CSE378: Computer Graphics

Project – Treasure Hunting

Names:

Ahmed Mohamed Ahmed Aly

Marawan Osama Mohamed Sherif

Ids: 18P9313, 18P1416, UEL

GitHub Link:

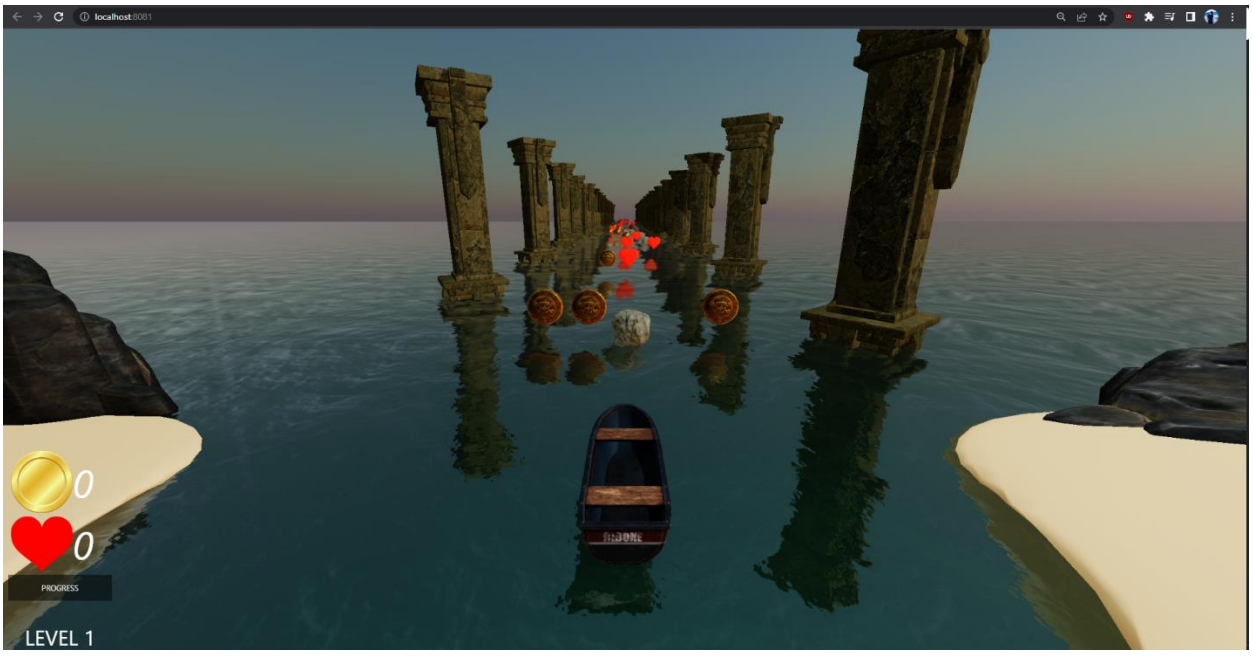
Link

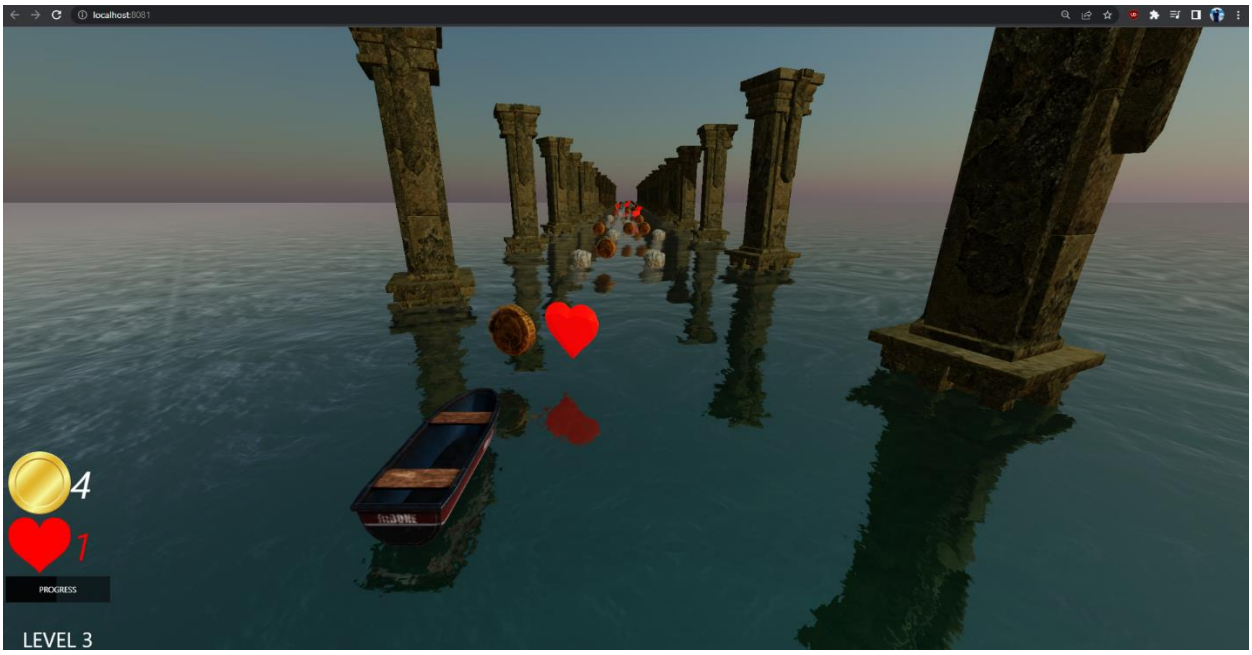
Demo:

Video

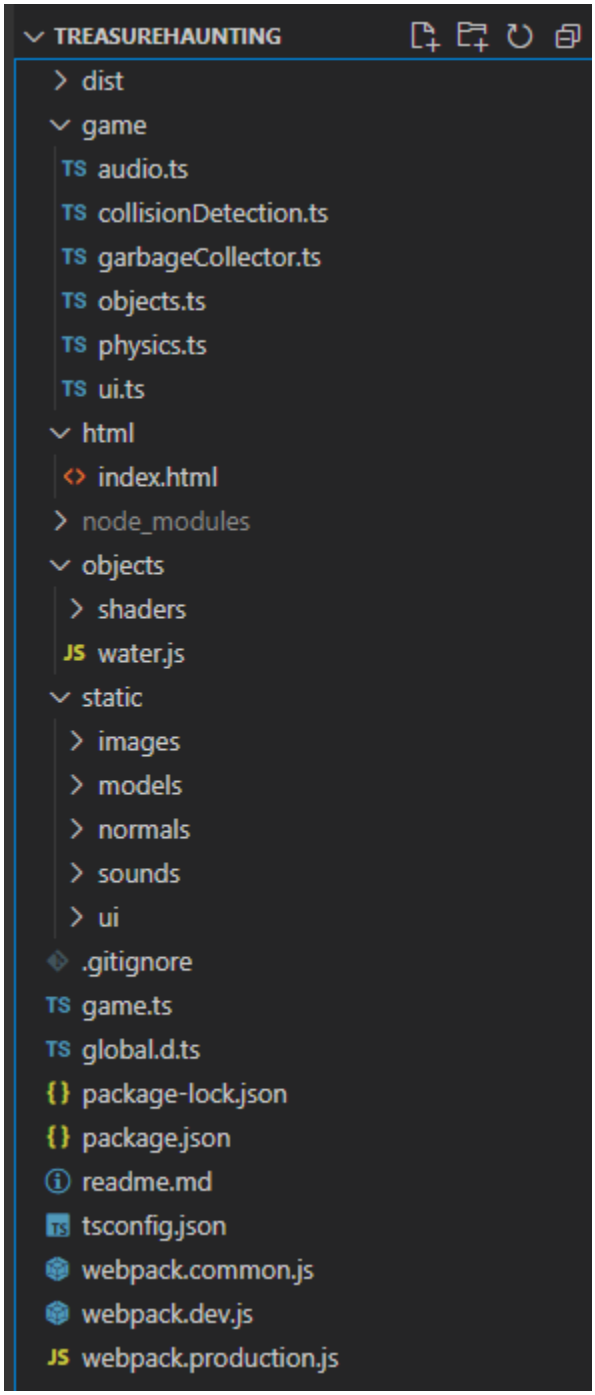
Snippets:


localhost:8081


localhost:8081
0
0
PROGRESS
LEVEL 1


localhost:8081
4
1
PROGRESS
LEVEL 3

Files hierarchy



Breaking down the code

Treasure Hunting game project code consists of:

- A 3D scene using the three.js library and renders it using a WebGL renderer. The scene includes complex 3D gltf models that include box-shaped accurate collisions, lighting, reflections, water, a skybox, particle effects and shaders.
- Scene is set with the skybox using the following code

```
import {Sky} from "three/examples/jsm/objects/Sky";
const sky = new Sky();
sky.scale.setScalar(10000);
scene.add(sky);
```

- Then the lighting was added as a singular light source (the sun) and its code was this

