```
class Game {
    constructor() {
        if (!Detector.webgl) Detector.addGetWebGLMessage();

        this.modes = Object.freeze({
            NONE: Symbol("none"),
            PRELOAD: Symbol("preload"),
            INITIALISING: Symbol("initialising"),
            CREATING_LEVEL: Symbol("creating_level"),
            ACTIVE: Symbol("active"),
            GAMEOVER: Symbol("gameover")
        });
        this.mode = this.modes.NONE;

        this.container;
        this.player = {};
        this.stats;
        this.controls;
        this.camera;
        this.scene;
        this.renderer;
        this.cellSize = 16;
        this.interactive = false;
        this.levelIndex = 0;
        this._hints = 0;
        this.score = 0;
        this.debug = false;
        this.debugPhysics = false;
        this.cameraFade = 0.05;
        this.mute = false;
        this.collect = [];

        this.messages = {
            text: [
                "Welcome to LostTreasure",
                "GOOD LUCK!"
            ],
            index: 0
        }

        if (localStorage && !this.debug) {
            //const levelIndex = Number(localStorage.getItem('levelIndex'));
            //if (levelIndex!=undefined) this.levelIndex = levelIndex;
        }

        this.container = document.createElement('div');
        this.container.style.height = '100%';
        document.body.appendChild(this.container);

        const sfxExt = SFX.supportsAudioType('mp3') ? 'mp3' : 'ogg';
        const game = this;
        this.anims = ["ascend-stairs", "gather-objects", "look-around", "push-button", "run"];
        this.tweens = [];

        this.assetsPath = '../assets/';

        const options = {
            assets: [
                `${this.assetsPath}sfx/gliss.${sfxExt}`,
                `${this.assetsPath}sfx/factory.${sfxExt}`,
                `${this.assetsPath}sfx/button.${sfxExt}`,
                `${this.assetsPath}sfx/door.${sfxExt}`,
                `${this.assetsPath}sfx/fan.${sfxExt}`,
                `${this.assetsPath}fbx/environment.fbx`,
                `${this.assetsPath}fbx/girl-walk.fbx`,
```

```
            `${this.assetsPath}fbx/usb.fbx`,
        ],
        oncomplete: function() {
            game.init();
            game.animate();
        }
    }

    this.anims.forEach(function(anim) { options.assets.push(`${game.assetsPath}fbx/${anim}.fbx`) });

    this.mode = this.modes.PRELOAD;

    document.getElementById("camera-btn").onclick = function() { game.switchCamera(); };
    document.getElementById("briefcase-btn").onclick = function() { game.toggleBriefcase(); };
    document.getElementById("action-btn").onclick = function() { game.contextAction(); };
    document.getElementById("sfx-btn").onclick = function() { game.toggleSound(); };

    this.actionBtn = document.getElementById("action-btn");

    this.clock = new THREE.Clock();

    //this.init();
    //this.animate();
    const preloader = new Preloader(options);
}

toggleBriefcase() {
    const briefcase = document.getElementById("briefcase");
    const open = (briefcase.style.opacity > 0);

    if (open) {
        briefcase.style.opacity = "0";
    } else {
        briefcase.style.opacity = "1";
    }
}

toggleSound() {
    this.mute = !this.mute;
    const btn = document.getElementById('sfx-btn');

    if (this.mute) {
        for (let prop in this.sfx) {
            let sfx = this.sfx[prop];
            if (sfx instanceof SFX) sfx.stop();
        }
        btn.innerHTML = '<i class="fas fa-volume-off"></i>';
    } else {
        this.sfx.factory.play
        this.sfx.fan.play();
        btn.innerHTML = '<i class="fas fa-volume-up"></i>';
    }
}

contextAction() {
    console.log('contextAction called ' + JSON.stringify(this.onAction));
    if (this.onAction !== undefined) {
        if (this.onAction.action != undefined) {
            this.action = this.onAction.action;
        }
    }

    const game = this;

    if (this.onAction.mode !== undefined) {
```

```
        switch (this.onAction.mode) {
            case 'open-doors':
                this.sfx.door.play();
                this.sfx.button.play();
                const door = this.doors[this.onAction.index];
                const left = door.doors[0];
                const right = door.doors[1];
                this.cameraTarget = { position: left.position.clone(), target: left.position.clone() };
                this.cameraTarget.position.y += 150;
                this.cameraTarget.position.x -= 950;
                //target, channel, endValue, duration, oncomplete, easing="inOutQuad"){
                this.tweens.push(new Tween(left.position, "z", left.position.z - 240, 2, function() {
                    game.tweens.splice(game.tweens.indexOf(this), 1);
                }));
                this.tweens.push(new Tween(right.position, "z", right.position.z + 240, 2, function() {
                    game.tweens.splice(game.tweens.indexOf(this), 1);
                    delete game.cameraTarget;
                    const door = game.doors[this.onAction.index];
                    const left = door.doors[0];
