CS324 Coursework Assignment Game - Pillbug

Betty Liu 2061245

December 29, 2023

Contents

Part I
How to run code

node_modules folder is included in the submitted zip that include all the needed library base on DCS machine. By simply type in npx vite or npm run dev in terminal could automatically jump to opended browser allow you to start the game. However, if any unexpected error occurred or you need to run this project from other machine (node version: '>=18.0.0' is highly recommended), simply delete node_modules folder. Then run npm install, and npx vite again to start the this game. Online demo is available from pillbug

Part II
Basic code implementation

1 Main function overview

I have split the implementation of different sections into separate functions, rather than writing them all in the main function, which has improved code readability and maintainability.

```
function main() {

    canvas = document.getElementById("gl-canvas");
    renderer = new THREE.WebGLRenderer({canvas, antialias: true});

    camera = createCamera(3);
    cameraUpper = createCamera(5);
    // set init camera
    CurrentCamera = camera;

    scene = new THREE.Scene();

    // fit screen size
    renderer.setSize(width, height);

    // Enable Shadows in the Renderer
    renderer.shadowMap.enabled = true;
    renderer.shadowMap.type = THREE.PCFSoftShadowMap;

    // set background color
    renderer.setClearColor(new THREE.Color("skyblue"));

    createLights();
    createdodecas();
    createStars();
    createTorus("red", -50, 2);
    createTorus("cyan", -20, 1);
    createTorus("yellow", -10, 0.5);

    createModel();

    // Create a plane that receives shadows (but does not cast them)
    createPlane();

    renderer.render(scene, CurrentCamera);

    // if clicked, switch camera
    window.addEventListener('click', function () {
        switchCamera();
    });

    animate();

    // resize window
    window.addEventListener('resize', onWindowResize);

    // model movement
    window.addEventListener('keydown', function (event) {
        keysPressed[event.key] = true;
    });
    window.addEventListener('keyup', function (event) {
        keysPressed[event.key] = false;
    });
}
```


Figure 1: main overview

2 Levels

There are two levels included in this game, day and night.

2.1 Model import

Model is the common part for each levels, GLTFLoader is used to load model and set it to suitable size and position by using scale method.

```
1 usage   acyanbird
function createModel() : void {
    loader.load( modelpath,  onLoad: function ( gltf ) : void  {
        model = gltf.scene;
        model.scale.set(0.3, 0.3, 0.3);
        model.position.set(0, 0.2, -3);
        gltf.scene.traverse(function(node) : void  {
            if (node.isMesh) {
                node.castShadow = true;
            }
        });
        scene.add( model );

        // add bounding box
        modelBox = new THREE.Box3().setFromObject(model);

    },  onProgress: undefined,  onError: function ( error ) : void  {
        console.error( error );
    } );
}
```

Figure 2: import model

2.2 Day level

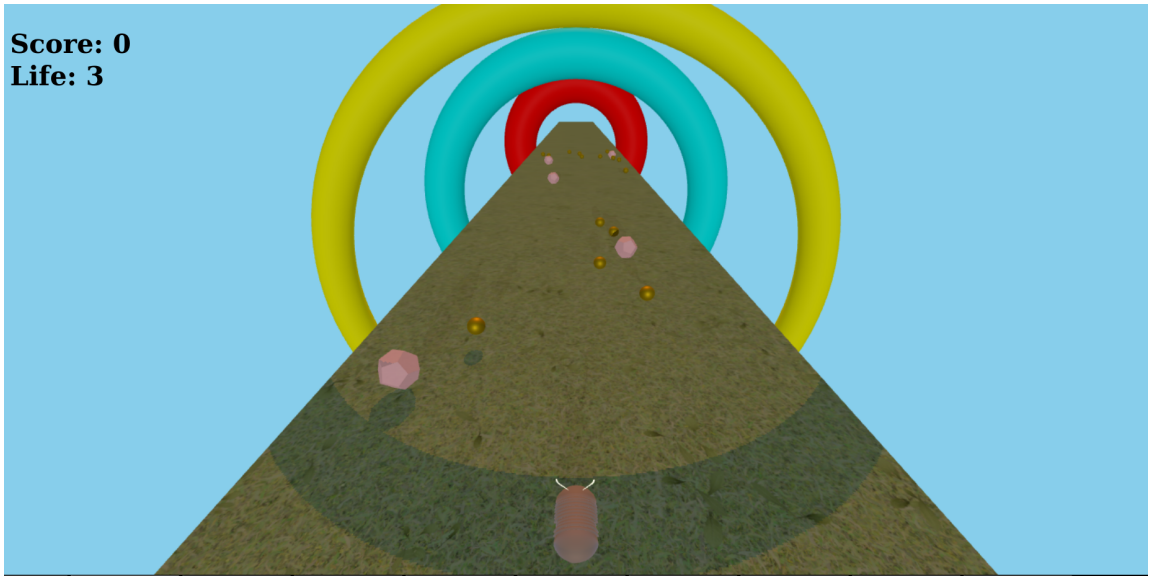

Figure 3: Day level overview

This level is made up of plane, 3 toruses, 5 DodecahedronGeometry and 15 SphereGeometry. The background color is set by renderer.setClearColor

```
// set background color
renderer.setClearColor(new THREE.Color("skyblue"));
```

