TankTrouble Version 2.0 Breakdown

**Main Idea:** This report highlights the function of every important file and code snippet in the code.

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

**Online Version:** <https://abdullah2cool.github.io/Tank.io/>

Everything is under Code/Releases/Version 2.0/

**Index.html:**

* Imports firebase and all its functionalities (probably don’t need all of them)

<script src="https://www.gstatic.com/firebasejs/4.6.1/firebase.js"></script>  
<script>  
 // Initialize Firebase  
 **var** config = {  
 apiKey: "AIzaSyBiRVH6DEyouNRjU92fUdVpSRFbqg6BkzP1dY",  
 authDomain: "tank-trouble-online.firebaseapp.com",  
 databaseURL: "https://tank-trouble-online.firebaseio.com",  
 projectId: "tank-trouble-online",  
 storageBucket: "tank-trouble-online.appspot.com",  
 messagingSenderId: "446961097968"  
 };  
 firebase.initializeApp(config);  
</script>

* The rest of the <script> tags import all the JavaScript files
* The <body> tag has a <div> tag with id = “content”. This is the div the phaser game runs in

**Main.ts:**

**constructor**(divID: **string**) {  
 **this**.game = **new** Phaser.Game(900, 800, Phaser.*AUTO*, divID, {  
 preload: **this**.preload,  
 create: **this**.create,  
 update: **this**.update  
 });  
}

* Creates the phaser game object and points it to where the necessary functions are located for the game to function
* Points it to the <div> that we created earlier

preload() {  
 **this**.game.load.tilemap('map', 'gameMap.json', **null**, Phaser.Tilemap.*TILED\_JSON*);  
 **this**.game.load.image('tiles', 'Tileset.png');  
 **this**.game.load.image("tank", "Red Tank.png");  
 **this**.game.load.image("otherTank", "Blue Tank.png");  
 **this**.game.load.image("bullet", "bullet.png");  
  
 **this**.FIREBASE = **new** util\_Firebase();  
}

* Loads the map and all the images and assigns them a key which can be used to reference these assets later in the program
* These assets are stored in cache
* Creates a new instance of the firebase utility class

create() {  
 **this**.game.physics.startSystem(Phaser.Physics.*ARCADE*);  
 **this**.game.stage.backgroundColor = '#dfdfdf';  
  
 **this**.map = **this**.game.add.tilemap('map');  
 **this**.map.addTilesetImage('tiles');  
 **this**.layer = **this**.map.createLayer("Tile Layer 1");  
 **this**.layer.resizeWorld();  
 **this**.map.setCollision([33]);  
  
 **this**.id = **this**.FIREBASE.generateKey();  
  
 **this**.tank = **new** Tank(**this**.game, 300, 200, "tank", **this**.id, **this**.layer);  
 **this**.game.add.existing(**this**.tank);  
 **this**.game.camera.follow(**this**.tank);  
  
 **this**.FIREBASE.pushNewestPlayer(**this**.tank.id);  
 **this**.FIREBASE.checkForPreviousPlayers(**this**.tank.id, **this**.game, **this**.layer, **this**.tank);  
 **this**.FIREBASE.checkForNewPlayers(**this**.tank.id, **this**.game, **this**.layer, **this**.tank);  
 **this**.FIREBASE.onClose(**this**.tank.id);  
}

* Starts the physics system
* Sets the background colour
* Creates a map object and tells it where all of it’s tiles are located
* Creates a layer from the map (my map has only one layer)
* Resizes the world based on the dimensions of the map
* Sets which tiles the game objects can collide with in the map
* Generates a new id for the player, create a tank with that id and adds it to the game
* Tells the camera to follow the player
* Pushes the tank’s info to firebase,
* Checks for previous players and sets up a method which will detect new players in the future
* Sets a method that will remove the player from firebase once the player leaves

**util\_Firebase.ts:**

* The functions names are self-explanatory
* General Structure:
  + ref = the place in firebase where the event will take place
  + every time a “value” changes in that place, get all the info in that place and update/create the appropriate property/object
  + OR
  + Update firebase with the passed values

**Tank.ts:**

* Extends the Phaser.Sprite class
* Creates a sprite and gives adds it to the physics system (gives it a hit-box)

**this**.weapon.trackSprite(**this**, 34, 0, **true**);  
**this**.weapon.onFire.add(**this**.bulletFire, **this**);  
**this**.weapon.onKill.add(**this**.bulletDead, **this**);

* The above code tells the weapon to follow the bullet and launch it at an offset
* It also attaches functions to run when a bullet is fired and when the bullet is dead and when it is fired