                    const right = door.doors[1];
                    const leftProxy = door.proxy[0];
                    const rightProxy = door.proxy[1];
                    leftProxy.position = left.position.clone();
                    rightProxy.position = right.position.clone();
                }))
                break;
            case 'collect':
                this.activeCamera = this.player.cameras.collect;
                this.collect[this.onAction.index].visible = false;
                if (this.collected == undefined) this.collected = [];
                this.collected.push(this.onAction.index);
                document.getElementById("briefcase").children[0].children[0].children[this.onAction.index].children[0].src =
this.onAction.src;

                break;
        }
    }
}

switchCamera(fade = 0.05) {
    const cams = Object.keys(this.player.cameras);
    cams.splice(cams.indexOf('active'), 1);
    let index;
    for (let prop in this.player.cameras) {
        if (this.player.cameras[prop] == this.player.cameras.active) {
            index = cams.indexOf(prop) + 1;
            if (index >= cams.length) index = 0;
            this.player.cameras.active = this.player.cameras[cams[index]];
            break;
        }
    }
    this.cameraFade = fade;
}

initSfx() {
    this.sfx = {};
    this.sfx.context = new(window.AudioContext || window.webkitAudioContext)();
    const list = ['gliss', 'door', 'factory', 'button', 'fan'];
    const game = this;
    list.forEach(function(item) {
        game.sfx[item] = new SFX({
            context: game.sfx.context,
            src: { mp3: `${game.assetsPath}sfx/${item}.mp3`, ogg: `${game.assetsPath}sfx/${item}.ogg` },
            loop: (item == 'factory' || item == 'fan'),
```

```
            autoplay: (item == 'factory' || item == 'fan'),
            volume: 0.3
        });
    })
}

set activeCamera(object) {
    this.player.cameras.active = object;
}

init() {
    this.mode = this.modes.INITIALISING;

    this.camera = new THREE.PerspectiveCamera(45, window.innerWidth / window.innerHeight, 1, 2000);

    let col = 0x605050;
    this.scene = new THREE.Scene();
    this.scene.background = new THREE.Color(col);
    this.scene.fog = new THREE.Fog(col, 500, 1500);

    let light = new THREE.HemisphereLight(0xffffff, 0x444444);
    light.position.set(0, 200, 0);
    this.scene.add(light);

    light = new THREE.DirectionalLight(0xffffff);
    light.position.set(0, 200, 100);
    light.castShadow = true;
    light.shadow.mapSize.width = 2048;
    light.shadow.mapSize.height = 2048;
    light.shadow.camera.top = 3000;
    light.shadow.camera.bottom = -3000;
    light.shadow.camera.left = -3000;
    light.shadow.camera.right = 3000;
    light.shadow.camera.far = 3000;
    this.scene.add(light);

    // ground
    var mesh = new THREE.Mesh(new THREE.PlaneBufferGeometry(2000, 2000), new THREE.MeshPhongMaterial({ color:
0x999999, depthWrite: false }));
    mesh.rotation.x = -Math.PI / 2;
    //mesh.position.y = -100;
    mesh.receiveShadow = true;
    //this.scene.add( mesh );

    var grid = new THREE.GridHelper(2000, 40, 0x000000, 0x000000);
    //grid.position.y = -100;
    grid.material.opacity = 0.2;
    grid.material.transparent = true;
    //this.scene.add( grid );

    // model
    const loader = new THREE.FBXLoader();
    const game = this;

    loader.load(`${this.assetsPath}fbx/girl-walk.fbx`, function(object) {

        object.mixer = new THREE.AnimationMixer(object);
        object.mixer.addEventListener('finished', function(e) {
            game.action = 'look-around';
            if (game.player.cameras.active == game.player.cameras.collect) {
                game.activeCamera = game.player.cameras.back;
                game.toggleBriefcase();
            }
        })
        object.castShadow = true;
```

```
        game.player.mixer = object.mixer;
        game.player.root = object.mixer.getRoot();

        object.name = "Character";

        object.traverse(function(child) {
            if (child.isMesh) {
                child.castShadow = true;
                child.receiveShadow = true;
            }
        });

        game.scene.add(object);
        game.player.object = object;
        game.player.walk = object.animations[0];

        game.joystick = new JoyStick({
            onMove: game.playerControl,
            game: game
        });

        game.createCameras();
        game.loadEnvironment(loader);
    }, null, this.onError);

    this.renderer = new THREE.WebGLRenderer({ antialias: true });
    this.renderer.setPixelRatio(window.devicePixelRatio);
    this.renderer.setSize(window.innerWidth, window.innerHeight);
    this.renderer.shadowMap.enabled = true;
    this.renderer.shadowMap.type = THREE.PCFSoftShadowMap; // default THREE.PCFShadowMap
    this.renderer.shadowMapDebug = true;
    this.container.appendChild(this.renderer.domElement);

    window.addEventListener('resize', function() { game.onWindowResize(); }, false);

    // stats
    if (this.debug) {
        this.stats = new Stats();