Figure 4: day background color

The plane is the main playing area which is PlaneGeometry, and turn 90 degree with x-axis. Set origin to (0, 0, -30) cause by default it is set on origin, half of the plane will out of camera view and waste computing resource. Using grass picture as its texture, enable repeat wrapping to prevent image deformation.

```
1 usage   acyanbird
function createPlane() : void {
    // create texture
    let texture = new THREE.TextureLoader().load( grassimg );

    texture.wrapS = THREE.RepeatWrapping;
    texture.wrapT = THREE.RepeatWrapping;
    texture.repeat.set( 1, 12 );
    // smoother surface
    let geometry : PlaneGeometry  = new THREE.PlaneGeometry( width: 10,  height: 120,  widthSegments: 5,  heightSegments: 5);
    // self lighting red
    let material : MeshPhongMaterial  = new THREE.MeshPhongMaterial( parameters: {map: texture, side: THREE.DoubleSide});
    let plane : Mesh  = new THREE.Mesh( geometry, material );

    plane.position.set( x: 0,  y: 0,  z: -60);
    plane.rotation.x = Math.PI / 2;
    plane.receiveShadow = true;
    scene.add( plane );
}
```

Figure 5: Day plane

3 toruses is used for decoration. createTorus that accept torus Z-coordinate, color and radius of the tube to prevent long duplicate code, and use this function in main.

```
3 usages   acyanbird
function createTorus(color, z, tube) : Mesh {
    let geometry : TorusGeometry  = new THREE.TorusGeometry( radius: 7, tube,  radialSegments: 16,  tubularSegments: 100 );
    let material : MeshPhongMaterial  = new THREE.MeshPhongMaterial( parameters: {color: color} );
    let torus : Mesh  = new THREE.Mesh( geometry, material );
    torus.castShadow = true;
    torus.position.set( x: 0,  y: 0, z)
    scene.add( torus );
    return torus;
}
```

Figure 6: Torus

Two kinds of moving objects, sphere and dodecahedron are create multiple and store in array. This can be used for further random generation and rotation which will be explained in animation part. As an example of sphere creation, single sphere is create and return using one function. And the other one will use for loop to create multiple objects and store them into array.

```
function createStar() : Mesh  {
    let geometry : SphereGeometry  = new THREE.SphereGeometry( radius: 0.2,  widthSegments: 32,  heightSegments: 32);
    let material : MeshMatcapMaterial  = new THREE.MeshMatcapMaterial( parameters: {color: 0xffff00});
    star = new THREE.Mesh(geometry, material);
    star.castShadow = true;
    star.receiveShadow = true;
    scene.add(star);
    return star;
}
1 usage   acyanbird *
function createStars() : void  {
    for (let i : number  = 0; i < 15; i += 1) {
        star = createStar();
        star.position.set(randomInt(-5, 5), 0.5, randomInt(-70, -50));
        stars.push(star);
        scene.add(star);
    }
}
```

Figure 7: create sphere

2.3 Night level

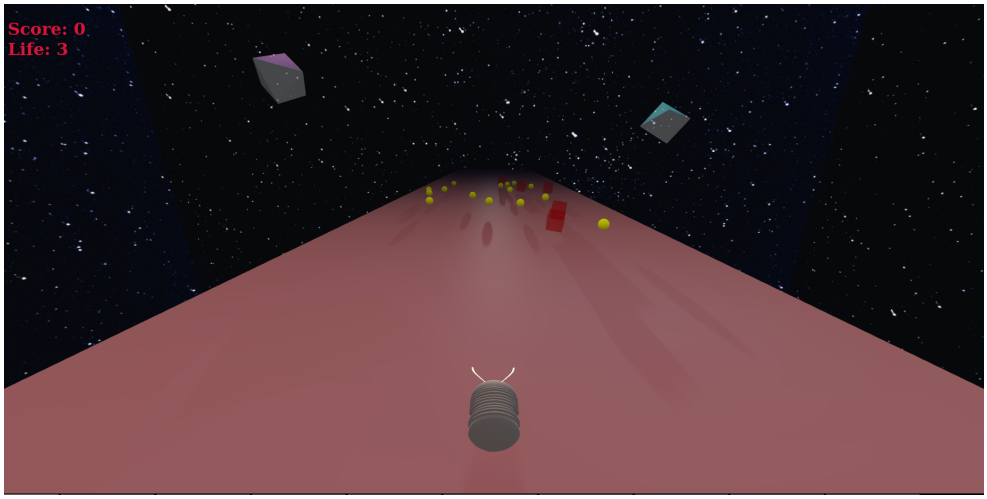

Figure 8: Night level overview

This level is made of plane, fog, two Octahedron, five BoxGeometry and fifteen SphereGeometry. Using load picture on CubeTexture as scene.background to add a skybox to the scene. FogExp2 is used to create gradient blur effect .

```
function createBackground() : void  {
    // Adds a background of stars to a skybox
    const cubeloader : CubeTextureLoader  = new THREE.CubeTextureLoader();
    let cubetexture = cubeloader.load( urls: [
        starbgsquare, // right
        starbgsquare, // left
        starbgsquare, // top
        starbgsquare, // bottom
        starbgsquare, // front
        starbgsquare  // back
    ]);
    scene.background = cubetexture;
}
```

