server is pretty bare-bone and isn’t doing a lot of error checking (A), so you might encounter some problems and/or crashes. Other than that, it’s pretty rock solid :P

Changelog

`0.2.0`
Added **Sending tank control messages** section
Connect upstream channel message Changed color and version string into object structure

Setting up a connection

As states in the brief protocol description we need to setup two socket connections with the server. First we setup the client's upstream channel socket by connecting on TCP **port 1359**. We send the connect message identifying ourself, if the server approves it will respond with a connection accepted message containing a client UID (Unique ID). The client must use the UID to setup the downstream channel socket by connecting to the server on TCP **port 1360**. If everything went ok the server will accept your second connection and responds with a *welcomeToGame* message. The state diagram below illustrates this flow.

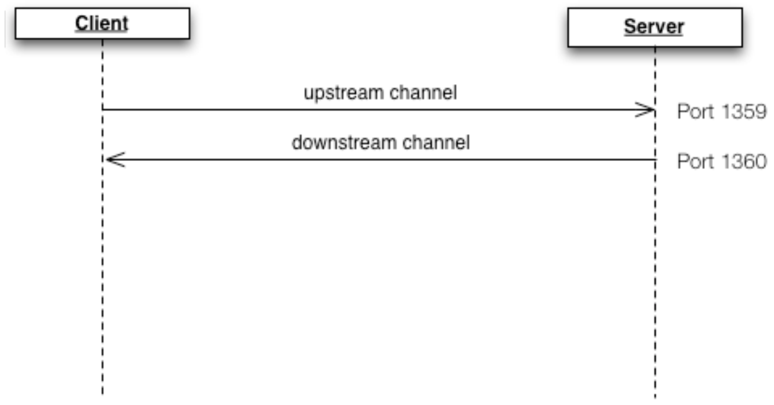

Client
Server
upstream
connect
connectionAccepted { "withUID": 12345 }
downstream
connect { "asUserWithUID": 12345 }
welcomeToGame

We now have two open socket connection with the server. and we should see a tank appearing within the server view. From this moment on we'll have to wait for the server to send a *gameWillStart* message.

The Protocol

Communicating with the Battle server is done over plain TCP sockets. Two connections to be precise. We're using two sockets to make communication for the client pretty easy, one socket will be used to communicate from client to server, the 'upstream channel', the other socket is used for communicating from server to client, the 'downstream channel'. Typically the upstream channel is used to send 'control messages' for your tank. On every message send from client to server over client's upstream channel the server will respond through the same channel with either an OK or an ERROR response. All messages send over the downstream channel, so from server to client, are consume only for the client (no response needed).

A pretty diagram



All messages are encoded as **single lined** JSON strings terminated by a newline character '\n'. For readability we will use newlines while describing the JSON messages in this document. Don't include them while communicating with the server, only terminate your message line with a '\n'.

JSON messages

All JSON commands should be pretty self explanatory.

Please note that all JSON commands should be encoded as a *single line* string, terminated by a newline character '\n'. So don't append the String "\n" to your JSON command string Ronald **but append '\n'.**

Client upstream channel

→ Client message to server

← Server responds

Client downstream channel

→ Client message to server

← Server message to client

Connect upstream channel message

→

```
{
    "connect": {
        "asUser": "[username]",
        "withTankColor": {
            "r": 255,
            "g": 0,
            "b": 255
        },
        "usingProtocolVersion": {
            "major": 0,
            "minor": 2,
            "revision": 0
        }
    }
}
```

`asUser`	Your username.
`usingProtocolVersion`	The protocol version

←

```
{
    "connectionAccepted": {
        "forUser": "[username]",
        "withUID": 14326
    }
}
```

`forUser`	Boucing your username.
`withUID`	The client’s UID

(Changed color and version string into object structure) Updated in 0.2.0
Available since 0.1.0

Connect downstream channel message

→

```
{
    "connect": {
        "asUserWithUID": 14326,
        "usingProtocolVersion": "0.1.0"
    }
}
```

`asUserWithUID`	The identifier you received while connecting the upstream channel.