        this.container.appendChild(this.stats.dom);
    }

    this.initSfx();
}

loadUSB(loader) {
    const game = this;

    loader.load(`${this.assetsPath}fbx/usb.fbx`, function(object) {
        game.scene.add(object);

        const scale = 0.2;
        object.scale.set(scale, scale, scale);
        object.name = "usb";
        object.position.set(-416, 0.8, -472);
        object.castShadow = true;

        game.collect.push(object);

        object.traverse(function(child) {
            if (child.isMesh) {
                child.castShadow = true;
                child.receiveShadow = true;
            }
        });
```

```
        game.loadNextAnim(loader);
    }, null, this.onError);
}

loadEnvironment(loader) {
    const game = this;

    loader.load(`${this.assetsPath}fbx/environment.fbx`, function(object) {
        game.scene.add(object);
        game.doors = [];
        game.fans = [];

        object.receiveShadow = true;
        object.scale.set(0.8, 0.8, 0.8);
        object.name = "Environment";
        let door = { trigger: null, proxy: [], doors: [] };

        object.traverse(function(child) {
            if (child.isMesh) {
                if (child.name.includes('main')) {
                    child.castShadow = true;
                    child.receiveShadow = true;
                } else if (child.name.includes('mentproxy')) {
                    child.material.visible = false;
                    game.environmentProxy = child;
                } else if (child.name.includes('door-proxy')) {
                    child.material.visible = false;
                    door.proxy.push(child);
                    checkDoor();
                } else if (child.name.includes('door')) {
                    door.doors.push(child);
                    checkDoor()
                } else if (child.name.includes('fan')) {
                    game.fans.push(child);
                }
            } else {
                if (child.name.includes('Door-null')) {
                    door.trigger = child;
                    checkDoor();
                }
            }

            function checkDoor() {
                if (door.trigger !== null && door.proxy.length == 2 && door.doors.length == 2) {
                    game.doors.push(Object.assign({}, door));
                    door = { trigger: null, proxy: [], doors: [] };
                }
            }
        });

        game.loadUSB(loader);
    }, null, this.onError);
}

createDummyEnvironment() {
    const env = new THREE.Group();
    env.name = "Environment";
    this.scene.add(env);

    const geometry = new THREE.BoxBufferGeometry(150, 150, 150);
    const material = new THREE.MeshBasicMaterial({ color: 0xffff00 });

    for (let x = -1000; x < 1000; x += 300) {
        for (let z = -1000; z < 1000; z += 300) {
            const block = new THREE.Mesh(geometry, material);
```

```
            block.position.set(x, 75, z);
            env.add(block);
        }
    }

    this.environmentProxy = env;
}

playerControl(forward, turn) {
    //console.log(`playerControl(${forward}), ${turn}`);
    turn = -turn;

    if (forward == 0 && turn == 0) {
        delete this.player.move;
    } else {
        this.player.move = { forward, turn };
    }

    if (forward > 0) {
        if (this.player.action != 'walk' && this.player.action != 'run') this.action = 'walk';
    } else if (forward < -0.2) {
        if (this.player.action != 'walk') this.action = 'walk';
    } else {
        if (this.player.action == "walk" || this.player.action == 'run') this.action = 'look-around';
    }

}

createCameras() {
    const front = new THREE.Object3D();
    front.position.set(112, 100, 200);
    front.parent = this.player.object;
    const back = new THREE.Object3D();
    back.position.set(0, 100, -250);
    back.parent = this.player.object;
    const wide = new THREE.Object3D();
    wide.position.set(178, 139, 465);
    wide.parent = this.player.object;
    const overhead = new THREE.Object3D();
    overhead.position.set(0, 400, 0);
    overhead.parent = this.player.object;
    const collect = new THREE.Object3D();
    collect.position.set(40, 82, 94);
    collect.parent = this.player.object;
    this.player.cameras = { front, back, wide, overhead, collect };
    game.activeCamera = this.player.cameras.wide;
    game.cameraFade = 1;
    setTimeout(function() {
        game.activeCamera = game.player.cameras.back;
        game.cameraFade = 0.01;
        setTimeout(function() { game.cameraFade = 0.1; }, 1500);
    }, 2000)
}

loadNextAnim(loader) {
    let anim = this.anims.pop();
    const game = this;
    loader.load(`${this.assetsPath}fbx/${anim}.fbx`, function(object) {
        game.player[anim] = object.animations[0];
        if (anim == 'push-button') {
            game.player[anim].loop = false;
        }
        if (game.anims.length > 0) {
            game.loadNextAnim(loader);
        } else {
```

```
            delete game.anims;
            game.action = "look-around";
            game.initPlayerPosition();
            game.mode = game.modes.ACTIVE;