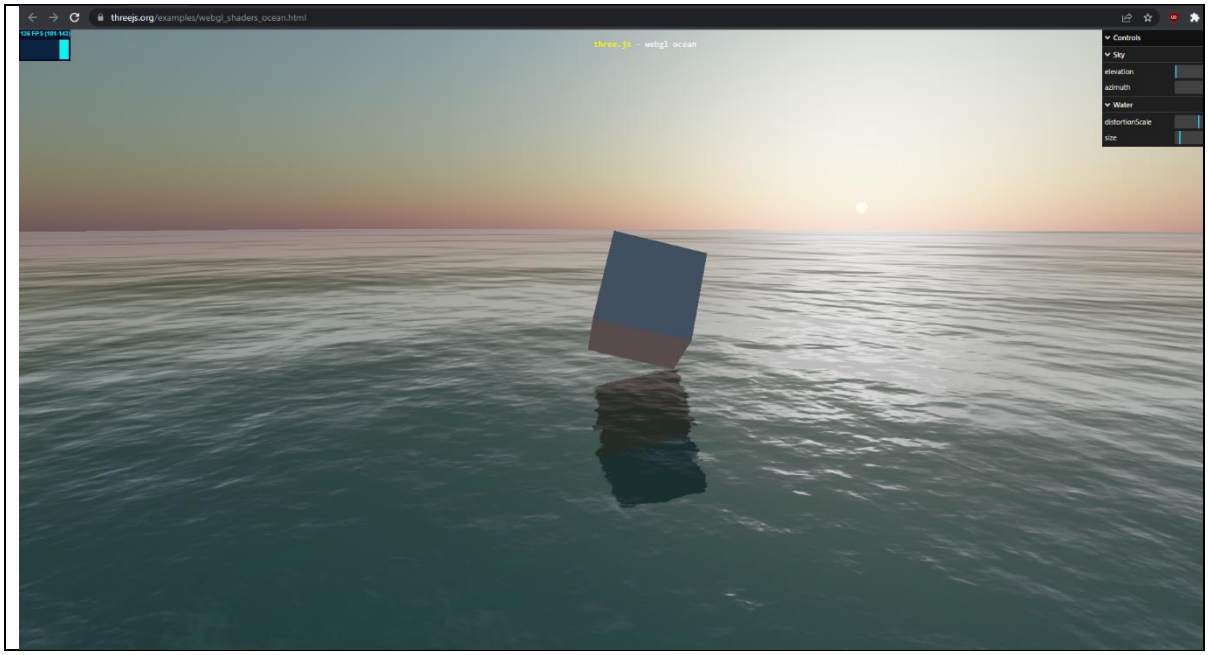
```
const sun = new Vector3();
const parameters = {
        elevation: 3,
        azimuth: 115
    };

    const pmremGenerator = new PMREMGenerator(renderer);

    const phi = MathUtils.degToRad(90- parameters.elevation);
    const theta = MathUtils.degToRad(180-parameters.azimuth);

    sun.setFromSphericalCoords(1, phi, theta);
```

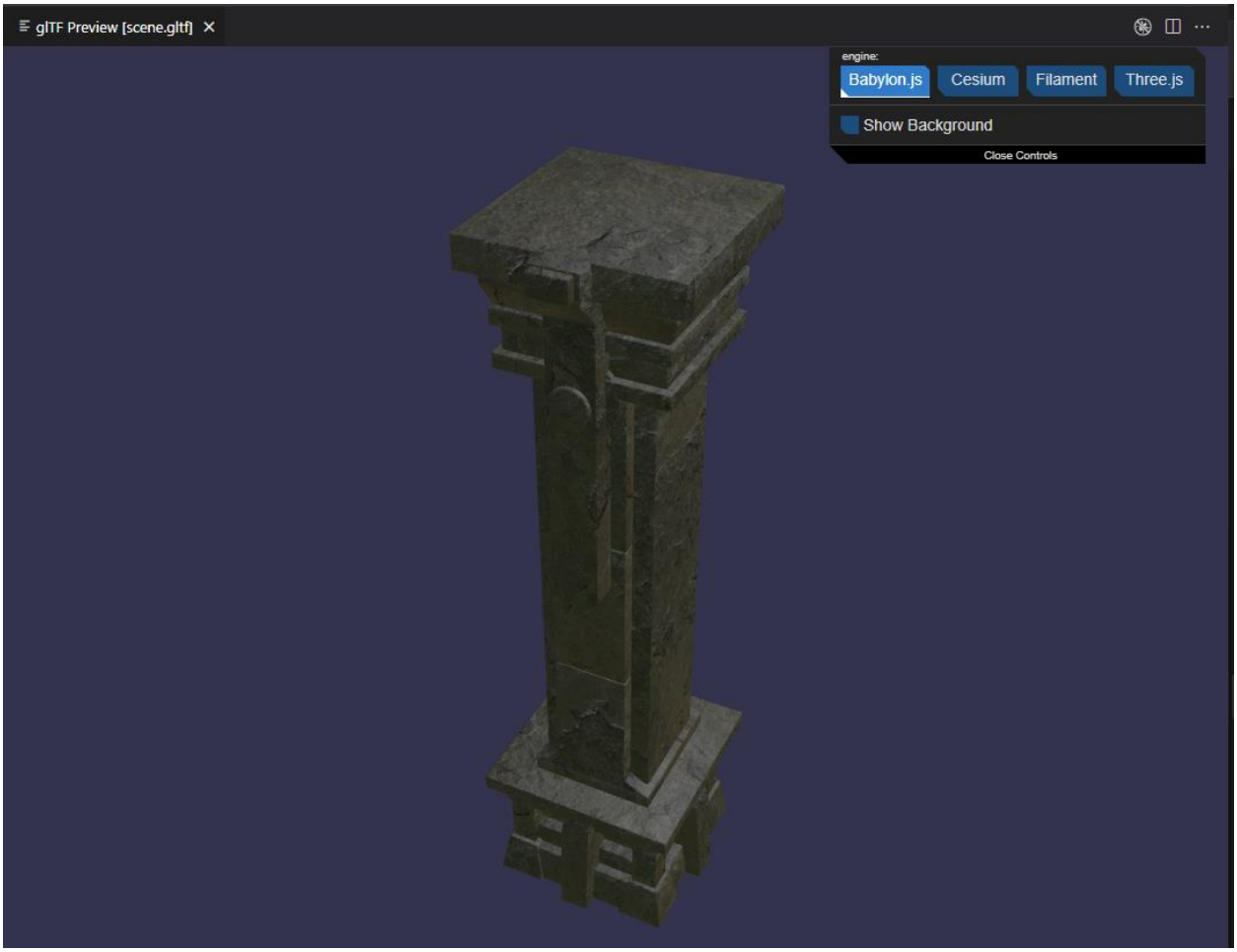
- Then the water code is added using the three.js example reference from their examples list. It looked like this