// collide with the map  
**this**.game.physics.arcade.collide(**this**, **this**.layer);  
// collide the bullets with the map  
**this**.game.physics.arcade.collide(**this**.weapon.bullets, **this**.layer);  
// collide with the the bullets  
**this**.game.physics.arcade.collide(**this**, **this**.weapon.bullets, **this**.bulletHit);  
// collide with other tank's bullets  
**this**.otherTanks.forEach(**function** (otherTank) {  
 **this**.game.physics.arcade.collide(otherTank, **this**.weapon.bullets, **this**.bulletHit);  
}, **this**);  
  
**this**.body.velocity.x = 0;  
**this**.body.velocity.y = 0;  
**this**.body.angularVelocity = 0;  
  
**if** (**this**.lefKey.isDown) {  
 **this**.body.angularVelocity = -200;  
}  
**else if** (**this**.rightKey.isDown) {  
 **this**.body.angularVelocity = 200;  
}  
  
**if** (**this**.upKey.isDown) {  
 **this**.game.physics.arcade.velocityFromAngle(**this**.angle, **this**.velocity, **this**.body.velocity);  
} **else if** (**this**.downKey.isDown) {  
 **this**.game.physics.arcade.velocityFromAngle(**this**.angle, -**this**.velocity, **this**.body.velocity);  
}  
**if** (**this**.shootKey.isDown) {  
 **this**.weapon.fire();  
}  
**let** x = 0;  
**this**.weapon.bullets.forEach(**function** (bull) {  
 **if** (bull.alive) {  
 **this**.bulletInfo[x] = 1;  
 } **else** {  
 **this**.bulletInfo[x] = 0;  
 }  
 x++;  
}, **this**);  
**this**.FIREBASE.updatePlayerInfo(**this**.id, **this**.x, **this**.y, **this**.rotation, **this**.bulletInfo);

* Collide with appropriate things and run a function if needed when there is a collision
* Move the tank based on the keys pressed
* Update the status of each bullet in an array of 0 and 1.
* This array gets passed to firebase along with the player’s id, position and rotation

**otherTank.ts:**

* Same as Tank.ts but with a few less functions
* The main part is fetching the info from firebase

**this**.FIREBASE.getDatabase().ref("Players/" + **this**.id).on("value", snap => {  
 **if** (!snap.exists()) {  
 console.log("Player doesn't exist anymore.")  
 **this**.destroy();  
 } **else** {  
 **this**.x = snap.val().x;  
 **this**.y = snap.val().y;  
 **this**.rotation = snap.val().r  
 }  
});

* Destroys the tank when the info isn’t available in firebase
* Updates the info every time it changes

**this**.FIREBASE.getDatabase().ref("Players/" + **this**.id + "/bullets").on("value", snap => {  
 **if** (snap.exists()) {  
 **let** bulletsInCloud = snap.val();  
 console.log("Local: " + **this**.bulletInfo);  
 console.log("Cloud: " + bulletsInCloud);  
 **this**.bulletInfo = bulletsInCloud;  
 **let** x = 0;  
 **this**.weapon.bullets.forEach(**function** (bull) {  
 **if** (bulletsInCloud[x] == 0) {  
 **this**.bulletInfo[x] = 0;  
 } **else** {  
 **this**.bulletInfo[x] = 1;  
 }  
 x++;  
 }, **this**);  
 console.log("Updated: " + **this**.bulletInfo);  
 }  
});

* Updates the status of the bullets based on what’s in the cloud

update() {  
 **this**.game.physics.arcade.collide(**this**.weapon.bullets, **this**.layer);  
 **this**.game.physics.arcade.collide(**this**.tank, **this**.weapon.bullets, **this**.bulletHit);  
 // collide with other tank's bullets  
 **this**.otherTanks.forEach(**function** (otherTank) {  
 **this**.game.physics.arcade.collide(otherTank, **this**.weapon.bullets, **this**.bulletHit);  
 }, **this**);  
  
 **let** x = 0;  
 **this**.weapon.bullets.forEach(**function** (bull) {  
 **if** (**this**.bulletInfo[x] == 1) {  
 **if** (!bull.alive) {  
 **this**.weapon.fire();  
 console.log("Other Player shot bullet.");  
 }  
 } **else** {  
 **if** (bull.alive) {  
 bull.kill();  
 console.log("The bullet died.");  
 }  
 }  
 x++;  
 }, **this**);  
}

* Collides itself with the map, its own bullets and otherTanks (last one doesn’t work that well)
* Shoots bullets if they exist in database and kills them if they don’t exist in the database