            const overlay = document.getElementById("overlay");
            overlay.classList.add("fade-in");
            overlay.addEventListener("animationend", function(evt) {
                evt.target.style.display = 'none';
            }, false);
        }
    }, null, this.onError);
}

initPlayerPosition() {
    //cast down
    const dir = new THREE.Vector3(0, -1, 0);
    const pos = this.player.object.position.clone();
    pos.y += 200;
    const raycaster = new THREE.Raycaster(pos, dir);
    const gravity = 30;
    const box = this.environmentProxy;

    const intersect = raycaster.intersectObject(box);
    if (intersect.length > 0) {
        this.player.object.position.y = pos.y - intersect[0].distance;
    }
}

getMousePosition(clientX, clientY) {
    const pos = new THREE.Vector2();
    pos.x = (clientX / this.renderer.domElement.clientWidth) * 2 - 1;
    pos.y = -(clientY / this.renderer.domElement.clientHeight) * 2 + 1;
    return pos;
}

showMessage(msg, fontSize = 20, onOK = null) {
    const txt = document.getElementById('message_text');
    txt.innerHTML = msg;
    txt.style.fontSize = fontSize + 'px';
    const btn = document.getElementById('message_ok');
    const panel = document.getElementById('message');
    const game = this;
    if (onOK != null) {
        btn.onclick = function() {
            panel.style.display = 'none';
            onOK.call(game);
        }
    } else {
        btn.onclick = function() {
            panel.style.display = 'none';
        }
    }
    panel.style.display = 'flex';
}

loadJSON(name, callback) {

    var xobj = new XMLHttpRequest();
    xobj.overrideMimeType("application/json");
    xobj.open('GET', `${name}.json`, true); // Replace 'my_data' with the path to your file
    xobj.onreadystatechange = function() {
        if (xobj.readyState == 4 && xobj.status == "200") {
            // Required use of an anonymous callback as .open will NOT return a value but simply returns undefined in
asynchronous mode
            callback(xobj.responseText);
```

```
        }
    };
    xobj.send(null);
}

onWindowResize() {
    this.camera.aspect = window.innerWidth / window.innerHeight;
    this.camera.updateProjectionMatrix();

    this.renderer.setSize(window.innerWidth, window.innerHeight);

}

set action(name) {
    if (this.player.action == name) return;
    const anim = this.player[name];
    const action = this.player.mixer.clipAction(anim, this.player.root);
    this.player.mixer.stopAllAction();
    this.player.action = name;
    action.timeScale = (name == 'walk' && this.player.move != undefined && this.player.move.forward < 0) ? -0.3 : 1;
    action.time = 0;
    action.fadeIn(0.5);
    if (name == 'push-button' || name == 'gather-objects') action.loop = THREE.LoopOnce;
    action.play();
    this.player.actionTime = Date.now();
}

movePlayer(dt) {
    const pos = this.player.object.position.clone();
    pos.y += 60;
    let dir = new THREE.Vector3();
    this.player.object.getWorldDirection(dir);
    if (this.player.move.forward < 0) dir.negate();
    let raycaster = new THREE.Raycaster(pos, dir);
    let blocked = false;
    const box = this.environmentProxy;

    if (this.environmentProxy != undefined) {
        const intersect = raycaster.intersectObject(box);
        if (intersect.length > 0) {
            if (intersect[0].distance < 50) blocked = true;
        }
    }

    if (!blocked) {
        if (this.player.move.forward > 0) {
            const speed = (this.player.action == 'run') ? 200 : 100;
            this.player.object.translateZ(dt * speed);
        } else {
            this.player.object.translateZ(-dt * 30);
        }
    }

    if (this.environmentProxy != undefined) {
        //cast left
        dir.set(-1, 0, 0);
        dir.applyMatrix4(this.player.object.matrix);
        dir.normalize();
        raycaster = new THREE.Raycaster(pos, dir);

        let intersect = raycaster.intersectObject(box);
        if (intersect.length > 0) {
            if (intersect[0].distance < 50) this.player.object.translateX(50 - intersect[0].distance);
        }
```

```
        //cast right
        dir.set(1, 0, 0);
        dir.applyMatrix4(this.player.object.matrix);
        dir.normalize();
        raycaster = new THREE.Raycaster(pos, dir);

        intersect = raycaster.intersectObject(box);
        if (intersect.length > 0) {
            if (intersect[0].distance < 50) this.player.object.translateX(intersect[0].distance - 50);
        }

        //cast down
        dir.set(0, -1, 0);
        pos.y += 200;
        raycaster = new THREE.Raycaster(pos, dir);
        const gravity = 30;

        intersect = raycaster.intersectObject(box);
        if (intersect.length > 0) {
            const targetY = pos.y - intersect[0].distance;
            if (targetY > this.player.object.position.y) {
                //Going up
                this.player.object.position.y = 0.8 * this.player.object.position.y + 0.2 * targetY;
                this.player.velocityY = 0;
            } else if (targetY < this.player.object.position.y) {
                //Falling