- Then we started adding the 3D gltf models beginning with the columns on the right and left side of the boat as it travels.

```
export let columnsModel: Object3D;
const columnsGLTF = 'static/models/columns/scene.gltf';
columnsModel = (await gltfLoader.loadAsync(columnsGLTF)).scene.children[0];
```

Which looked like this:



Then to add it to the left and right of the boat and keep generating

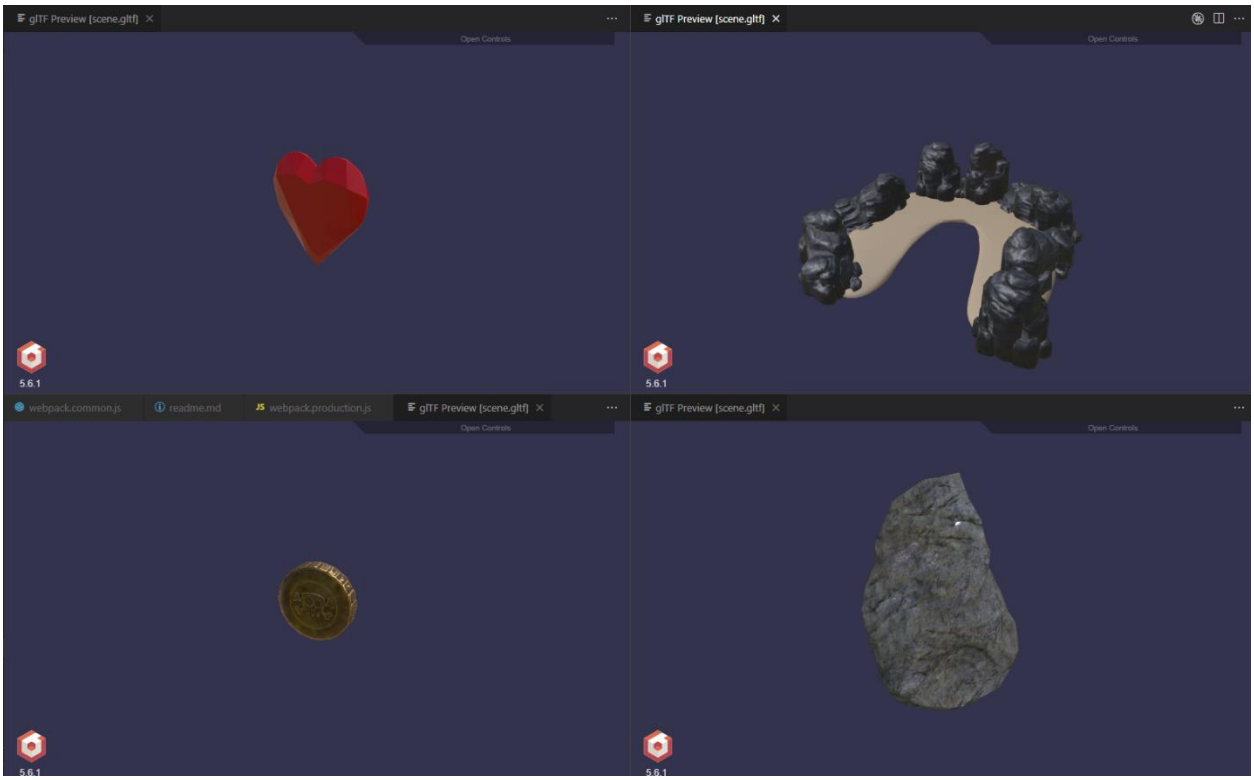
```
export const addBackgroundBit = (count: number, horizonSpawn: boolean = false)
=> {
    // Work out how far away this background bit should be
    let zOffset = (horizonSpawn ? -1400 : -(60 * count));
    // Create a copy of our original column model
    let thisColumn = columnsModel.clone();
    // Set the scale appropriately for the scene
    thisColumn.scale.set(1, 1, 1);
    // Set the position of the columns one to the left and the other to the
right
    thisColumn.position.set(count % 2 == 0 ? 60 - Math.random() : -60 -
Math.random(), -10, zOffset);
    // Rotate the column to a better angle
    thisColumn.rotation.set(MathUtils.degToRad(-90), 0, Math.random());
    // Finally, add the column to the scene
    scene.add(thisColumn);
```

```
    // Add the column to the beginning of the environmentBits array to keep
track of them (so we can clean up later)
    environmentBits.unshift(thisColumn);// add to beginning of array
}
```