Figure 9: night fog

Creating the plane similar to a daytime level but with the addition of a self-illumination effect to make the ground plane appear to emit a red light on its own. Additionally, the level is wider and shorter than the daytime level to create a different gaming experience.

```
function createPlane() : void  {
    // smoother surface
    let geometry : PlaneGeometry  = new THREE.PlaneGeometry( width: 13,  height: 60,  widthSegments: 2,  heightSegments: 10);
    // self lighting red
    let material : MeshPhongMaterial  = new THREE.MeshPhongMaterial( parameters: {
        color: 0x999999,
        emissive: 0xff0000,
        emissiveIntensity: 0.2,
        side: THREE.DoubleSide
    });
    let plane : Mesh  = new THREE.Mesh(geometry, material);

    plane.position.set( x: 0,  y: 0,  z: -30);
    plane.rotation.x = Math.PI / 2;
    plane.receiveShadow = true;
    scene.add(plane);
}
```

Figure 10: night plane

Two continuously rotating octahedra are used as decorations, with transparent and reflective materials applied. Additionally, two different-colored light sources are aimed at them to create a more immersive lighting effect. The

rotating process is done in animation function.

```
function createCrystal() {
    let geometry = new THREE.OctahedronGeometry(0.4);
    let material = new THREE.MeshPhysicalMaterial({
        color: "white", // Purple color
        transparent: true,
        opacity: 0.5,
        roughness: 0.3,
        reflectivity: 0.7,
        clearcoat: 0.8
    });
    crystal1 = new THREE.Mesh(geometry, material);
    crystal1.position.set(-3, 4, -5);
    crystal1.castShadow = true;
    crystal1.rotation.y = Math.PI / 4;

    crystal2 = new THREE.Mesh(geometry, material);
    crystal2.position.set(3, 3.5, -6);
    crystal2.castShadow = true;
    crystal2.rotation.y = Math.PI / 4;

    scene.add(crystal1, crystal2);

    let spotlight1 = new THREE.SpotLight(0x5e2e68, 18);
    spotlight1.position.set(-3, 5, -5);
    spotlight1.target = crystal1;
    spotlight1.castShadow = true;

    let spotlight2 = new THREE.SpotLight(0x1e8fac, 18);
    spotlight2.position.set(3, 4.5, -6);
    spotlight2.target = crystal2;
    scene.add(spotlight1, spotlight2);
}
```

Figure 11: night decoration

BoxGeometry and SphereGeometry part is similar to day level except for

changing geometry and material.

3 Light Sources

The light sources is different between day and night level:

3.1 Day level

There are one amibient light and two diretional light provided here. The amibient light globally illuminates all objects in the scene equally. And two directional light: the orange light from top of scene imulates sunlight, while the white light from back of scene provides a glossy appearance to the sphere.

```
function createLights() : void  {
    ambientLight = new THREE.AmbientLight( color: 0xffffff,  intensity: 0.8);
    scene.add(ambientLight);

    const directionalLight : DirectionalLight  = new THREE.DirectionalLight( color: 0xffffff,  intensity: 0.8);
    directionalLight.position.set( x: 0,  y: 3,  z: 20);
    directionalLight.castShadow = true;
    scene.add(directionalLight);

    const directionalLight2 : DirectionalLight  = new THREE.DirectionalLight( color: "orange",  intensity: 1);
    directionalLight2.position.set( x: 0,  y: 20,  z: -20);
    directionalLight2.castShadow = true;
    scene.add(directionalLight2);

}
```

Figure 12: day light

3.2 Night level

There are one amibient light, one diretional light and two spotlight. The directional light comes from back of the scene to create shadows. Two spotlight is set at the top of each Octahedron and point to them using target method.

```
function createLights() : void  {
    ambientLight = new THREE.AmbientLight( color: 0xffffff,  intensity: 0.8);
    scene.add(ambientLight);

    directionalLight = new THREE.DirectionalLight( color: 0xffffff,  intensity: 1.5);
    directionalLight.position.set( x: 0,  y: 5,  z: -50);
    directionalLight.castShadow = true;
    scene.add(directionalLight);

}
```

```
let spotlight1 : SpotLight  = new THREE.SpotLight( color: 0x5e2e68,  intensity: 18);
spotlight1.position.set( x: -3,  y: 5,  z: -5);
spotlight1.target = crystal1;
spotlight1.castShadow = true;

let spotlight2 : SpotLight  = new THREE.SpotLight( color: 0x1e8fac,  intensity: 18);
spotlight2.position.set( x: 3,  y: 4.5,  z: -6);
spotlight2.target = crystal2;
scene.add(spotlight1, spotlight2);
```

Figure 13: night light

4 Menu

A seperated html file is used as main menu. It contains basic instruction of game, and link to different levels:

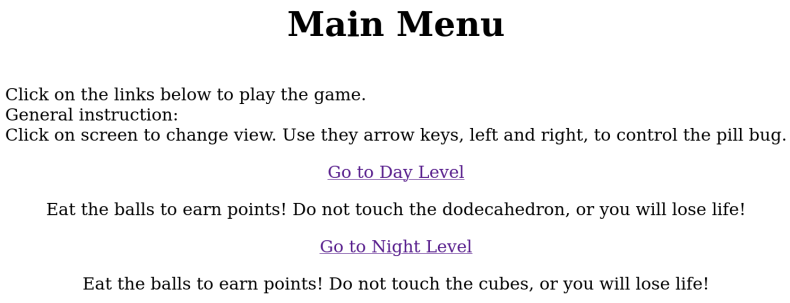

Figure 14: main menu overview

5 Camera

I use two camera to provide different view, player can change the view by click on the screen. The implementation of two cameras are same in both levels. Two PerspectiveCamera is set at (0, 0, 3) and (0, 0, 5), they both look at negative z axis.

```
function createCamera(y) : PerspectiveCamera  {
    let newcamera : PerspectiveCamera  = new THREE.PerspectiveCamera( fov: 75, aspect,  near: 0.1,  far: 1000)
    newcamera.position.y = y;
    newcamera.lookAt(new THREE.Vector3( x: 0,  y: 0,  z: -10));
    return newcamera;
}
```

Figure 15: camera creation

Changing camera is done by monitor users' clicking, and render the scene again using other camera.

```
1 usage   acyanbird
function switchCamera() : void  {
    if (CurrentCamera === camera) {
        CurrentCamera = cameraUpper;
    } else {
        CurrentCamera = camera;
    }
    renderer.render(scene, CurrentCamera);
}
```

Figure 16: cmera change

Part III
Bonus code implementation

6 Animation

This part is implement base on requestAnimationFrame. It is similar in two levels so I will introduce them together

6.1 Rotation

Many objects like Dodecahedron, Sphere are rotating during game. This is achieved by adding a rotation angle of 0.01 to both the x and y axes every time a new frame is rendered. Here's an example of the night level, where all rotations are implemented in the same way.

```
// rotate crystal
crystal1.rotation.x += 0.01;
crystal1.rotation.y += 0.01;

crystal2.rotation.x += 0.01;
crystal2.rotation.y += 0.01;
```

Figure 17: rotation

6.2 Obstacle movement and random generation

The gain point and lose life object is moving in a constant speed, their z position is updated in each frame by iterating through each element in the stored array. When it pass origin or collide with model, a random number genration is used to generate the random position. Obstacles are only visible within certain position range.

```
cubes.forEach(cube => {
    // move cube
    cube.position.z += speed;

    // set cube visible range
    cube.visible = cube.position.z < 0 && cube.position.z > -60;

    // reset cube position
    if (cube.position.z > 0) {
        // cube.visible = true;
        cube.position.x = randomNum(-5, 5);
        cube.position.z += randomNum(-90, -40);
    }
```

Figure 18: sample of movement and random generation

6.3 Moving model

Moving the model by detecting user keyboard input. When the user continuously presses the left or right arrow keys, it moves the model along the x-axis. Additionally, limiting the model's movement range to ensure it stays on the plane.

```
// model movement
window.addEventListener( type: 'keydown', listener: function (event : KeyboardEvent ) : void {
    keysPressed[event.key] = true;
});
window.addEventListener( type: 'keyup', listener: function (event : KeyboardEvent ) : void {
    keysPressed[event.key] = false;
});

if (keysPressed['ArrowLeft'] && model.position.x > -3.5) {
    model.position.x -= 0.05
}
if (keysPressed['ArrowRight'] && model.position.x < 3.5) {
    model.position.x += 0.05
}
```

Figure 19: model moving

6.4 Game over detection

Keep tracking on life, and when the life equal to 0, end the game and stop rendering. Also jump out final score by updating html line also change the block to visible to show the final result and let user get back to main menu.

```
if (!gameend) {
    requestAnimationFrame(animate);
}

gameend = true;
document.getElementById( elementId: "finalScore").innerHTML = "Final Score: " + score;
document.getElementById( elementId: "end").style.display = "block";
```

Figure 20: game over

7 Collision detection

I used bounding boxes as the collision detection method. When importing a model, a Box3 object is created as its bounding box, which is updated based on the model's current position. Depending on the shape of the obstacles, I used either a box or a sphere as the collision bounding object, I slightly adjust the sphere radius for better representation. During each frame update, model's bounding box is recalculated base on current position. On each loop, box or sphere are generated for the obstacles. Then checke whether they intersected. If they did, a collision occurred. This resulted in resetting the obstacle's position, reducing lives or increasing the score, and playing sound effects.

```
// detect collision
let starSphere : Sphere  = new THREE.Sphere(star.position,  radius: 0.18);
if (modelBox && modelBox.intersectsSphere(starSphere)) {
    // reset star position
    star.position.x = randomNum(-4, 4);
    star.position.z += randomNum(-110, -60);
    score += 1;
    document.getElementById( elementId: "score").innerHTML = "Score: " + score;
    // play sound effect
    let audio : HTMLAudioElement  = new Audio(getpoint);
    audio.play();
}
```

Figure 21: collision detection

8 Sound

Sound effects are done in the collision detection part by adding audio.play() method. When user gain point, lose life or the game is over, different sound effects are played.