←

```
{
    "welcomeToGame": {
        "withState": "lobby",
        "rules": {
            "moveSpeed": 10,
            "rotationSpeed": 45,
            "turretRotationSpeed": 35,
            "fireInterval": 2,
            "ballisticsTravelSpeed": 150
            "fieldOfView": 90,
            "turretFieldOfView": 45,
            "hp": 100,
            "ballisticDamage": 20,
            "enemyHitScore": 10,
            "enemyKillScore": 50,
        },
        "currentPlayers": [
                              "Ahmed",
                              "Bartol",
                              "Borre",
                              "Bruno",
                              "..."
                              ],
        "serverTimestamp", 10.43655346
    }
}
```

Available since 0.1.0

Sending tank control messages

All tank control messages are communicated by the client to the server over the upstream channel. In all cases the server will respond **through the same channel** with a response message. This can either be an *acknowledge message* echoing the command you just send or an *error message*.

All messages put your tank into a certain state. Lets say you send the *moveForwardWithSpeed* to your tank and the server responds with an ACK message. This now means your tank will keep on moving forward until it either hits some object or you tell it to stop moving by sending the *stop* message. Messages can override the current state without in some cases. So for example you're moving forward with speed 1.0 and you send the *moveBackwardWithSpeed: 0.5* message, this will override your current forward move state and immediately put your tank in reverse with a speed of 0.5. The only exception to this is the scanning state, while scanning no other messages other than *stop: scanning* are accepted.

Example message flow

An example message is shown below. In this example we send the *moveForwardWithSpeed* message to the server and read the response from the server. We'll give an example of an acknowledge flow and an error response flow.

Acknowledge flow

→

```
{ "moveForwardWithSpeed": 1.0 }
```

←

```
{ "ack": "moveForwardWithSpeed" }
```

Error flow

→

```
{ "moveForwardWithSpeed": 1.0 }
```

←

```
{
    "errorOccured": {
        "whileExecutingCommand": "moveForwardWithSpeed",
        "withReason": "ObstacleInTheWay"
    }
}
```

Another error flow

→

```
{ "moveForwardWithSpeed": 299792458 }
```

←

```
{
    "error": {
        "withReason": "InvalidValue"
    }
}
```

For the next messages described in this chapter we'll not explicitly mention the possible responses because they all meet the structure described above. The only variable is the error reason.

These message specific error messages are described per message. There are also some general defined error codes which could be returned as response on every message.

General error codes
InvalidCommand
InvalidValue
YouDied
ScanInProgress

Putting the tank in Drive

```
{ "moveForwardWithSpeed": 1.0 }
```

The speed should be a floating point number between **0.0** and **1.0**. Speed is relative to the game defined movement speed. You can vary your speed for whatever tactical reason you can come up with :p

Error codes

ObstacleInTheWay

EndOfWorld

Available since 0.2.0

Putting the tank in R

```
{ "moveBackwardWithSpeed": 1.0 }
```

The speed should be a floating point number between **0.0** and **1.0**. Speed is relative to the game defined movement speed. You can vary your speed for whatever tactical reason you can come up with :p

Error codes

ObstacleInTheWay

EndOfWorld

Available since 0.2.0

Rotating the tank

```
{ "rotateTank": 60.0 }
```

Rotate the tank around its Y axis. The value should be between in range of **-180.0** to **180.0** degrees.

Error codes

Available since 0.2.0

Rotating the turret

```
{ "rotateTurret": 45.0 }
```

Rotate the turret relative to the rotation of the tank. The value should be between in range of **-180.0** to **180.0** degrees.

Error codes

Available since 0.2.0

Stop command

```
{ "stop": "[action]" }
```

Stops the given ***action*** (if in progress). Valid values for ***action*** are:
- moving
- tankRotation
- turretRotation
- scanning

Error codes

ActionIsNotBeingPerformed

Available since 0.2.0