- Then the boat's model was added

```
export let boatModel: Object3D;
const boatGLTF = 'static/models/boat/scene.gltf';
boatModel = (await gltfLoader.loadAsync(boatGLTF)).scene.children[0];
// Set the appropriate scale for our boat model
    boatModel.scale.set(0.3, 0.3, 0.3);
    scene.add(boatModel);
```

- All the other objects (hearts, rocks, coins and the starting bay) were added using the same method



- We then add a “challenge row”. It consists of a mixture of 5 random elements, either a heart, a rock or a coin. The row could have any mixture of these 3 objects and empty spaces.

```
export const addChallengeRow = (count: number, horizonSpawn: boolean = false)
=> {
    // Work out how far away this challenge row should be
    let zOffset = (horizonSpawn ? -1400 : -(count * 60));
    // Create a Group for the objects
    let rowGroup = new Group();
    rowGroup.position.z = zOffset;
    for (let i = 0; i < 5; i++) {

        const random = Math.random() * 10;

        // If it's less than 2, create a coin
        if (random < 2) {
            let coin = addCoin(i);
            coin.name = 'coin';
            rowGroup.add(coin);
```

```
        }
        // If it's less than 4, spawn a rock
        else if (random < 4) {
            let rock = addRock(i);
            rock.name = 'rock';
            rowGroup.add(rock);
        }
       // but if it's more than 9, spawn a life
        else if (random > 9) {
            let life = addLife(i);
            life.name = 'life';
            rowGroup.add(life);
        }
    }
    // Add the row to the challengeRows array to keep track of it, and so we
can clean them up later
    challengeRows.unshift({rowParent: rowGroup, index:
sceneConfiguration.challengeRowCount++});
    // add the row to the scene
    scene.add(rowGroup);
}
```

- Then we created the functionality for the UI elements

```
import {sceneConfiguration, sceneSetup} from "../game";
import { seaAudioLoader } from "./audio";

export const coinUiElement = document.getElementById('coinCount')!;
export const lifeUiElement = document.getElementById('lifeCount')!;
export const progressUiElement = document.getElementById('courseProgress')!;

export const endLevelDescriptor = document.getElementById('levelDescriptor')!;
export const endLevelBoatStatus = document.getElementById('boatStatus')!;

export const nextLevelButton = document.getElementById('nextLevel')!;

export const startAgainButton = document.getElementById('startOver')!;
export const startGameButton = document.getElementById('startGame')!;
export const enableSoundsButton = document.getElementById('enableSounds')!;

export const startPanel = document.getElementById('levelStartPanel')!;


export const uiInit = () => {
    startAgainButton.onclick = () => {
        nextLevel(true);
    }
```

```
    enableSoundsButton.onclick = () => {
        if (!sceneConfiguration.soundEnabled) {

            sceneConfiguration.soundEnabled = true;
            seaAudioLoader();
            enableSoundsButton.classList.add('hidden');
        }
    }
}

export const nextLevel = (reset: boolean = false) => {

    document.getElementById('endOfLevel')!.style!.display = '';
    document.getElementById('endOfLevel')!.classList.remove('fadeOut');
    document.getElementById('endOfLevel')!.classList.add('hidden');
    document.getElementById('startGame')!.classList.remove('hidden');
    document.getElementById('levelStartPanel')!.classList.remove('hidden');

    sceneConfiguration.cameraStartAnimationPlaying = false;
    sceneConfiguration.boatMoving = false;
    sceneConfiguration.speed = 0.05;
    sceneConfiguration.speed = sceneConfiguration.level * 0.1;
    if (reset) {
        sceneSetup(1)
    } else {
        sceneSetup(++sceneConfiguration.level);
    }
}

export const updateLevelEndUI = (damaged: boolean) => {
    endLevelDescriptor.innerText = `LEVEL ${sceneConfiguration.level}`;
    if (damaged) {
        endLevelBoatStatus.innerText = '';
        endLevelBoatStatus.innerText = 'NOOOO Captain!\r\n\r\n Your boat has
sank, you hit too many rocks !';
        nextLevelButton.classList.add('hidden');
        startAgainButton.classList.remove('hidden');
    } else {

        let lifeCount = sceneConfiguration.data.lifesCollected;
        console.log(lifeCount);
        endLevelBoatStatus.innerText = '';
        if (lifeCount >= 5) {
            endLevelBoatStatus.innerText = 'NICE Captain! \r\n\r\nYour boat is
in a perfect condition!';
        } else if (lifeCount >= 1 && lifeCount < 5) {
```

```
            endLevelBoatStatus.innerText = 'GOOD JOB Captain! \r\n\r\nYour boat
is in a decent condition.';
        } else if (lifeCount == 0) {
            endLevelBoatStatus.innerText = 'No lifes in the sea? \r\n\r\nYour
ship is in the same condition as when you left!';
        } else if (lifeCount < 0) {
            endLevelBoatStatus.innerText = 'BECAREFULL Captain! \r\n\r\nYour
ship is almost sank. Try to hit less rocks next time!';
        }

        nextLevelButton.classList.remove('hidden');
        startAgainButton.classList.add('hidden');

    }
}

export const showLevelEndScreen = () => {

    document.getElementById('endOfLevel')!.style!.display = 'flex';
    document.getElementById('endOfLevel')!.classList.add('fadeOut');
    document.getElementById('coinCountLevelEnd')!.innerText =
String(sceneConfiguration.data.coinsCollected);


}
export const setProgress = (progress: string) => {
    let progressElement = document.getElementById('loadingProgress');
    if (progressElement != null) {
        progressElement.innerText = progress;
    }
}
```