9 Heads-Up Display (HUD)

The HUD is implemented using an HTML element positioned in the top left corner, continuously displaying the title. When collisions occur, such as when the user scores or loses lives, it updates by rewriting HTML elements. Users can check their score and remaining lives at any time.

Part IV
Blender Model Design

This model is constructed based on real-life insects pillbug, and the render preview is as follows.

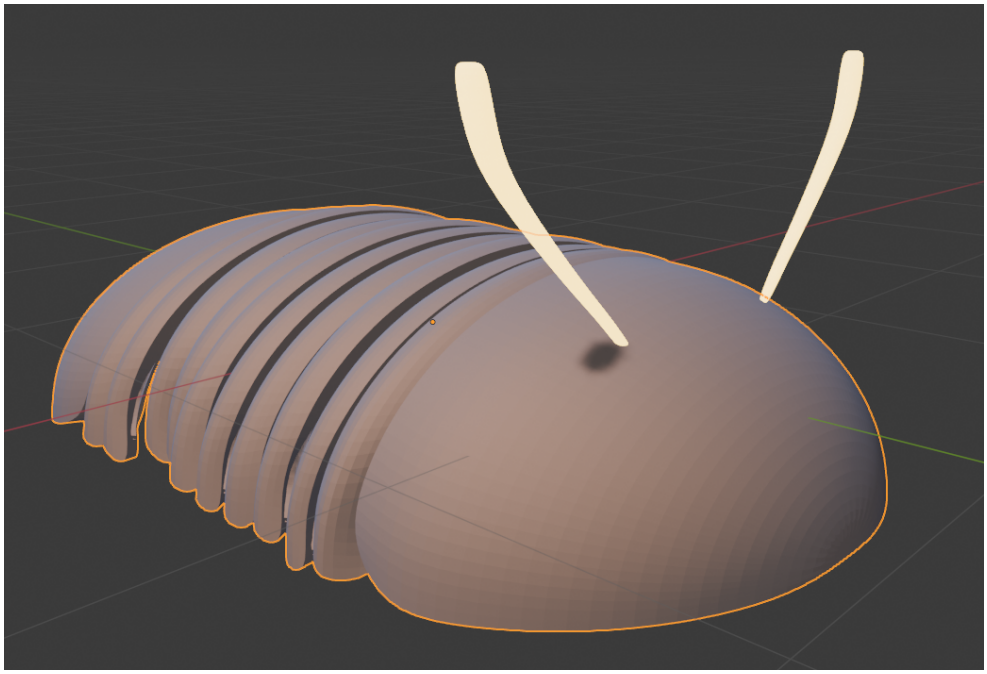

Figure 22: render preview

Create an UV sphere, select half of the vertices, then stretch it, and subsequently delete the lower half.

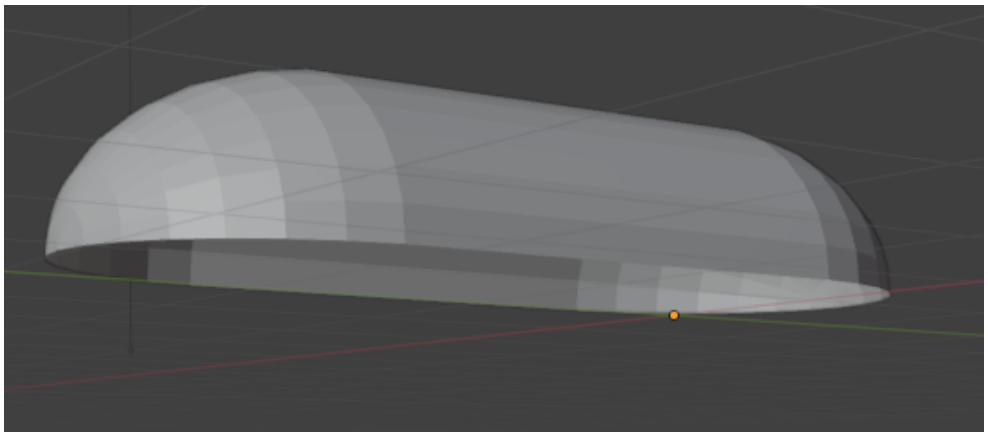

Figure 23: cut

Loop cut to form body segments, copy and paste each segment and seperate them. Then delete the original sphere.

