TankTrouble Version 4.0 Server Breakdown

**Main Idea:** This report explains how the newly implemented server serves the application and controls the communication between players.

**Git Repo:** <https://github.com/Abdullah2Cool/TankTrouble-Online.git>

**Online Version:** <https://tank-io.herokuapp.com/>

**Server File:** Version 4.0/server.ts

**Main Client File:** Version 4.0/client/src/States/GameState.ts

**Terminology:**

The following are the two main functions used by the server and client to communicate with each other.

**socket.emit**(“message name”, {key: value, key: value})

* ======emit a message with a specific name and the given data.

**socket.on**(“message name”, function(data)) {

var x = data[key];

}

* ====== listen to a message with the given name and run a function that receives the data as a parameter.

**To start the server and the game:**

* Navigate to the root folder using the command prompt
* Type the command: *npm start*
* The server will start on *localhost:3000*

The following code creates a server and serves the client folder (the game) to localhost: 3000:

**import** {Player} **from** "./server/Player";  
  
**var** express = require('express');  
**var** app = express();  
**var** server = require('http').createServer(app);  
  
app.get('/', **function** (req, res) {  
 res.sendFile(\_\_dirname + '/client/index.html');  
});  
app.get('/\*', **function** (req, res) {  
 **var** file = req.params[0];  
 // console.log('\t :: Express :: file requested : ' + file);  
 res.sendFile(\_\_dirname + "/client/" + file);  
});  
  
server.listen(process.env.PORT || 3000);  
console.log("Server started on localhost:3000");

* The first 4 lines of code are importing the required node modules and files
* The starting point of the program is the *index.html* file inside the client folder

The next bit of code attaches the socket.io module to the server and creates two containers to hold all the sockets(connections) and Player data. The player and the socket object share the same key (socket.id) in their respective containers.

**var** io = require('socket.io')(server, {});  
  
**var** ALL\_SOCKETS = {};  
**var** ALL\_PLAYERS = {};

* The socket.io object connected to the server is now represented by the variable ‘*io*’

Here is the player class used by the server to hold all the information about each client:

**Path:** Version 4.0/server/player.ts

**export class** Player {  
 x = 400;  
 y = 400;  
 r = 0;  
 name;  
 id;  
 health = 20;  
  
 **constructor**(id, name) {  
 **this**.id = id;  
 **this**.name = name;  
 }  
}

Inside **index.html** (Version 4.0/client/index.html)

The socket.io client-side library is imported:

<script src="/socket.io/socket.io.js"></script>

Inside **GameState.ts**, when the game initializes a connection/socket:

**this**.socket = io();

the server is alerted, and the following code executes in the server:

io.on('connect', **function** (socket) {  
 console.log("Socket connected:", socket.id);  
 ALL\_SOCKETS[socket.id] = socket;

Then the client **emits** a message with name “start” with the client’s name:

**this**.socket.emit("start", {name: **this**.name});

The server listens for this message and runs a function that receives the player’s name:

**var** player;  
socket.on("start", **function** (data) {  
  
 player = **new** Player(socket.id, data.name);  
  
 console.log("Recieved Name:", data.name);  
  
 // tell the client their own id and the rest of the player  
 socket.emit("serverState", {  
 id: socket.id,  
 otherPlayers: ALL\_PLAYERS  
 });  
  
 ALL\_PLAYERS[socket.id] = player;  
  
 // tell everyone else that there is a new player  
 socket.broadcast.emit("newPlayer", {  
 id: socket.id,  
 newPlayer: player,  
 name: data.name  
 });  
});

* The server creates an instance of the player with the name and the server assigned id
* This id along with the container holding all other player’s info back to the client (message name = “serverState”)
* Then it adds the newly created player to the list of ALL\_Players[]
* The server then emits a message to all other players about the creation of a new player (message name = “newPlayer”)

socket.emit === sends the message to the sender only

socket.broadcast.emit === sends the message to everyone but the sender

The client has attached appropriate functions to run when it receives a message from the server.

**this**.socket.on("serverState", **this**.onServerState.bind(**this**));  
**this**.socket.on("newPlayer", **this**.onNewPlayer.bind(**this**));  
**this**.socket.on("removed", **this**.onRemoved.bind(**this**));  
**this**.socket.on("update", **this**.onUpdate.bind(**this**));  
**this**.socket.on("shoot", **this**.onShoot.bind(**this**));

Inside onServerState:

onServerState(data) {  
 **this**.id = data.id;  
  
 console.log("My id from server:", **this**.id);  
 console.log("Other Players:", **this**.otherPlayers);  
  
 **for** (**var** x **in** data.otherPlayers) {  
 **var** p = data.otherPlayers[x];  
 **var** t = **new** otherTank(**this**.game, p.x, p.y, p.id, p.name);  
  
 **this**.otherPlayers[x] = t;  
 // this.tank.addNewPlayer(t);  
 **this**.otherGroup.add(t)  
 }  
}

* The client receives
* assigned id from the server and creates new otherTank objects based on the info provided by the server