- Then we added collision of the boat to the objects in the challenge rows. Each model has a box around it. If two boxes were to get in the proximity of each other, then a collision would occur. Hearts and coins would increase with collisions with their respective models, and hitting a rock would decrease the hearts.

```
export const detectCollisions = () => {
    // If the level is over, don't detect collisions
    if (sceneConfiguration.levelOver) return;
    // Using the dimensions of our boat, create a box that is the width and
height of our model
    // This box doesn't appear in the world, it's merely a set of coordinates
that describe the box
```

```
    // in world space.
    const boatBox = new Box3().setFromObject(boatModel);
    // For every challange row that we have on the screen...
    challengeRows.forEach(x => {
    // ...update the global position matrix of the row, and its children.
    x.rowParent.updateMatrixWorld();
    // Next, for each object within each challenge row...
    x.rowParent.children.forEach(y => {
    y.children.forEach(z => {
    // ...create a box that is the width and height of the object
    const box = new Box3().setFromObject(z);
    // Check if the box with the obstacle overlaps (or intersects with) our
boat
    if (box.intersectsBox(boatBox)) {
    // If it does, get the center position of that box
    let destructionPosition = box.getCenter(z.position);
    // Queue up the destruction animation to play (the boxes flying out from
the boat)
    playDestructionAnimation(destructionPosition);
    // Remove the object that has been hit from the parent
    // This removes the object from the scene
    y.remove(z);
    // Now, we check what it was that we hit, whether it was a rock, shield, or
coin
    if (y.userData.objectType !== undefined) {
    let type = y.userData.objectType as ObjectType;
    switch (type) {
    // If it was a rock...
    case ObjectType.ROCK:
        // ...play the rock sound
        if (sceneConfiguration.soundEnabled){
            rockAudioLoader();
        }
        // ...remove one shield from the players' score
        sceneConfiguration.data.lifesCollected--;
        // Update the UI with the new count of shields
        lifeUiElement.innerText =
String(sceneConfiguration.data.lifesCollected);
        // If the player has less than 0 shields...
        if (sceneConfiguration.data.lifesCollected <= 0) {
            // ...add the 'danger' CSS class to make the text red (if it's not
already there)
            if (!lifeUiElement.classList.contains('danger')) {
                lifeUiElement.classList.add('danger');
            }
```

```
        } else { //Otherwise, if it's more than 0 shields, remove the danger
CSS class
            // so the text goes back to being white
            lifeUiElement.classList.remove('danger');
        }

        // If the ship has sustained too much damage, and has less than -5
shields...
        if (sceneConfiguration.data.lifesCollected <= -5) {
            // ...end the scene
            endLevel(true);
        }
        break;
    // If it's a coin...
    case ObjectType.COIN:
        // ...play the coin sound
        if(sceneConfiguration.soundEnabled){
        coinAudioLoader();
        }
        // Update the UI with the new count of coins, and increment the count
of
        // currently collected coins
        coinUiElement.innerText =
String(++sceneConfiguration.data.coinsCollected);
        break;
    // If it's a shield...
    case ObjectType.LIFE:
        // ...play the heart sound
        if(sceneConfiguration.soundEnabled){
        heartAudioLoader();
        }
        // Update the UI with the new count of shields, and increment the count
of
        // currently collected shields
        lifeUiElement.innerText =
String(++sceneConfiguration.data.lifesCollected);
        break;
    }
    }
    }
    });
    })
    });
    }
```