                if (this.player.velocityY == undefined) this.player.velocityY = 0;
                this.player.velocityY += dt * gravity;
                this.player.object.position.y -= this.player.velocityY;
                if (this.player.object.position.y < targetY) {
                    this.player.velocityY = 0;
                    this.player.object.position.y = targetY;
                }
            }
        }
    }
}

animate() {
    const game = this;
    const dt = this.clock.getDelta();

    requestAnimationFrame(function() { game.animate(); });

    if (this.tweens.length > 0) {
        this.tweens.forEach(function(tween) { tween.update(dt); });
    }

    if (this.player.mixer != undefined && this.mode == this.modes.ACTIVE) {
        this.player.mixer.update(dt);
    }

    if (this.player.action == 'walk') {
        const elapsedTime = Date.now() - this.player.actionTime;
        if (elapsedTime > 1000 && this.player.move.forward > 0) this.action = 'run';
    }
    if (this.player.move != undefined) {
        if (this.player.move.forward != 0) this.movePlayer(dt);
        this.player.object.rotateY(this.player.move.turn * dt);
    }

    if (this.player.cameras != undefined && this.player.cameras.active != undefined) {
        this.camera.position.lerp(this.player.cameras.active.getWorldPosition(new THREE.Vector3()), this.cameraFade);
        let pos;
        if (this.cameraTarget != undefined) {
```

```
            this.camera.position.copy(this.cameraTarget.position);
            pos = this.cameraTarget.target;
        } else {
            pos = this.player.object.position.clone();
            pos.y += 60;
        }
        this.camera.lookAt(pos);
    }

    this.actionBtn.style = 'display:none;';
    let trigger = false;

    if (this.doors !== undefined) {
        this.doors.forEach(function(door) {
            if (game.player.object.position.distanceTo(door.trigger.position) < 100) {
                game.actionBtn.style = 'display:block;';
                game.onAction = { action: 'push-button', mode: 'open-doors', index: 0 };
                trigger = true;
            }
        });
    }

    if (this.collect !== undefined && !trigger) {
        this.collect.forEach(function(object) {
            if (object.visible && game.player.object.position.distanceTo(object.position) < 100) {
                game.actionBtn.style = 'display:block;';
                game.onAction = { action: 'gather-objects', mode: 'collect', index: 0, src: "usb.jpg" };
                trigger = true;
            }
        });
    }

    if (!trigger) delete this.onAction;

    if (this.fans !== undefined) {
        let vol = 0;
        this.fans.forEach(function(fan) {
            const dist = fan.position.distanceTo(game.player.object.position);
            const tmpVol = 1 - dist / 1000;
            if (tmpVol > vol) vol = tmpVol;
            fan.rotateZ(dt);
        });
        this.sfx.fan.volume = vol;
    }

    this.renderer.render(this.scene, this.camera);

    if (this.stats != undefined) this.stats.update();

  }

  onError(error) {
    const msg =
      console.error(JSON.stringify(error));
    console.error(error.message);
  }
}

class Easing {
  // t: current time, b: begInnIng value, c: change In value, d: duration
  constructor(start, end, duration, startTime = 0, type = 'linear') {
    this.b = start;
    this.c = end - start;
    this.d = duration;
    this.type = type;
```

```
    this.startTime = startTime;
}

value(time) {
    this.t = time - this.startTime;
    return this[this.type]();
}

linear() {
    return this.c * (this.t / this.d) + this.b;
}

inQuad() {
    return this.c * (this.t /= this.d) * this.t + this.b;
}

outQuad() {
    return -this.c * (this.t /= this.d) * (this.t - 2) + this.b;
}

inOutQuad() {
    if ((this.t /= this.d / 2) < 1) return this.c / 2 * this.t * this.t + this.b;
    return -this.c / 2 * ((--this.t) * (this.t - 2) - 1) + this.b;
}

projectile() {
    let c = this.c;
    let b = this.b;
    let t = this.t;
    this.t *= 2;
    let result;
    let func;
    if (this.t < this.d) {
        result = this.outQuad();
        func = "outQuad";
    } else {
        this.t -= this.d;
        this.b += c;
        this.c = -c;
        result = this.inQuad();
        func = "inQuad";
    }
    console.log("projectile: " + result.toFixed(2) + " time:" + this.t.toFixed(2) + " func:" + func);
    this.b = b;
    this.c = c;
    this.t = t;
    return result;
}

inCubic() {
    return this.c * (this.t /= this.d) * this.t * this.t + this.b;
}

outCubic() {
    return this.c * ((this.t = this.t / this.d - 1) * this.t * this.t + 1) + this.b;
}

inOutCubic() {
    if ((this.t /= this.d / 2) < 1) return this.c / 2 * this.t * this.t * this.t + this.b;
    return this.c / 2 * ((this.t -= 2) * this.t * this.t + 2) + this.b;
}

inQuart() {
    return this.c * (this.t /= this.d) * this.t * this.t * this.t + this.b;
}
```

```
outQuart() {
    return -this.c * ((this.t = this.t / this.d - 1) * this.t * this.t * this.t - 1) + this.b;
}

inOutQuart() {