Inside onNewPlayer:

onNewPlayer(data) {  
 console.log("Detected new player:", data.newPlayer);  
  
 console.log(data);  
  
 **var** t = **new** otherTank(**this**.game, data.newPlayer.x, data.newPlayer.y, data.id, data.name);  
 **this**.otherPlayers[data.id] = t;  
 // this.tank.addNewPlayer(t);  
 **this**.otherGroup.add(t)  
}

* Creates an instance of new player and adds it to the game

**At this point, the game has been served to the client, a connection between the client and the server has been established, the client knows about the existence of other players and all other players know about the newly connected client. Now during the game:**

In the update function of the game that runs 60 fps, the client it’s current information:

**this**.socket.emit('position', {  
 x: **this**.tank.x,  
 y: **this**.tank.y,  
 r: **this**.tank.rotation,  
 id: **this**.id,  
 health: **this**.tank.health,  
});

The server stores it in the player object:

socket.on("position", **function** (data) {  
 ALL\_PLAYERS[socket.id].x = data.x;  
 ALL\_PLAYERS[socket.id].y = data.y;  
 ALL\_PLAYERS[socket.id].r = data.r;  
 ALL\_PLAYERS[socket.id].health = data.health;  
 // console.log("ID:", socket.id, "Received health:", data.health);  
});

Every time a client shoots, it emits a ‘shoot’ message:

bulletFire(bullet, weapon) {  
 bullet.body.bounce.setTo(1, 1);  
 // this.game.world.bringToTop(this);  
 **this**.socket.emit('shoot');  
}

* This is done by the tank object itself inside the “bulletFire” function
* Path = Version 4.0/client/src/GameObjects/Tank.ts

The server listens for it and then tells the all other clients to shoot a bullet originating from the tank that represents the shooter locally on their machines:

socket.on('shoot', **function** (data) {  
 socket.broadcast.emit('shoot', {id: socket.id});  
});

The alerted clients respond accordingly:

onShoot(data) {  
 **this**.otherPlayers[data.id].weapon.fire();  
 // console.log(data.id, "shot a bullet.");  
}

Inside the server, there is a function that runs 30 fps and creates a packet of information that holds the updated stats about each player. This packet is then sent to every client, so they can update the positions of all other players.

setInterval(**function** () {  
 **var** pack = {};  
 **for** (**var** i **in** ALL\_PLAYERS) {  
 **var** player;  
 player = ALL\_PLAYERS[i];  
 pack[i] = {  
 x: player.x,  
 y: player.y,  
 r: player.r,  
 health: player.health,  
 };  
 }  
  
 **for** (**var** i **in** ALL\_SOCKETS) {  
 **var** socket = ALL\_SOCKETS[i];  
 socket.emit("update", pack);  
 }  
  
}, 1000 / 30);

The client listens to this message and responds accordingly:

onUpdate(data) {  
 // console.log("Everyone else's info:");  
  
 **for** (**var** i **in** data) {  
 **var** x, y, r, id, health;  
 **if** (i != **this**.id) {  
 x = data[i].x;  
 y = data[i].y;  
 r = data[i].r;  
 id = data[i].id;  
 health = data[i].health  
  
 **if** (**this**.otherPlayers[i] == **null**) {  
 console.log("Player is null.");  
 } **else** {  
 **this**.otherPlayers[i].updateInfo(x, y, r, health);  
 }  
 }  
 }  
}

* Each client makes sure to ignore info about themselves
* This updated info is passed to the **otherTank** object
* Path = Version 4.0/client/src/GameObjects/otherTank.ts

The otherTank object updates its values:

updateInfo(x, y, r, health) {  
 **this**.game.add.tween(**this**).to({x: x, y: y, rotation: r}, 1000 / 60, "Sine.easeInOut", **true**);  
 **this**.health = health;  
}

* The tank tweens to the new position and rotation to avoid jerkiness

**This is all that the game needs to function as a multiplayer game. The last thing we need to handle is when a player disconnects:**

Inside the server:

socket.on("disconnect", **function** () {  
 **delete** ALL\_SOCKETS[socket.id];  
 **delete** ALL\_PLAYERS[socket.id];  
  
 socket.broadcast.emit("removed", {  
 id: socket.id  
 });  
  
 console.log("Socket disconnected:", socket.id);  
});

* Delete the player from the two main lists
* Tell everyone else that the player left

By the client:

onRemoved(data) {  
 **this**.otherPlayers[data.id].displayName.destroy();  
 **this**.otherPlayers[data.id].weapon.bullets.destroy();  
 **this**.otherPlayers[data.id].healthBar.kill();  
 **this**.otherPlayers[data.id].destroy();  
 **delete this**.otherPlayers[data.id];  
 console.log("Player Removed:", data.id);  
 console.log("Other's list:", **this**.otherPlayers);  
}

* Destroys all related assets to the player and remove them from the list of all players