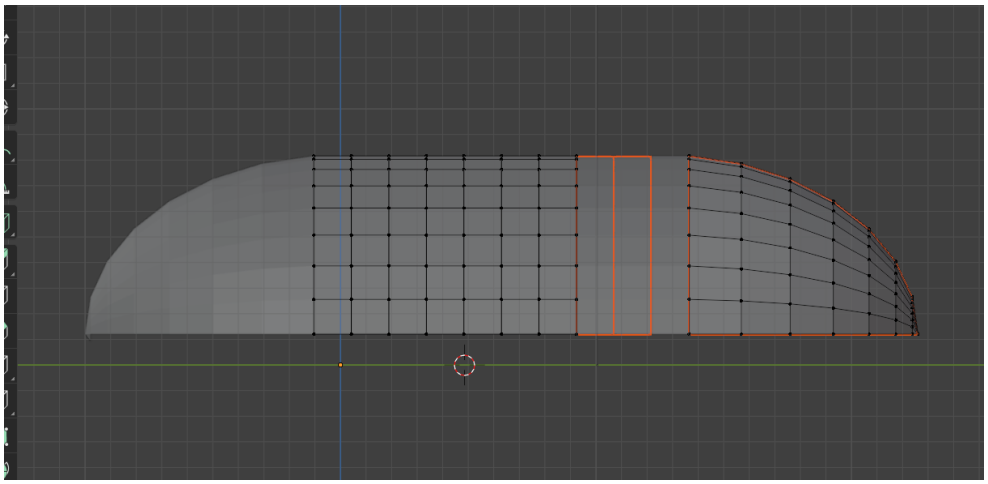

Figure 24: seperate

Continue loop cut each segment .except head and tail. Bend and deform each body segment to make them look more like “feet” by manipulating the control points.

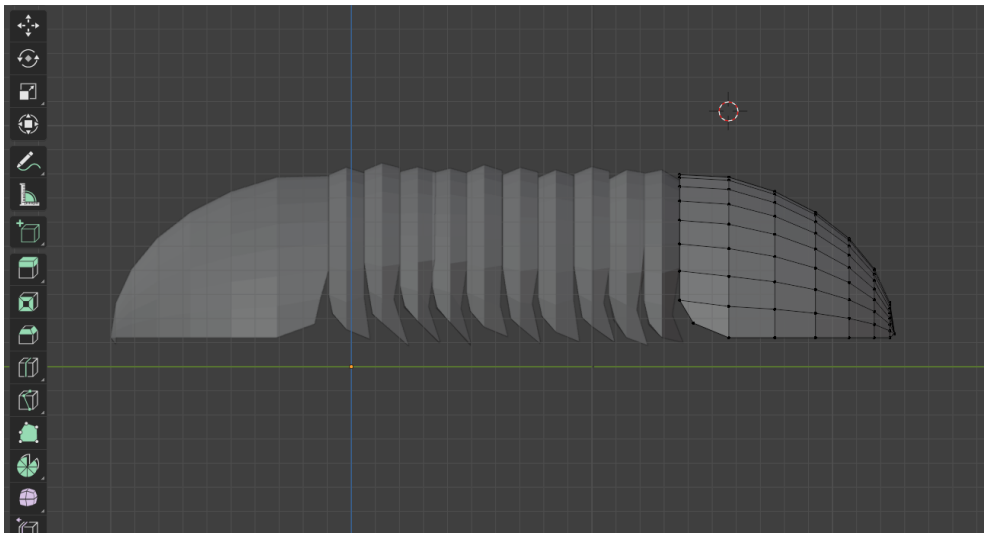

Figure 25: modify segments

Generate subdivision surface make model smoother.