    if ((this.t /= this.d / 2) < 1) return this.c / 2 * this.t * this.t * this.t * this.t + this.b;
    return -this.c / 2 * ((this.t -= 2) * this.t * this.t * this.t - 2) + this.b;
}

inQuint() {
    return this.c * (this.t /= this.d) * this.t * this.t * this.t * this.t + this.b;
}

outQuint() {
    return this.c * ((this.t = this.t / this.d - 1) * this.t * this.t * this.t * this.t + 1) + this.b;
}

inOutQuint() {
    if ((this.t /= this.d / 2) < 1) return this.c / 2 * this.t * this.t * this.t * this.t * this.t + this.b;
    return this.c / 2 * ((this.t -= 2) * this.t * this.t * this.t * this.t + 2) + this.b;
}

inSine() {
    return -this.c * Math.cos(this.t / this.d * (Math.PI / 2)) + this.c + this.b;
}

outSine() {
    return this.c * Math.sin(this.t / this.d * (Math.PI / 2)) + this.b;
}

inOutSine() {
    return -this.c / 2 * (Math.cos(Math.PI * this.t / this.d) - 1) + this.b;
}

inExpo() {
    return (this.t == 0) ? this.b : this.c * Math.pow(2, 10 * (this.t / this.d - 1)) + this.b;
}

outExpo() {
    return (this.t == this.d) ? this.b + this.c : this.c * (-Math.pow(2, -10 * this.t / this.d) + 1) + this.b;
}

inOutExpo() {
    if (this.t == 0) return this.b;
    if (this.t == this.d) return this.b + this.c;
    if ((this.t /= this.d / 2) < 1) return this.c / 2 * Math.pow(2, 10 * (this.t - 1)) + this.b;
    return this.c / 2 * (-Math.pow(2, -10 * --this.t) + 2) + this.b;
}

inCirc() {
    return -this.c * (Math.sqrt(1 - (this.t /= this.d) * this.t) - 1) + this.b;
}

outCirc() {
    return this.c * Math.sqrt(1 - (this.t = this.t / this.d - 1) * this.t) + this.b;
}

inOutCirc() {
    if ((this.t /= this.d / 2) < 1) return -this.c / 2 * (Math.sqrt(1 - this.t * this.t) - 1) + this.b;
    return this.c / 2 * (Math.sqrt(1 - (this.t -= 2) * this.t) + 1) + this.b;
}

inElastic() {
    let s = 1.70158,
        p = 0,
```

```
        a = this.c;
    if (this.t == 0) return this.b;
    if ((this.t /= this.d) == 1) return this.b + this.c;
    if (!p) p = this.d * .3;
    if (a < Math.abs(this.c)) { a = this.c; let s = p / 4; } else { let s = p / (2 * Math.PI) * Math.asin(this.c / a) };
    return -(a * Math.pow(2, 10 * (this.t -= 1)) * Math.sin((this.t * this.d - s) * (2 * Math.PI) / p)) + this.b;
}

outElastic() {
    let s = 1.70158,
        p = 0,
        a = this.c;
    if (this.t == 0) return this.b;
    if ((this.t /= this.d) == 1) return this.b + this.c;
    if (!p) p = this.d * .3;
    if (a < Math.abs(this.c)) { a = this.c; let s = p / 4; } else { let s = p / (2 * Math.PI) * Math.asin(this.c / a) };
    return a * Math.pow(2, -10 * this.t) * Math.sin((this.t * this.d - s) * (2 * Math.PI) / p) + this.c + this.b;
}

inOutElastic() {
    let s = 1.70158,
        p = 0,
        a = this.c;
    if (this.t == 0) return this.b;
    if ((this.t /= this.d / 2) == 2) return this.b + this.c;
    if (!p) p = this.d * (.3 * 1.5);
    if (a < Math.abs(this.c)) { a = this.c; let s = p / 4; } else { let s = p / (2 * Math.PI) * Math.asin(this.c / a) };
    if (this.t < 1) return -.5 * (a * Math.pow(2, 10 * (this.t -= 1)) * Math.sin((this.t * this.d - s) * (2 * Math.PI) / p)) + this.b;
    return a * Math.pow(2, -10 * (this.t -= 1)) * Math.sin((this.t * this.d - s) * (2 * Math.PI) / p) * .5 + this.c + this.b;
}

inBack() {
    let s = 1.70158;
    return this.c * (this.t /= this.d) * this.t * ((s + 1) * this.t - s) + this.b;
}

outBack() {
    let s = 1.70158;
    return this.c * ((this.t = this.t / this.d - 1) * this.t * ((s + 1) * this.t + s) + 1) + this.b;
}

inOutBack() {
    let s = 1.70158;
    if ((this.t /= this.d / 2) < 1) return this.c / 2 * (this.t * this.t * (((s *= (1.525)) + 1) * this.t - s)) + this.b;
    return this.c / 2 * ((this.t -= 2) * this.t * (((s *= (1.525)) + 1) * this.t + s) + 2) + this.b;
}

inBounce(t = this.t, b = this.b) {
    return this.c - this.outBounce(this.d - t, 0) + b;
}

outBounce(t = this.t, b = this.b) {
    if ((t /= this.d) < (1 / 2.75)) {
        return this.c * (7.5625 * t * t) + b;
    } else if (t < (2 / 2.75)) {
        return this.c * (7.5625 * (t -= (1.5 / 2.75)) * t + .75) + b;
    } else if (t < (2.5 / 2.75)) {
        return this.c * (7.5625 * (t -= (2.25 / 2.75)) * t + .9375) + b;
    } else {
        return this.c * (7.5625 * (t -= (2.625 / 2.75)) * t + .984375) + b;
    }
}

inOutBounce() {
    if (this.t < this.d / 2) return this.inBounce(this.t * 2, 0) * .5 + this.b;
```

```
        return this.outBounce(this.t * 2 - this.d, 0) * .5 + this.c * .5 + this.b;
    }
}

class Tween {
    constructor(target, channel, endValue, duration, oncomplete, easing = "inOutQuad") {
        this.target = target;
        this.channel = channel;
        this.oncomplete = oncomplete;
        this.endValue = endValue;
        this.duration = duration;
        this.currentTime = 0;
        this.finished = false;
        //constructor(start, end, duration, startTime=0, type='linear')
        this.easing = new Easing(target[channel], endValue, duration, 0, easing);
    }

    update(dt) {
        if (this.finished) return;
        this.currentTime += dt;
        if (this.currentTime >= this.duration) {