- Then we coded in the particle effects for the collecting animation (collecting hearts and coins or hitting rocks)

```
const playDestructionAnimation = (spawnPosition: Vector3) => {

    // Create six boxes
    for (let i = 0; i < 6; i++) {
        // Our destruction 'bits' will be black, but have some transparency to
them
        let destructionBit = new Mesh(new BoxGeometry(1, 1, 1), new
MeshBasicMaterial({
            color: 'black',
            transparent: true,
            opacity: 0.4
        }));

        // Each destruction bit object within the scene will have a 'lifetime'
property associated to it
        // This property is incremented every time a frame is drawn to the
screen
        // Within our animate loop, we check if this is more than 500, and if
it is, we remove the object
        destructionBit.userData.lifetime = 0;
        // Set the spawn position of the box
        destructionBit.position.set(spawnPosition.x, spawnPosition.y,
spawnPosition.z);
        // Create an animation mixer for the object
        destructionBit.userData.mixer = new AnimationMixer(destructionBit);

        // Spawn the objects in a circle around the boat
        let degrees = i / 45;

        // Work out where on the circle we should spawn this specific
destruction bit
        let spawnX = Math.cos(radToDeg(degrees)) * 15;
        let spawnY = Math.sin(radToDeg(degrees)) * 15;

        // Create a VectorKeyFrameTrack that will animate this box from its
starting position to the final
        // 'outward' position (so it looks like the boxes are exploding from
the ship)
        let track = new VectorKeyframeTrack('.position', [0, 0.3], [
            boatModel.position.x, // x 1
            boatModel.position.y, // y 1
            boatModel.position.z, // z 1
            boatModel.position.x + spawnX, // x 2
            boatModel.position.y, // y 2
```

```
            boatModel.position.z + spawnY, // z 2
        ]);

        // Create an animation clip with our VectorKeyFrameTrack
        const animationClip = new AnimationClip('animateIn', 10, [track]);
        const animationAction =
destructionBit.userData.mixer.clipAction(animationClip);

        // Only play the animation once
        animationAction.setLoop(LoopOnce, 1);

        // When complete, leave the objects in their final position (don't
reset them to the starting position)
        animationAction.clampWhenFinished = true;
        // Play the animation
        animationAction.play();
        // Associate a Clock to the destruction bit. We use this within the
render loop so ThreeJS knows how far
        // to move this object for this frame
        destructionBit.userData.clock = new Clock();
        // Add the destruction bit to the scene
        scene.add(destructionBit);

        // Add the destruction bit to an array, to keep track of them
        destructionBits.push(destructionBit);
    }
}
```

- We then coded in the garbage collection, which removed the columns and items that the boat passes to save on memory, resources and not make the game have a low frame per second count.

```
import {scene} from "../game";
import {challengeRows, environmentBits} from "./objects";

export const garbageCollector = () => {
    let environmentObjectsForCollection = environmentBits.filter(x =>
x.position.z > 100);
    let challengeRowsForCollection = challengeRows.filter(x =>
x.rowParent.position.z > 100);

    for (let i = 0; i < environmentObjectsForCollection.length - 1; i++) {
```

```
        let environmentObjectIndex =
environmentBits.indexOf(environmentObjectsForCollection[i]);
        scene.remove(environmentBits[environmentObjectIndex]);
        environmentBits.splice(environmentObjectIndex, 1);

        console.log(Removing environment object at index ${i} from scene);
    }

    for (let i = 0; i < challengeRowsForCollection.length - 1; i++) {
        let challengeRowIndex =
challengeRows.indexOf(challengeRowsForCollection[i]);
        scene.remove(challengeRowsForCollection[i].rowParent);

        console.log(Removing challenge line at index ${i} from scene);

        challengeRows.splice(challengeRowIndex, 1);

    }
}
```

- Lastly, the last of the assets that were added was the audio. We added audio for the water, collecting items and hitting rocks.

```
import { AudioListener, Audio, AudioLoader  } from "three";

export const seaListener = new AudioListener();
export const coinListener = new AudioListener();
export const heartListener = new AudioListener();
export const rockListener = new AudioListener();

const coinSound = new Audio( coinListener );
const seaSound = new Audio( seaListener );
const heartSound = new Audio( heartListener );
const rockSound = new Audio( rockListener );


export const coinAudioLoader = () => {
    const coinAudioLoader = new AudioLoader();
    coinAudioLoader.load( 'static/sounds/coinSound.mp3', function( buffer ) {
    coinSound.setBuffer( buffer );
    coinSound.setVolume( 0.8 );
    coinSound.play();
});
}

export const heartAudioLoader = () => {
    const heartAudioLoader = new AudioLoader();
```

```
    heartAudioLoader.load( 'static/sounds/heartSound.mp3', function( buffer ) {
    heartSound.setBuffer( buffer );
    heartSound.setVolume( 0.8 );
    heartSound.play();
});
}

export const rockAudioLoader = () => {
    const rockAudioLoader = new AudioLoader();
    rockAudioLoader.load( 'static/sounds/rockSound.mp3', function( buffer ) {
        rockSound.setBuffer( buffer );
        rockSound.setVolume( 0.8 );
        rockSound.play();
});
}

export const seaAudioLoader = () => {
    const seaAudioLoader = new AudioLoader();
    seaAudioLoader.load( 'static/sounds/seaSound.mp3', function( buffer ) {
        seaSound.setBuffer( buffer );
        seaSound.setLoop( true );
        seaSound.setVolume( 0.3 );
        seaSound.play();
    });
}
```