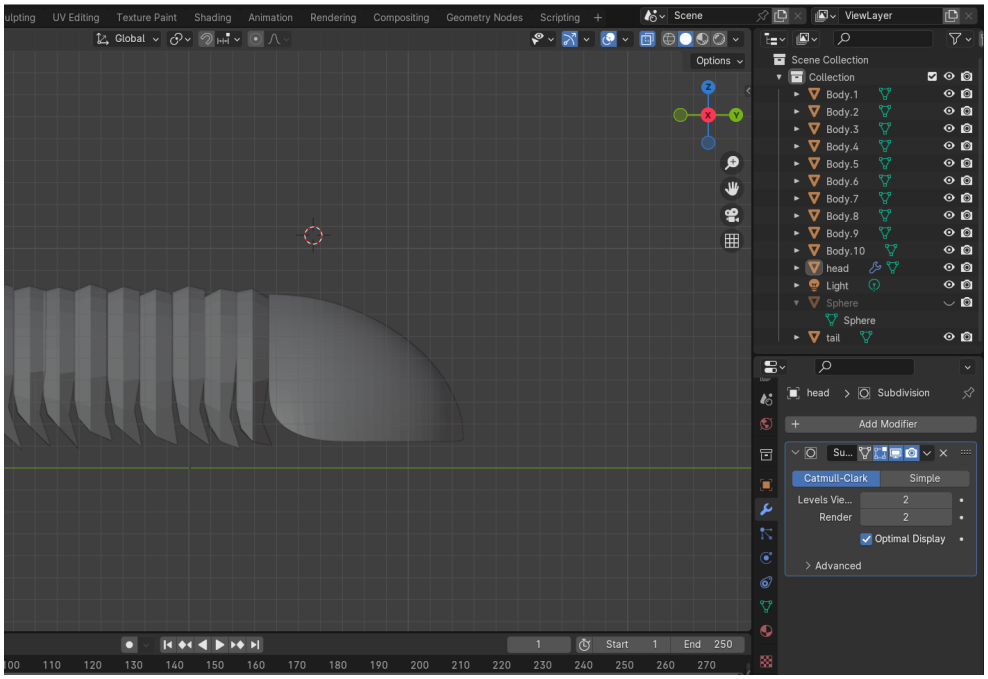

Figure 26: smoother

Then solidify, set thickness to 0.1

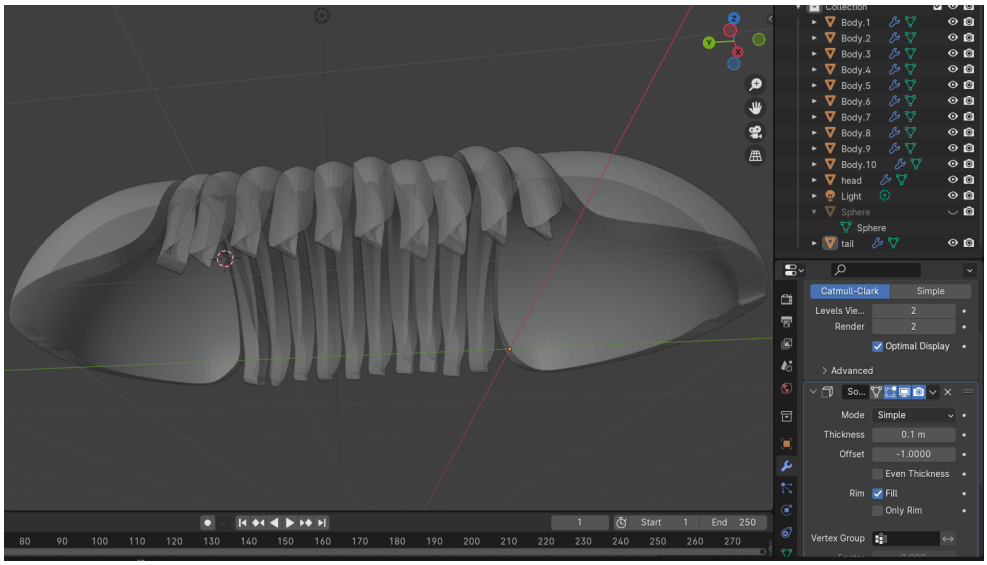

Figure 27: solidify

Union each segment and turn it into one part. Then only head segment left and it become bug itself.

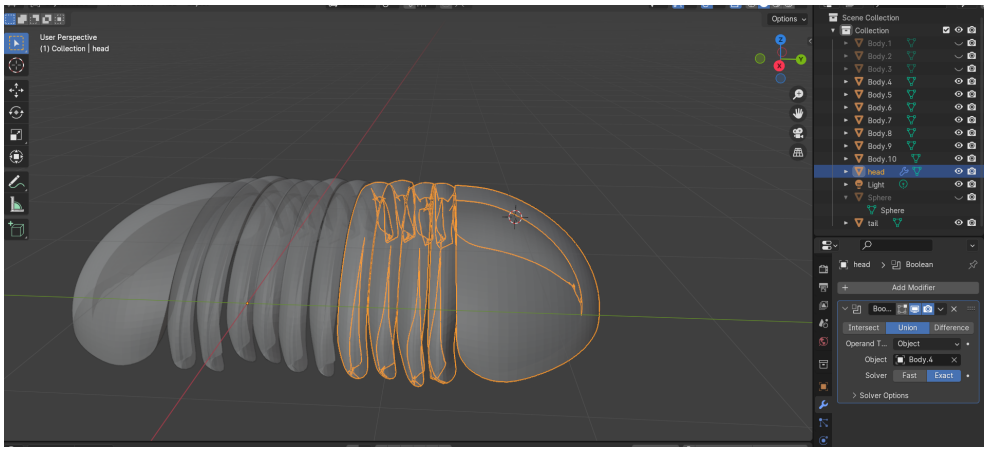

Figure 28: processing union

Because model is made of Quads and triangles only, using triangulate to split other polygon.

Making antenna by create a cylinder using triangle fan to fill. Make the

cylinder thicker at the top and thinner at the bottom by deforming the upper and lower edges.

Loop cut it, twist it by proportional editing and make it smoother.