            this.target[this.channel] = this.endValue;
            if (this.oncomplete) this.oncomplete();
            this.finished = true;
        } else {
            this.target[this.channel] = this.easing.value(this.currentTime);
        }
    }
}

class SFX {
    constructor(options) {
        this.context = options.context;
        const volume = (options.volume != undefined) ? options.volume : 1.0;
        this.gainNode = this.context.createGain();
        this.gainNode.gain.setValueAtTime(volume, this.context.currentTime);
        this.gainNode.connect(this.context.destination);
        this._loop = (options.loop == undefined) ? false : options.loop;
        this.fadeDuration = (options.fadeDuration == undefined) ? 0.5 : options.fadeDuration;
        this.autoplay = (options.autoplay == undefined) ? false : options.autoplay;
        this.buffer = null;

        let codec;
        for (let prop in options.src) {
            if (SFX.supportsAudioType(prop)) {
                codec = prop;
                break;
            }
        }

        if (codec != undefined) {
            this.url = options.src[codec];
            this.load(this.url);
        } else {
            console.warn("Browser does not support any of the supplied audio files");
        }
    }

    static supportsAudioType(type) {
        let audio;

        // Allow user to create shortcuts, i.e. just "mp3"
        let formats = {
            mp3: 'audio/mpeg',
            wav: 'audio/wav',
```

```
        aif: 'audio/x-aiff',
        ogg: 'audio/ogg'
    };

    if (!audio) audio = document.createElement('audio');

    return audio.canPlayType(formats[type] || type);
}

load(url) {
    // Load buffer asynchronously
    const request = new XMLHttpRequest();
    request.open("GET", url, true);
    request.responseType = "arraybuffer";

    const sfx = this;

    request.onload = function() {
        // Asynchronously decode the audio file data in request.response
        sfx.context.decodeAudioData(
            request.response,
            function(buffer) {
                if (!buffer) {
                    console.error('error decoding file data: ' + sfx.url);
                    return;
                }
                sfx.buffer = buffer;
                if (sfx.autoplay) sfx.play();
            },
            function(error) {
                console.error('decodeAudioData error', error);
            }
        );
    }

    request.onerror = function() {
        console.error('SFX Loader: XHR error');
    }

    request.send();
}

set loop(value) {
    this._loop = value;
    if (this.source != undefined) this.source.loop = value;
}

play() {
    if (this.buffer == null) return;
    if (this.source != undefined) this.source.stop();
    this.source = this.context.createBufferSource();
    this.source.loop = this._loop;
    this.source.buffer = this.buffer;
    this.source.connect(this.gainNode);
    this.source.start(0);
}

set volume(value) {
    this._volume = value;
    this.gainNode.gain.setTargetAtTime(value, this.context.currentTime + this.fadeDuration, 0);
}

pause() {
    if (this.source == undefined) return;
    this.source.stop();
```

```
    }

    stop() {
        if (this.source == undefined) return;
        this.source.stop();
        delete this.source;
    }
}

class JoyStick {
    constructor(options) {
        const circle = document.createElement("div");
        circle.style.cssText = "position:absolute; bottom:35px; width:80px; height:80px; background:rgba(126, 126, 126, 0.5);
border:#fff solid medium; border-radius:50%; left:50%; transform:translateX(-50%);";
        const thumb = document.createElement("div");
        thumb.style.cssText = "position: absolute; left: 20px; top: 20px; width: 40px; height: 40px; border-radius: 50%; background:
#fff;";
        circle.appendChild(thumb);
        document.body.appendChild(circle);
        this.domElement = thumb;
        this.maxRadius = options.maxRadius || 40;
        this.maxRadiusSquared = this.maxRadius * this.maxRadius;
        this.onMove = options.onMove;
        this.game = options.game;
        this.origin = { left: this.domElement.offsetLeft, top: this.domElement.offsetTop };

        if (this.domElement != undefined) {
            const joystick = this;
            if ('ontouchstart' in window) {
                this.domElement.addEventListener('touchstart', function(evt) { joystick.tap(evt); });
            } else {
                this.domElement.addEventListener('mousedown', function(evt) { joystick.tap(evt); });
            }
        }
    }

    getMousePosition(evt) {
        let clientX = evt.targetTouches ? evt.targetTouches[0].pageX : evt.clientX;
        let clientY = evt.targetTouches ? evt.targetTouches[0].pageY : evt.clientY;
        return { x: clientX, y: clientY };
    }

    tap(evt) {
        evt = evt || window.event;
        // get the mouse cursor position at startup:
        this.offset = this.getMousePosition(evt);
        const joystick = this;
        if ('ontouchstart' in window) {
            document.ontouchmove = function(evt) { joystick.move(evt); };