Game.ts

- As for the logic of the main part of the project, this calls upon all the major flags for the logic.

```
export const sceneConfiguration = {
    // flag: scene is ready
    ready: false,
    // flag: initial animation
    cameraStartAnimationPlaying: false,
    // flags: start game configuration
    cameraMovingToStartPosition: false,
    boatMoving: false,
    soundEnabled: false,
    // coin and life rotation speed
    rotationStep: 10,
    // track player progress between levels
    data: {
        coinsCollected: 0,
        lifesCollected: 0
```

```
    },
    // level progress
    courseLength: 500,
    courseProgress: 0,
    levelOver: false,
    level: 1,
    /// level progress 0.0 to 1.0.
    coursePercentComplete: () => (sceneConfiguration.courseProgress /
sceneConfiguration.courseLength),

    /// How many 'columns' are in the scene
    columnsCount: 0,
    /// How many 'challenge rows' are in the scene
    challengeRowCount: 0,
    /// The current speed of the boat
    speed: 0.0
}
```

- A very important detail, is that the boat does not actually move, everything around the boat is the thing that is in motion. This is due to the water and sky limitations. Also, inside the animation function is finally the function calls for the logic and general gameplay loop which includes the rotation of the coins and hearts.

```
// game play logic
 if (sceneConfiguration.boatMoving) {
    // Detect if the boat has collided
    detectCollisions();
    // Move the columns towards the boat z position
    for (let i = 0; i < environmentBits.length; i++) {
        let mesh = environmentBits[i];
        mesh.position.z += sceneConfiguration.speed;
    }
    // Move the challenge rows towards the boat z position
    for (let i = 0; i < challengeRows.length; i++) {
        let mesh = challengeRows[i];
        mesh.rowParent.position.z += sceneConfiguration.speed;
        // Rotate the coins and lifes around thier y axis
        let children = mesh.rowParent.children;
        for (let child in children){
            if (mesh.rowParent.children[child].name == 'coin'){
            mesh.rowParent.children[child].rotateOnAxis(new Vector3(0, 0, 1),
0.01);
            }else if (mesh.rowParent.children[child].name == 'life'){
                mesh.rowParent.children[child].rotateOnAxis(new Vector3(0, 0,
1), 0.01);
```

```
            }
        }
    }

    // creating new columns on the horizon
    if ((!environmentBits.length || environmentBits[0].position.z > -1300)
&& !sceneConfiguration.levelOver) {
        addBackgroundBit(sceneConfiguration.columnsCount++, true);
    }

    // creating new challenge row on the horizon
    if ((!challengeRows.length || challengeRows[0].rowParent.position.z > -
1300) && !sceneConfiguration.levelOver) {
        addChallengeRow(sceneConfiguration.challengeRowCount++, true);
    }

    // If the starter bay hasn't already been removed from the scene, move it
away from the player
    if (starterBay != null) {
        starterBay.position.z += sceneConfiguration.speed;
    }

    // If the starter bay is outside of the players' field of view, remove it
from the scene
    if (starterBay.position.z > 200) {
        scene.remove(starterBay);
    }
}
```

- A neat little feature we added was the ability for window resize. No matter how big you make the window, the game will still be centered no matter what using the following code.

```
// resize window
window.addEventListener('resize', onWindowResize, false);
function onWindowResize() {
    camera.aspect = window.innerWidth / window.innerHeight
    camera.updateProjectionMatrix()
    renderer.setSize(window.innerWidth, window.innerHeight)
    updateWaterMaterial()
}
```

- We then added the constraints for the boat using the CLAMP

```
// boat left and right constraints
const clamp = (num: number, min: number, max: number) => Math.min(Math.max(num, min), max);
```

- Animation function to check the scene configuration to activate the appropriate animation.
    - For example, The camera setup for rotation at the beginning of the game. Where after you press the start button, the camera rotates to view the level.

```
if (sceneConfiguration.ready) {
    // ready camera animation
    if (!sceneConfiguration.cameraStartAnimationPlaying) {
        camera.position.x = 20 * Math.cos(cameraAngleStartAnimation);
        camera.position.z = 20 * Math.sin(cameraAngleStartAnimation);
        camera.position.y = 30;
        camera.lookAt(boatModel.position);
        cameraAngleStartAnimation += 0.005;

    }
```

References

Three.js. three.js docs. (n.d.). Retrieved January 6, 2023, from https://threejs.org/docs/

Three.js webgl - shaders - ocean. (n.d.). Retrieved January 12, 2023, from https://threejs.org/examples/webgl_shaders_ocean.html