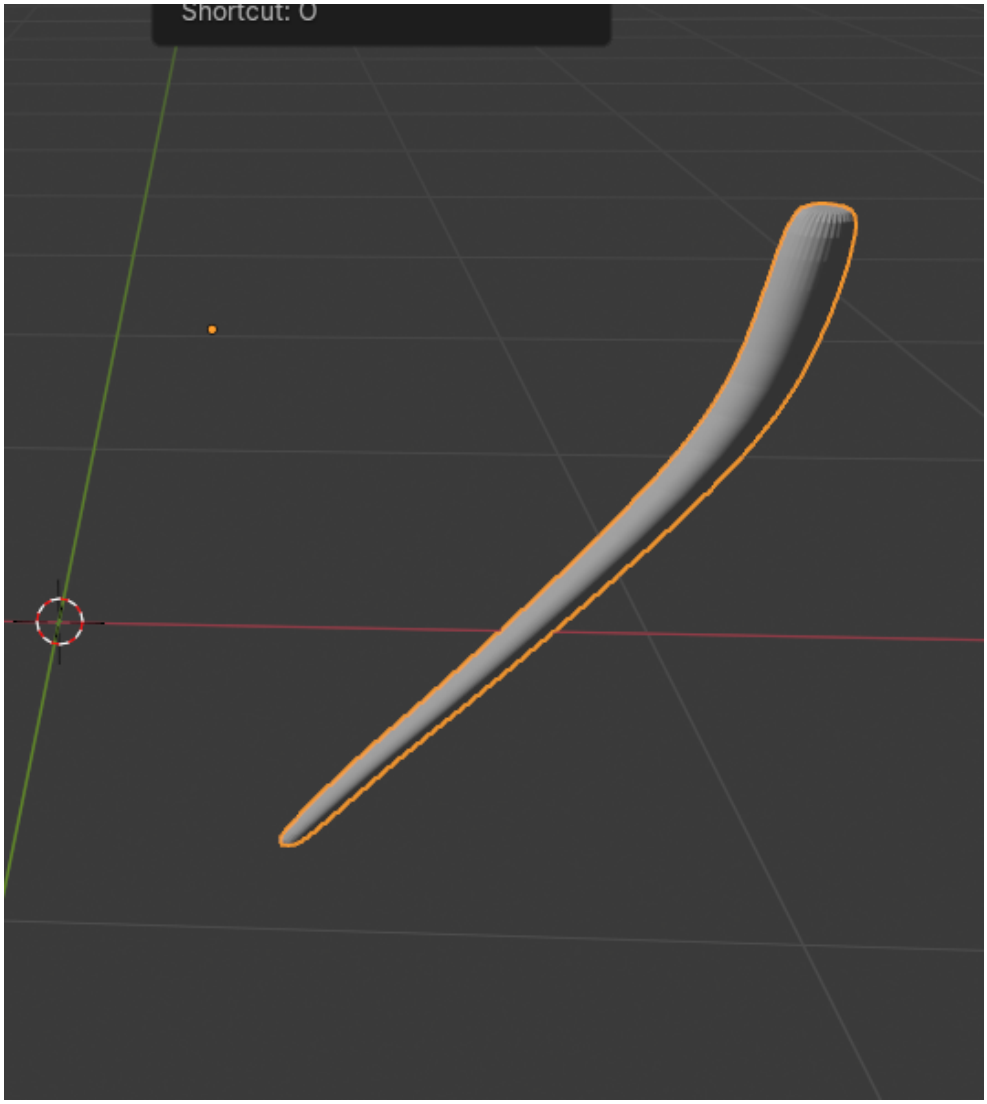

Figure 29: antenna

(sorry but I really forget to backup this part)
Copy paste and flip to get a new one, move them onto the pillbug.

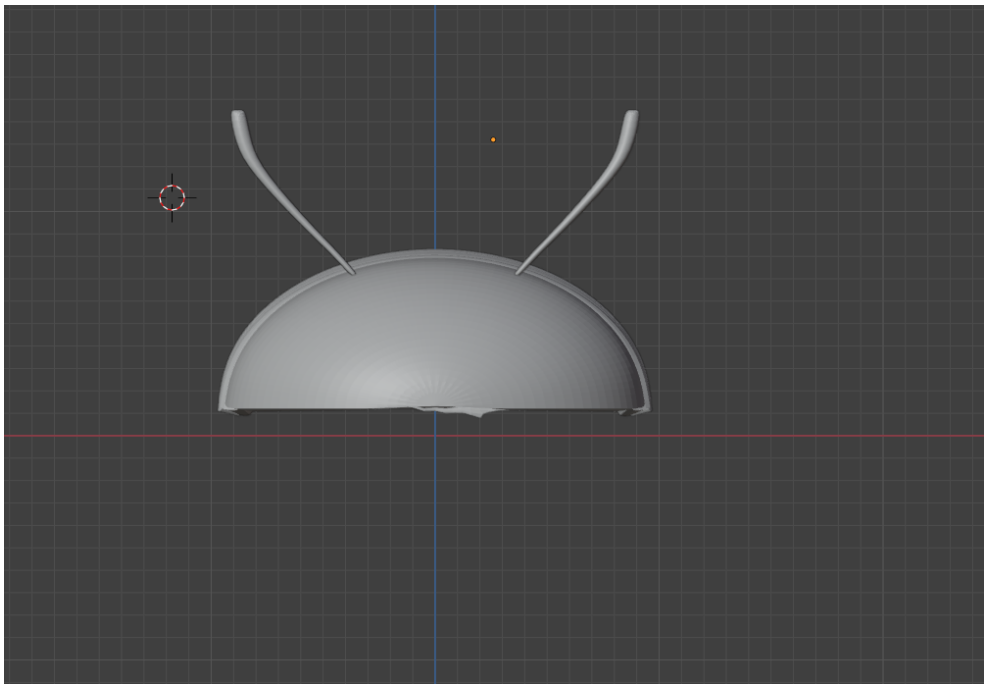

Figure 30: pillbug uncolor

Set up the material for each part. Set emmision to two antennas, metallic to the body. Then the model is done by export it to glb format.

Part V
Picture and sound material

All pictures, sound effect copyright information can be found on the asset folder.