            document.ontouchend = function(evt) { joystick.up(evt); };
        } else {
            document.onmousemove = function(evt) { joystick.move(evt); };
            document.onmouseup = function(evt) { joystick.up(evt); };
        }
    }

    move(evt) {
        evt = evt || window.event;
        const mouse = this.getMousePosition(evt);
        // calculate the new cursor position:
        let left = mouse.x - this.offset.x;
        let top = mouse.y - this.offset.y;
        //this.offset = mouse;

        const sqMag = left * left + top * top;
```

```
      if (sqMag > this.maxRadiusSquared) {
        //Only use sqrt if essential
        const magnitude = Math.sqrt(sqMag);
        left /= magnitude;
        top /= magnitude;
        left *= this.maxRadius;
        top *= this.maxRadius;
      }

      // set the element's new position:
      this.domElement.style.top = `${top + this.domElement.clientHeight/2}px`;
      this.domElement.style.left = `${left + this.domElement.clientWidth/2}px`;

      const forward = -(top - this.origin.top + this.domElement.clientHeight / 2) / this.maxRadius;
      const turn = (left - this.origin.left + this.domElement.clientWidth / 2) / this.maxRadius;

      if (this.onMove != undefined) this.onMove.call(this.game, forward, turn);
    }

    up(evt) {
      if ('ontouchstart' in window) {
        document.ontouchmove = null;
        document.touchend = null;
      } else {
        document.onmousemove = null;
        document.onmouseup = null;
      }
      this.domElement.style.top = `${this.origin.top}px`;
      this.domElement.style.left = `${this.origin.left}px`;

      this.onMove.call(this.game, 0, 0);
    }
}

class Preloader {
    constructor(options) {
      this.assets = {};
      for (let asset of options.assets) {
        this.assets[asset] = { loaded: 0, complete: false };
        this.load(asset);
      }
      this.container = options.container;

      if (options.onprogress == undefined) {
        this.onprogress = onprogress;
        this.domElement = document.createElement("div");
        this.domElement.style.position = 'absolute';
        this.domElement.style.top = '0';
        this.domElement.style.left = '0';
        this.domElement.style.width = '100%';
        this.domElement.style.height = '100%';
        this.domElement.style.background = '#000';
        this.domElement.style.opacity = '0.7';
        this.domElement.style.display = 'flex';
        this.domElement.style.alignItems = 'center';
        this.domElement.style.justifyContent = 'center';
        this.domElement.style.zIndex = '1111';
        const barBase = document.createElement("div");
        barBase.style.background = '#aaa';
        barBase.style.width = '50%';
        barBase.style.minWidth = '250px';
        barBase.style.borderRadius = '10px';
        barBase.style.height = '15px';
        this.domElement.appendChild(barBase);
        const bar = document.createElement("div");
```

```
        bar.style.background = '#2a2';
        bar.style.width = '50%';
        bar.style.borderRadius = '10px';
        bar.style.height = '100%';
        bar.style.width = '0';
        barBase.appendChild(bar);
        this.progressBar = bar;
        if (this.container != undefined) {
            this.container.appendChild(this.domElement);
        } else {
            document.body.appendChild(this.domElement);
        }
    } else {
        this.onprogress = options.onprogress;
    }

    this.oncomplete = options.oncomplete;

    const loader = this;

    function onprogress(delta) {
        const progress = delta * 100;
        loader.progressBar.style.width = `${progress}%`;
    }
}

checkCompleted() {
    for (let prop in this.assets) {
        const asset = this.assets[prop];
        if (!asset.complete) return false;
    }
    return true;
}

get progress() {
    let total = 0;
    let loaded = 0;

    for (let prop in this.assets) {
        const asset = this.assets[prop];
        if (asset.total == undefined) {
            loaded = 0;
            break;
        }
        loaded += asset.loaded;
        total += asset.total;
    }

    return loaded / total;
}

load(url) {
    const loader = this;
    var xobj = new XMLHttpRequest();
    xobj.overrideMimeType("application/json");
    xobj.open('GET', url, true);
    xobj.onreadystatechange = function() {
        if (xobj.readyState == 4 && xobj.status == "200") {
            loader.assets[url].complete = true;
            if (loader.checkCompleted()) {
                if (loader.domElement != undefined) {
                    if (loader.container != undefined) {
                        loader.container.removeChild(loader.domElement);
                    } else {
                        document.body.removeChild(loader.domElement);
```

```
                }
            }
            loader.oncomplete();
        }
    }
};
xobj.onprogress = function(e) {
    const asset = loader.assets[url];
    asset.loaded = e.loaded;
    asset.total = e.total;
    loader.onprogress(loader.progress);
}
xobj.send(null);
}
}
```